Interactive System Productivity Facility (ISPF)

Dialog Developer's Guide and Reference

OS/390 Version 2 Release 8.0

SC28-1273-03

Interactive System Productivity Facility (ISPF)

Dialog Developer's Guide and Reference

OS/390 Version 2 Release 8.0

SC28-1273-03

Note
Before using this document, read the general information under "Notices" on page ix.

Fourth Edition (September 1999)

This edition applies to ISPF for Version 2 Release 8 of the licensed program OS/390 (program number 5647-A01) and to all subsequent releases and modifications until otherwise indicated in new editions. Make sure you are using the correct edition for the level of the product.

Order publications by phone or fax. IBM Software Manufacturing Solutions takes publication orders between 8:30 a.m. and 7:00 p.m. eastern standard time (EST). The phone number is (800) 879-2755. The fax number is (800) 284-4721.

You can also order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address below.

A form for comments appears at the back of this publication. If the form has been removed, and you have ISPF-specific comments, address your comments to:

```
International Business Machines Corporation
Attn: Information Development
Department T99 / Building 062
P.O. Box 12195
Research Triangle Park, NC 27709-2195

FAX (United States & Canada): 1+919+254-0206
FAX (Other Countries): Your International Access Code +1+919+254-0206
```

If you have OS/390 - specific comments, address them to:

```
International Business Machines Corporation
Department 55JA, Mail Station P384
522 South Road
Poughkeepsie, NY  12601-5400
United States of America

FAX (United States & Canada): 1+914+432-9405
FAX (Other Countries):  Your International Access Code +1+914+432-9405

IBMLink (United States customers only): KGNVMC(MHVRCFS)
IBM Mail Exchange: USIB6TC9 at IBMMAIL
Internet: mhvrcfs@vnet.ibm.com
```

If you would like a reply, be sure to include your name, address, telephone number, or FAX number.

Make sure to include the following in your comment or note:

```
Title and order number of this book
Page number or topic related to your comment
```

The ISPF development team maintains a site on the World-Wide Web, as does the OS/390 development team. The URLs for the sites are:

```
http://www.software.ibm.com/ad/ispf
http://www.ibm.com/s390/os390
```

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© Copyright International Business Machines Corporation 1980, 1999. All rights reserved.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Figures vii

Notices ix
Programming Interface Information. x
Trademarks x

Preface xiii
About This Book xiii
Who Should Use This Book xiii
What Is in This Book? xiii
Notation Conventions xv
Terms Used in This Book xv

Summary of Changes xvii
ISPF Product Changes xvii
ISPF DM Component Changes xvii
ISPF PDF Component Changes xviii
ISPF SCLM Component Changes xix
ISPF Client/Server Component Changes xix
ISPF User Interface Considerations xx
ISPF Migration Considerations xx
ISPF Profiles xx
Year 2000 Support for ISPF xx

What's in the OS/390 V2R8.0 ISPF library? xxiii
OS/390 V2R8.0 ISPF xxiii

Elements and Features in OS/390 . . . xxv

The ISPF User Interface xxix
Some Terms You Should Know xxix
How to Navigate in ISPF without Using Action Bars xxx
How to Navigate in ISPF Using the Action Bar Interface xxx
Action Bars xxx
Action Bar Choices xxxiii
Point-and-Shoot Text Fields xxxiv
Function Keys xxxv
Selection Fields xxxv
Command Nesting xxxvi

Chapter 1. Introduction to ISPF 1
What Is ISPF? 1
What Is a Dialog?. 1
Functions 2
Variables. 2
Command Tables 3
Panel Definitions 3
Message Definitions 3
File-tailoring Skeletons 3
Tables 3
What Does a Dialog Do? 4
Developing a Dialog. 4
How Dialog Elements Interact 5
Dialog Variables 7

Chapter 2. Controlling ISPF Sessions 9
Dialog Control and Data Flow 9
Processing a Dialog 9
Starting a Dialog 10
Syntax for Issuing the ISPSTART Command 10
Using the ISPSTART Command 18
Invoking a Dialog from a Selection Panel . . . 19
Invoking a Dialog from a Master Application Menu. 19
What the SELECT Service Does 20
Invoking the SELECT Service. 22
Terminating a Dialog 22
Return Codes from Terminating Dialogs . . . 23
An Example Using the ZISPFRC Return Code 24
ISPF Test and Trace Modes 25
Test Modes 26
ISPF Trace Modes. 27
Invoking Authorized Programs 27
Invoking TSO Commands 27
Compiled REXX Requirements 28
CLIST Requirements. 28
Restrictions to Using Attention Exits from CLISTs 28
Examples of CLIST Attention Exit Process Flow 29
Using APL2 30
Invoking APL2 30
Executing APL2 Functions. 32
Invoking ISPF Dialog Services in the APL2 Environment 32
APL2 Workspace as the ISPF Function Pool 33
Interface between ISPF and APL2 34
Subtasking Support 34
ESTAE Restrictions 34
ISPF Services in Batch Mode 35
Command Processors in the TSO Batch Environment 35
Batch Display Facility for Background Panel Processing 37
ISPF Graphical User Interface in Batch Mode 40

Chapter 3. Introduction to Writing Dialogs 43
Using the Display Services. 43
Example: Creating a Display with TBDISPL 44
Processing Selected Rows 46
Adding Table Rows Dynamically during Table Display Scrolling 47
Example: Dynamic Table Expansion 51
Using the Variable Services 62
Searching Variable Pools 63

© Copyright IBM Corp. 1980, 1999

SELECT Service and Variable Access 63
Function Pools and Dialog Functions 64
Command Procedures, Program Functions, and Function Pools. 65
Use a Variable Service to Create or Delete Defined Variables. 66
Creating Implicit Variables. 66
Naming Defined and Implicit Variables . . . 67
Sharing Variables among Dialogs 67
Saving Variables across ISPF Sessions 67
Removing Variables from the Shared or Profile Pool 68
Read-Only Profile Pool Extension Variables 68
Variables Owned by ISPF 70
Variable Formats 70
System Variables Communicate between Dialogs and ISPF 70
Using VDEFINE, VDELETE, VRESET, VCOPY, VMASK, and VREPLACE 71
Using the VGET, VPUT, and VERASE Services 72
Summary of Variable Services 72
Using the Table Services 72
Where Tables Reside. 73
Accessing Data 73
Services That Affect an Entire Table. 74
Services That Affect Table Rows 74
Protecting Table Resources. 75
Example: Create and Update a Simple Table 75
Determining Table Size 76
Example: Function Using the DISPLAY, TBGET, and TBADD Services 76
Specifying DBCS Search Argument Format for Table Services 85
Using the File-Tailoring Services. 86
Skeleton Files 86
Example of Using File-Tailoring Services . . . 87
Using the PDF Services. 89
BROWSE, EDIT, and EDREC 89
BRIF, EDIF, and EDIREC 89
Library Access Services. 90
Where to Find Examples and Listings of PDF Services 90
Using the Miscellaneous Services 91
CONTROL Service 91
GDDM Services 92
GETMSG Service 92
LIBDEF Service 92
LIST Service 92
LOG Service 92
PQUERY Service 92

Chapter 4. Common User Access (CUA) Guidelines 93

Using the Dialog Tag Language to Define Dialog Elements. 93
Keylists 93
Action Bars and Pull-Downs 94
Pop-Up Windows. 94
Moveable Pop-Ups 95
WINDOW Command 96
Manual Movement 97
Pop-Up Movement Considerations 97
Field-Level Help 97
Extended Help. 97
Keys Help 98
Reference Phrase Help 98
START Service. 100

Chapter 5. Panel Definition Statement Guide 103

Introduction to Panel Definition Sections 104
Guidelines for Formatting Panels 105
Requirements for Specifying Message and Command Line Placement. 107
Factors That Affect a Panel's Size 111
Syntax Rules and Restrictions for Panel Definition 112
Using Blanks and Comments 112
Formatting Items in Lists 113
Using Variables and Literal Expressions in Text Fields. 113
Validating DBCS Strings 115
Special Requirements for Defining Certain Panels 115
Defining Menus 115
Defining Table Display Panels 134
Formatting Panels That Contain Dynamic Areas 147
Formatting Panels That Contain a Graphic Area 153
Using DBCS-Related Variables in Panels . . . 155
Using Preprocessed Panels. 155
Restrictions for Using ISPPREP 157
Using ISPPREP with the SELECT Service . . . 157
Handling Error Conditions and Return Codes 159

Chapter 6. Panel Definition Statement Reference 161

Defining Panel Sections. 161
Defining the Action Bar Choice Section . . . 161
Defining the Action Bar Choice Initialization Section 167
Defining the Action Bar Choice Processing Section 168
Defining the Area Section 169
Defining the Attribute Section 176
Defining the Body Section 212
Defining the CCSID Section 218
Defining the END Section 219
Defining the HELP Section. 219
Defining the Initialization Section 220
Defining the LIST Section 221
Defining the Model Section 222
Defining the Panel Section. 222
Formatting Panel Definition Statements 233
The Assignment Statement. 234
The ELSE Statement 239
EXIT and GOTO Statements 241
The IF Statement 243
The PANEXIT Statement 247
The REFRESH Statement 252
The TOG Statement 253
The VEDIT Statement 254
The VER Statement 255
The VGET Statement 264

The VPUT Statement 266
Using ISPF Control Variables 266
.ALARM. 268
.ATTR and .ATTRCHAR 269
.AUTOSEL 272
.CSRPOS. 272
.CSRROW 273
.CURSOR 273
.HELP 275
.HHELP 275
.MSG 275
.NRET 276
.PFKEY 277
.RESP. 277
.TRAIL 278
.ZVARS 278

Chapter 7. ISPF Help and Tutorial Panels. 281
Processing Help 282
Help Requests from an Application Panel. . . 282
Help Available from a Help Panel 283
Ending Help 284
ISPF Default Keylist for Help Panels 284
The ISPF Tutorial Panels 285

Chapter 8. Defining Messages 291
How to Define a Message 292
Message Display Variations 296
Messages Tagged with CCSID 297
Modeless Message Pop-Ups 298
Message Pop-Up Text Formatting 298
English Rules for Message Text Formatting 299
Asian Rules for Message Text Formatting. . . 299
Substitutable Parameters in Messages 301
Syntax Rules for Consistent Message Definition 301
DBCS-Related Variables in Messages 302

Chapter 9. Defining File-Tailoring Skeletons 303
Considerations for Data Records. 303
Considerations for Control Statements 305
Sample Skeleton File. 311
DBCS-Related Variables in File Skeletons 311

Chapter 10. Extended Code Page Support 313
Translating Common Characters 313
Z Variables 313
Panels Tagged with CCSID 314
Messages Tagged with CCSID 314
GETMSG Service 314
TRANS Service 315
ISPccsid Translate Load Modules 315
ISPccsid Translate Load Module Generation Macro 315
ISPCCSID Macro 315
Description of Parameters 315
ISPccsid Translate Load Module Definition Examples 316
KANA and NOKANA Keywords 317
Character Translation 317
Supported CCSIDs 318
Base Code Pages for Terminals 320
Adding Translate Tables for Extended Code Page Support 320
Base CCSIDs 322
Extended Code Page Translate Tables Provided by ISPF 323
Example of User-Modifiable ISPF Translate Table 323

Appendix A. Character Translations for APL, TEXT, and Katakana 327

Appendix B. ISPTTDEF Specify Translate Table Set 331

Appendix C. Diagnostic Tools and Information 333
ISPF Debug Tools. 333
Diagnostic Information 333
Using the ENVIRON System Command . . . 334
ENVIRON Command Syntax and Parameter Descriptions 335
Abend Panels Provide Diagnostic Information 340
ISPF Statistics Entry in a PDS Directory . . . 343
Common Problems Using ISPF 344
Messages 344
Unexpected Output 346
Abend Codes and Information 346
Terminal I/O Error Codes 348
Register Linkage Conventions 349
Obtaining Message IDs 350
Installation, Maintenance, and Migration Diagnostics 351
Common Installation and Maintenance Problems 351
Migration from Version 2 and Version 3 to Version 4.2 352

Appendix D. Dialog Variables 355
PDF Non-Modifiable Variables 360

Appendix E. System Variables 361
Time and Date. 361
General 362
ZSCRNAME Examples 365
Terminal and Function Keys 366
Scrolling. 368
PRINTG Command 368
Table Display Service 369
LIST Service 369
LOG and LIST Data Sets 369
Dialog Error 370
Tutorial Panels. 370
Selection Panels 370

DTL Panels or Panels Containing a)PANEL Section 370

Index 371

Readers' Comments — We'd Like to Hear from You 383

Figures

1. Panel with an Action Bar Pull-Down Menu xxxi
2. Pop-Up Selected from an Action Bar Pull-Down xxxii
3. Panel with an Action Bar and Point-and-Shoot Fields xxxii
4. An Unavailable Choice on a Pull-Down xxxiv
5. Using ISPF 5
6. Typical Dialog Organization Starting with a Menu 6
7. Typical Dialog Starting with a Function 7
8. Control and Data Flow 9
9. Application Dialog Running under ISPF 10
10. ISPSTART Command 0
11. Sample Selection Panel 19
12. ISPF Master Application Menu (ISP@MSTR) 20
13. SELECT Service Used to Invoke and Process a Dialog 21
14. Sample Background ISPF Job 25
15. Sample Dialog Using System Variable ZISPFRC 25
16. MVS Batch Job 36
17. TBDISPL Panel Definition 44
18. TBDISPL Display 45
19. Panel Definition Dynamic Table Expansion 52
20. PL/I Dialog Function Example Program 53
21. Initial Display for Dynamic Table Expansion Example 59
22. Second Display for Dynamic Table Expansion Example 60
23. Third Display for Dynamic Table Expansion Example 61
24. Fourth Display for Dynamic Table Expansion Example 62
25. Control and Data Flow in a Dialog 64
26. CLIST to Create a Read-Only Extension Table 69
27. Panel Definition SER 78
28. Panel Display SER. 79
29. Panel Display SER with an ISPF-provided Message Superimposed on Line 1. 80
30. Message EMPX21 81
31. Panel Display SER—Short Form of Message EMPX210 Superimpose Line 1 81
32. Panel Display SER—Long Form of Message EMPX210 Superimposed on Line 3 82
33. Panel Definition DATA 83
34. Panel Display DATA 84
35. Sample Skeleton File 87
36. Example Panel Displaying Three Pop-Up Windows 95
37. Reference Phrase Help Example 99
38. Sample Panel Definition Format 105
39. CUA Panel Definition. 109
40. Sample CUA Panel (SAMPAN on ISPKLUP) 111
41. Example of a Menu (ISP@MSTR) 116
42. Master Application Menu Definition 122
43. Master Application Menu DTL Source 123
44. ISPF Primary Option Menu Definition 128
45. Master Application Menu DTL Source 129
46. Parts of a TBDISPL display 135
47. Table Display Panel Definition. 145
48. Table as Displayed. 146
49. Table Display Panel Definition with Several Model Lines 146
50. Table as Displayed with Several Model Lines 147
51. Panel Definition Illustrating SCROLL and EXTEND 149
52. ZGE Characteristics 151
53. Dynamic Area with Character Attributes 152
54. Panel for Specifying Preprocessed Panel Input and Output Files (ISPPREPA) 156
55. Action Bar Section Example 167
56. Invalid Scrollable Area Definition. 172
57. Valid Scrollable Area Definition 173
58. Scrollable Area Screen Display (Part 1 of 3) 175
59. Scrollable Area Screen Display (Part 2 of 3) 175
60. Scrollable Area Screen Display (Part 3 of 3) 176
61. Panel Definition Illustrating a Graphic Area 182
62. Panel Definition with Graphic Area 183
63. Definition of Panel Graphic Area with Overlapping Text Field 183
64. Attribute Section in a Panel Definition 205
65. Group Box Definition. 210
66. Sample Panel Definition 217
67. Sample Panel—When Displayed 218
68. Panel Processing 232
69. Sample Panel Definition with TRANS and TRUNC 237
70. Sample Panel Definition with IF and ELSE Statement. 241
71. Standard Parameter List Format 250
72. TOG Statement Example. 254
73. VEDIT Example 255
74. Sample Panel Definition with Verification 264
75. Sample Panel Definition with Control Variables 268
76. Example of Z Variable Place-Holders 279
77. Help Panel Flow 283
78. Sample Tutorial Hierarchy 287
79. Sample Tutorial Panel Definition (Panel B) 288
80. Sample Tutorial Panel Definition (Panel F2) 289
81. Sample Messages 292
82. Example Syntax for Defining Messages 292
83. Sample Skeleton File 311
84. Basic ISP00111 Translate Module. 316
85. ISP00222 Translate Module with Two Direct CCSID Entries 316
86. Translation to CCSID 00500 from CCSID XXXXX 321
87. Translation to CCSID XXXXX from CCSID 00500 321
88. Internal Character Representations for APL Keyboards 328

© Copyright IBM Corp. 1980, 1999

89. Internal Character Representations for Text Keyboards 329
90. ENVIRON Settings Panel (ISPENVA) 334
91. Error Recovery Panel (ISPPRS1) 341
92. Common Abend Codes (ISP93010) 342
93. Additional Diagnostic Information (ISPPRS2) 342

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non_IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504–1785, USA.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact the IBM Corporation, Mail Station P300, 522 South Road, Poughkeepsie, NY 12601–5400, USA. Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries in writing to

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OR NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be

© Copyright IBM Corp. 1980, 1999

incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

If you are viewing this information softcopy, the photographs and color illustrations may not appear. For users of BookManager Library Reader for Windows, Version 2.0.2 at 5799–pxy Service Level : S9903ENU provides the best results.

Programming Interface Information

This book primarily documents information that is NOT intended to be used as Programming Interfaces of ISPF.

This book also documents intended Programming Interfaces that allow the customer to write programs to obtain the services of ISPF. This information is identified where it occurs, either by an introductory statement to a chapter or section or by the following marking:

```
-----------Programming Interface information----------------
```

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries:

ACF/VTAM
AD/Cycle
AIX
APL2
Application Development
AS/400
BookMaster
CICS
CICS OS/2
CUA
Common User Access
CSP/AD
CSP/AE
CSP/2AD
CSP/370AD
CSP/370RS
C370
DATABASE 2
DB2
DFSMS/MVS
GDDM
IBM
IMS/ESA
MVS/DFP
MVS/ESA
MVS/XA
MVS/SP
Operating System/2
OS/2
OS/400
Personal System/2
PS/2
Presentation Manager
RACF
Repository Manager
Repository Manager/MVS
Resource Access Control Facility
SAA
Series/1
System Application Architecture
System/370
System/390
VisualAge
VM/XA
VTAM

The following terms are trademarks of other companies:

C++	American Telephone and Telegraph Company
HP	Hewlett-Packard
HP-UX	Hewlett-Packard
Java	Sun Microsystems, Inc
Microsoft	Microsoft Corporation
Novell	Novell, Inc.
WordPerfect	Corel Corporation

Windows is a trademark of Microsoft Corporation.

Preface

This book describes how to use the ISPF Dialog Manager elements from programs or command procedures.

This book is intended to help you learn about and use the ISPF product.

About This Book

This book is a guide for learning and using the Dialog Manager component of the ISPF product. It contains information on controlling ISPF sessions, defining ISPF panels, and defining messages. It is supplemented by *ISPF Examples* which contains examples of working dialogs.

The *ISPF Dialog Developer's Guide and Reference* provides:
- An introduction to ISPF basics
- Information on running ISPF sessions
- Guideline information for:
  - Writing panel definitions
  - Defining messages
  - Defining file-tailoring skeletons
- A system variable summary.

Who Should Use This Book

This book is for programmers who develop ISPF application dialogs, system analysts, and system programmers.

Users are expected to know at least one of the ISPF-supported programming or command procedure languages: Assembler, PL/I, COBOL, VS FORTRAN, C, Pascal, APL2, CLIST, and REXX.

Users also should be familiar with the MVS operating system.

What Is in This Book?

Chapter 1. Introduction to ISPF, describes what ISPF is and what it does for you. In previous releases of the product, the information in this chapter was contained in *ISPF Dialog Management Guide and Reference*

Chapter 2. Controlling ISPF Sessions, describes how to start and stop an ISPF session and how to use many of the ISPF facilities. In previous releases of the product, the information in this chapter was contained in *ISPF Dialog Management Guide and Reference*

Chapter 3. Introduction to Writing Dialogs, provides introductory information on how to write dialogs using the ISPF services for display, variable, table, file tailoring, and PDF. In previous releases of the product, the information in this chapter was contained in *ISPF Services Guide*

© Copyright IBM Corp. 1980, 1999

Chapter 4. Common User Access (CUA) Guidelines, describes how ISPF supports the Common User Access (CUA) guidelines. In previous releases of the product, the information in this chapter was contained in *ISPF Dialog Management Guide and Reference*

Chapter 5. Panel Definition Statement Guide, provides guide-type information for sections, panel definition statements, and control variables. It explains how to create panels using the panel definition statements. In previous releases of the product, the information in this chapter was contained in *ISPF Dialog Management Guide and Reference*

Chapter 6. Panel Definition Statement Reference, provides reference information on how to create ISPF panels using Dialog Tag Language (DTL) and the ISPF DTL conversion utility, DTL and panel definition statements, or panel definition statements. In previous releases of the product, the information in this chapter was contained in *ISPF Dialog Management Guide and Reference*

Chapter 7. ISPF Help and Tutorial Panels, describes online help and tutorial panels that a developer can include to provide online information for an application user. In previous releases of the product, the information in this chapter was contained in *ISPF Dialog Management Guide and Reference*

Chapter 8. Defining Messages, describes how to create and change ISPF messages using an existing message definition or the MSG and MSGMBR tags of the DTL. In previous releases of the product, the information in this chapter was contained in *ISPF Dialog Management Guide and Reference*

Chapter 9. Defining File-Tailoring Skeletons, describes ISPF skeleton definitions and how to create or change skeletons. In previous releases of the product, the information in this chapter was contained in *ISPF Dialog Management Guide and Reference*

Chapter 10. Extended Code Page Support, describes how extended code page support allows panels, messages, and variable application data to be displayed correctly on terminals using any of the supported code pages. In previous releases of the product, the information in this chapter was contained in *ISPF Dialog Management Guide and Reference*

Appendix A. Character Translations for APL, TEXT, and Katakana, contains the character translation tables for APL, TEXT, and Katakana. In previous releases of the product, the information in this appendix was contained in *ISPF Dialog Management Guide and Reference*

Appendix B. ISPTTDEF Specify Translate Table Set, contains a program, ISPTTDEF, that can be used for specifying the set of terminal translate tables to be used. In previous releases of the product, the information in this appendix was contained in *ISPF Dialog Management Guide and Reference*

Appendix C. Diagnostic Tools and Information, contains information to help you diagnose ISPF problems. In previous releases of the product, the information in this appendix was contained in *ISPF Dialog Management Guide and Reference*

Appendix D. Dialog Variables, describes the ISPF dialog function pool variables that are both read from and written to by several of the PDF library access services. In previous releases of the product, the information in this appendix was contained in *ISPF Dialog Management Guide and Reference*

Appendix E. System Variables, describes system variables with type and pool information. The variables are also discussed with the ISPF service to which they apply. In previous releases of the product, the information in this appendix was contained in *ISPF/PDF Guide and Reference*

Notation Conventions

This book uses the following notation conventions:

- Uppercase commands and their uppercase parameters to show required entry
- Lowercase characters to show parameters that can be specified by the user
- Brackets [] to show optional parameters (required parameters do not have brackets)
- An OR (|) symbol to show two or more parameters you *must* select from
- Stacked parameters to show two or more parameters you *can* select from

  Note: You can choose one or none. If you choose none, ISPF uses the underscored parameter.
- Braces {} with stacked parameters to show that you *must* select one. For example:

  ```
  {KEYWORD1(variable) [OPTPAR1(variable)]}
  {KEYWORD2(variable)}
  {KEYWORD3(variable) [OPTPAR2(variable)]}
  {KEYWORD4(variable) [OPTPAR3(variable)]}
  ```

  indicates that you must select either KEYWORD1, KEYWORD2, KEYWORD3, or KEYWORD4.
- Underscores to show defaults
- Ellipsis (...) to show that the parameter can be repeated, specifying additional items of the same category.

Terms Used in This Book

In this book, there are terms that might have a different meaning or might not be clear to you. The following list contains those terms and their definition:

Terminal. Any of the supported display devices

Library. A partitioned data set

Data set. A sequential or partitioned data set

Command procedure. A CLIST or REXX EXEC

Menu. Selection panel.

Summary of Changes

OS/390 V2R8.0 ISPF contains the following changes and enhancements:
- ISPF Product and Library Changes
- ISPF Dialog Manager Component Changes
- ISPF PDF Component Changes
- ISPF SCLM Component Changes
- ISPF Client/Server Component Changes

ISPF Product Changes

Changes to the ZENVIR variable. Characters 1 through 8 contain the product name and sequence number in the format *ISPF x.y*, where x.y indicates:
- <= 4.2 means the version.release of ISPF
- = 4.3 means ISPF for OS/390 release 2
- = 4.4 means ISPF 4.2.1 and ISPF for OS/390 release 3
- = 4.5 means ISPF for OS/390 Version 2 Release 5.0
- = 4.8 means ISPF for OS/390 Version 2 Release 8.0

The ZENVIR variable is used by IBM personnel for internal purposes. The x.y numbers DO NOT directly correlate to an ISPF release number in all cases. For example, as shown above, a ZENVIR value of 4.3 DOES NOT mean ISPF Version 4 Release 3. NO stand-alone version of ISPF exists above ISPF Version 4 Release 2 Modification 1.

The ZOS390RL variable contains the OS/390 release on your system.

The ZISPFOS system variable contains the level of ISPF code that is running as part of the OS/390 release on your system. This might or might not match ZOS390RL. For this release, the variable contains **ISPF for OS/390 Version 2 Release 8.0**.

The ZPFKEY system variable contains PFKey values.

Support of CCSIDs 1140 through 1149 was added so the EURO currency symbol can be used in ISPF.

The Samples and Macros libraries each have an index member called @INDEX.

ISPF DM Component Changes

The DM component of ISPF includes the following new functions and enhancements:
- Support added for "VER(&variable, DSNAMEQ)".
- Support added for four-digit year on TBSTATS command and Option 7.4.
- Support added for four-digit year on TBSORT command.
- Message numbers were added to ISPF Line Mode messages.
- Support added for date format in Configuration table.
- Support added for a print utility exit for ISPF termination and Log/List commands.
- Support added for NEXT and PREV commands for Dialog Test Tables.

© Copyright IBM Corp. 1980, 1999

- The ZSCRNAME modifiable shared variable, containing the current screen name, was added.
- The ZPFKEY variable, which returns the name of the PFKey on exit from a panel, was added.
- The TPUT buffer size variable was added to the configuration table.
- The ISPSPROF system variables were moved to the configuration table.
- Support was added for commands chained after the START command.
- ISPDTLC enhancements:
  - New invocation options: NOPLEB / PLEB, NOMCOMMENT / MCOMMENT.
  - New tags: ATTENTION.
  - Added "German to Swiss German" character transform to create Swiss German panels from German DTL source files.
  - Expanded number of DTL source files for interactive processing from 4 to 12.
  - Allow DTL type comments (<: -- or <!--) in MVS profile data stream.
  - Allow SOURCE tag within ABC, AREA, PANEL, and REGION tags.
  - Added "Options" to action bar of interactive panel as an alternate method of setting conversion options.
  - Added support for macro tags.

ISPF PDF Component Changes

The ISPF PDF component contains the following new functions and enhancements:

- Conversion of the ISPF configuration table to a keyword format.
- Introduction of an interactive tool to create and update the keyword file and to build the needed load module from the keyword file.
- Edit CUT and PASTE commands. Data is saved in data spaces with multiple clipboards available.
- The Edit MOVE, COPY, CREATE, and REPLACE commands now accept a data set name or data set name and member name as a parameter.
- A VSAM editor or browser can be specified in the configuration table and is invoked automatically when a VSAM data set it specified in Options 1, 2, 3, or 11.
- Through the EPDF command, Edit, View, and Browse functions are available from any command line.
- A new edit macro VOLUME command to retrieve the volume of the data set being edited.
- The edit macro RECFM has been enhanced to return the full record format rather than just F or V.
- The new edit macro SESSION command returns EDIT, EDIF, or VIEW, as well as indicating whether the session was initiated from SCLM.
- Edit STATS mode is no longer forced to OFF if a sequential data set is edited. The STATS mode is simply ignored.
- The target data set for the Move/Copy utility or the Edit CREATE and REPLACE commands can be allocated automatically if it does not exist.
- Edit highlighting of FIND strings and the cursor phrase is enabled for data wider than 256 bytes.
- When the View REPLACE command is used to update the member being viewed, the confirmation panel shows whether the member has been updated by someone else during the View session.

- LMMDISP, LMMFIND, and LMMLIST return individual variables for load module statistics when STATS(YES) is specified.
- The COBOL options and PL/I options in Foreground and Batch have been consolidated into 1 COBOL option and 1 PL/I option.
- Data set information processes multivolume data sets with more than 20 volumes.
- Member list REFRESH command added.
- Member list SORT and LOCATE honor the collating sequence table in the PDF translate tables.
- The creation date is displayed on the Confirm Delete panel for VSAM data sets.
- AIX paths are identified in Option 3.4 with *PATH* in the volume column.
- REFLIST function improved.
- A warning message displays on the first data change made while in View.
- RIGHT and LEFT commands are supported in member list to enable the presentation of additional data such as full 4–character year dates.
- The LMMDISP service allows the selection of members that do not exist in the data set being processed.
- SuperC supports VSAM files.
- SuperC supports the FMSTOP performance option which stops on the first mismatch for file compare. FMSTOP is also supported for string searches.

ISPF SCLM Component Changes

The ISPF SCLM component contains the following new functions and enhancements:

- When a language that is not valid is specified on the SPROF panel, a scrollable table display of the valid languages is presented so the user can choose the desired language.
- ISPLNK is a valid CALLMETH for non-BUILD translators.
- A sample DTL parser and translator are available.
- Additional samples added to SAMPLIB.
- An SCLM EDIT service was added to enable editing of SCLM controlled parts from a dialog.
- The data set name and member is added to the SPROF panel (FLMEINFO) to assist users when selecting a language for a part.
- The ability to specify that SCLM temporary load libraries should be allocated as PDSEs has been added.
- Two new exit points added.
- SCLM warning messages issued by the ISPF editor when an SCLM controlled member is edited are only issued if the member's directory entry indicates the member is SCLM controlled.
- SCLM Versioning can be directed to ignore sequence number differences.

ISPF Client/Server Component Changes

The ISPF Client/Server Component enables a panel to be displayed unchanged (except for panels with graphic areas) at a workstation using the native display function of the operating system of the workstation. ISPF manuals call this "running in GUI mode."

The ISPF Client/Server component changes are:

- Support added to provide for an automatic download of the Client/Server component
- Enhanced usability and function of the Client/Server component download panel
- Support added to enable initiation of a workstation connection (without GUI display) while in split screen mode.
- New WSCON and WSDISCON commands to improve entry to the ISPF C/S interface.
- Enable one or more ISPF screens to **Switch** back and forth between GUI and 3270 modes by using the new Switch commands.

ISPF User Interface Considerations

Many changes have been made to the ISPF Version 4 user interface to conform to CUA guidelines. If you prefer to change the interface to look and act more like the Version 3 interface, you can do the following:

- Use the CUAATR command to change the screen colors
- Use the ISPF Settings panel to specify that the TAB or HOME keys position the cursor to the command line rather than to the first action bar item
- Set the command line to the top of the screen by deselecting ***Command line at bottom*** on the ISPF Settings panel
- Set the primary keys to f13–24 by selecting 2 for Primary range on the Tailor Function Key Definition Display panel
- Use the KEYLIST OFF command to turn keylists off
- Use the PSCOLOR command to change point-and-shoot fields to blue.
- Change the DFLTCOLR field in the PDF configuration table ISRCONFG to disable action bars and or edit highlighting

ISPF Migration Considerations

When migrating to OS/390 V2R8.0 or higher for the first time, you must convert your ISPF customization to the new format. Refer to the section entitled ***The ISPF Configuration Table*** in the *ISPF Planning and Customizing manual.*

When migrating from one version of ISPF to another, you must be sure to reassemble and re-link the SCLM project definition.

ISPF Profiles

Major changes have been made to the ISPF profiles for ISPF Version 4.2 and OS/390 V2R8.0 ISPF. If you are moving back and forth between a Version 3.3 or Version 3.5 system and a Version 4.2 or an OS/390 V2R8.0 system, you must run with separate profiles.

Year 2000 Support for ISPF

ISPF is fully capable of using dates for the year 2000 and beyond. All of your existing applications should continue to run (some may need minor changes, as explained below) when the year 2000 comes. The base support for the year 2000 was added to OS/390 Version 1 Release 2.0, but the same level of support is

available for ISPF Version 3.5, ISPF Version 4, and OS/390 Version 1 Release 1.0 as well. To get support for the earlier versions, be sure that your system has the correct APARs installed. All ISPF APARs that add or correct function relating to the year 2000 contain the YR2000 identifier in the APAR text. You should search for these APARs to ensure you have all the function available.

What function is included?

- ISPF Dialog variable ZSTDYEAR now correctly shows the year for dates past 1999. Earlier versions always showed the first 2 characters of the year as 19.
- A new ISPF dialog variable (ZJ4DATE) is available for Julian dates with a 4–digit year.
- An ISPF Configuration Table field enables PDF to interpret 2 character year dates as either a 19xx or 20xx date. The default value is 65. Any 2-character year date whose year is less than or equal to this value is considered a 20xx date, anything greater than this value is considered 19xx. To see what value has been set by the ISPF Configuration Table, use the new ZSWIND variable.
- New parameters in the LMMSTATS service (CREATED4 and MODDATE4) for specifying 4-character year dates. All existing parameters still exist and you can continue to use them. If both the 2-character year date parameters (CREATED and MODDATE) and the 4-character year date parameters (CREATED4 and MODDATE4) are specified, the 2-character versions are used.
- Dialog variables ZLC4DATE and ZLM4DATE have been added.
  - You ***can*** set them before making an LMMREP or LMMADD call. Do this to specify a 4-character *created* or *last modified* date to set in the ISPF statistics.
  - They ***are*** set by LMMFIND, LMMLIST and LMMDISP to the current value of the created and last modified dates in the ISPF statistics.

What might need to change? Some minor changes to your existing ISPF dialogs might be necessary, especially in ISPF dialogs that use the Library Access Services to manipulate ISPF member statistics.

- For those services that accept both 4-character year dates and 2-character year dates, you can specify one or the other. If you specify both, the 2-character year date is used to avoid affecting existing dialogs. When the 2-character year date is used, the configuration table field mentioned above is used to determine whether the date should be interpreted as 19xx or 20xx.
- ISPF will not necessarily show 4-character dates in all circumstances but it will process them correctly. For example, a member list might only display 2-character year dates but will sort those dates in the proper order.
- SCLM stores dates past the year 1999 in a new internal format. If an accounting file contains dates in this new format, it cannot be processed by a system without year 2000 support. Accounting files without dates past 1999 can be processed with or without the year 2000 support.
- No conversion of the LMF control file is necessary.

What's in the OS/390 V2R8.0 ISPF library?

You can order the ISPF books using the numbers provided below.

OS/390 V2R8.0 ISPF

Title	Order Number
OS/390 V2R8.0 ISPF Dialog Tag Language Guide and Reference	SC28-1219-03
OS/390 V2R8.0 ISPF Planning and Customizing	SC28-1298-03
OS/390 V2R8.0 ISPF User's Guide	SC28-1239-03
OS/390 V2R8.0 ISPF Services Guide	SC28-1272-03
OS/390 V2R8.0 ISPF Dialog Developer's Guide and Reference	SC28-1273-03
OS/390 V2R8.0 ISPF Reference Summary	SC28-1308-03
OS/390 V2R8.0 ISPF Edit and Edit Macros	SC28-1312-03
OS/390 V2R8.0 ISPF Library Management Facility	SC28-1317-03
OS/390 V2R8.0 ISPF Messages and Codes	GC28-1326-03
OS/390 V2R8.0 ISPF Software Configuration and Library Manager Project Manager's and Developer's Guide	SC34-4750-01
OS/390 V2R8.0 ISPF Software Configuration and Library Manager Reference	SC28-1320-03
OS/390 V2R8.0 ISPF Application Server User's Guide and Reference	SC34-4752-01
Entire library Bill of Forms	SBOF-8567

The licensed books that were declassified in OS/390 Version 2 Release 4 appear on the OS/390 Online Library Collection, SK2T-6700.

The remaining licensed books for OS/390 Version 2 appear on the OS/390 Licensed Product Library, LK2T-2499, in unencrypted form.

© Copyright IBM Corp. 1980, 1999

Elements and Features in OS/390

You can use the following table to see the relationship of a product you are familiar with and how it is referred to in OS/390 Version 2 Release 8.0. OS/390 V2R8.0 is made up of elements and features that contain function at or beyond the release level of the products listed in the following table. The table gives the name and level of each product on which an OS/390 element or feature is based, identifies the OS/390 name of the element or feature, and indicates whether it is part of the base or optional. For more compatibility information about OS/390 elements see *OS/390 Planning for Installation, GC28-1726*

Product Name and Level	**Name in OS/390**	**Base or Optional**
BookManager BUILD/MVS V1R3	BookManager BUILD	optional
BookManager READ/MVS V1R3	BookManager READ	base
MVS/Bulk Data Transfer V2	Bulk Data Transfer (BDT)	base
MVS/Bulk Data Transfer File-to-File V2	Bulk Data Transfer (BDT) File-to-File	optional
MVS/Bulk Data Transfer SNA NJE V2	Bulk Data Transfer (BDT) SNA NJE	optional
IBM OS/390 C/C++ V1R2	C/C++	optional
DFSMSdfp V1R3	DFSMSdfp	base
DFSMSdss	DFSMSdss	optional
DFSMShsm	DFSMShsm	optional
DFSMSrmm	DFSMSrmm	optional
DFSMS/MVS Network File System V1R3	DFSMS/MVS Network File System	base
DFSORT R13	DFSORT	optional
EREP MVS V3R5	EREP	base
FFST/MVS V1R2	FFST/MVS	base
GDDM/MVS V3R2 • GDDM-OS/2 LINK • GDDM-PCLK	GDDM	base
GDDM-PGF V2R1.3	GDDM-PGF	optional
GDDM-REXX/MVS V3R2	GDDM-REXX	optional
IBM High Level Assembler for MVS & VM & VSE V1R2	High Level Assembler	base
IBM High Level Assembler Toolkit	High Level Assembler Toolkit	optional
ICKDSF R16	ICKDSF	base
ISPF V4R2M1	ISPF	base
Language Environment for MVS & VM V1R5	Language Environment	base
Language Environment V1R5 Data Decryption	Language Environment Data Decryption	optional

© Copyright IBM Corp. 1980, 1999

Product Name and Level	Name in OS/390	Base or Optional
MVS/ESA SP V5R2.2		
BCP	BCP or MVS	base
ESCON Director Support	ESCON Director Support	base
Hardware Configuration Definition (HCD)	Hardware Configuration Definition (HCD)	base base
JES2 V5R2.0	JES2	optional
JES3 V5R2.1	JES3	base
LANRES/MVS V1R3.1	LANRES	base
IBM LAN Server for MVS V1R1	LAN Server	base
MICR/OCR Support	MICR/OCR Support	base
OS/390 UNIX System Services	OS/390 UNIX System Services	base
OS/390 UNIX Application Services	OS/390 UNIX Application Services	base
OS/390 UNIX DCE Base Services (OSF DCE level 1.1)	OS/390 UNIX DCE Base Services	base
OS/390 UNIX DCE Distributed File Services (DFS) (OSF DCE level 1.1)	OS/390 UNIX DCE Distributed File Services (DFS)	optional
OS/390 UNIX DCE User Data Privacy	OS/390 UNIX DCE User Data Privacy	optional
SOMobjects Application Development Environment (ADE) V1R1	SOMobjects Application Development Environment (ADE)	base
SOMobjects Runtime Library (RTL)	SOMobjects Runtime Library (RTL)	base
SOMobjects service classes	SOMobjects service classes	
Open Systems Adapter Support Facility (OSA/SF) R1	Open Systems Adapter Support Facility (OSA/SF)	base
MVS/ESA RMF V5R2	RMF	optional
OS/390 Security Server	Resource Access Control Facility (RACF) • DCE Security Server • OS/390 Firewall Technologies • Lightweight Directory Access Protocol (LDAP) Client and Server • Open Cryptographic Enhanced Plug-ins (OCEP)	optional
SDSF V1R6	SDSF	optional
SMP/E	SMP/E	base
	Softcopy Print	base
SystemView for MVS Base	SystemView for MVS Base	base
IBM TCP/IP V3R1 • TCP/IP CICS Sockets • TCP/IP IMS Sockets • TCP/IP Kerberos • TCP/IP Network Print Facility (NPF) • TCP/IP OS/390 Communications Service IP Applications • TCP/IP OS/2 Offload	TCP/IP • TCP/IP CICS Sockets • TCP/IP IMS Sockets • TCP/IP Kerberos • TCP/IP Network Print Facility (NPF) • TCP/IP OS/390 Communications Service IP Applications • TCP/IP OS/2 Offload	base • optional • optional • optional • optional • optional • optional
TIOC R1	TIOC	base
Time Sharing Option Extensions (TSO/E) V2R5	TSO/E	base

Product Name and Level	Name in OS/390	Base or Optional
VisualLift for MVS V1R1.1	• VisualLift Run-Time Environment (RTE) • VisualLift Application Development Environment (ADE)	• base • optional
VTAM V4R3 with the AnyNet feature	VTAM	base
3270 PC File Transfer Program V1R1.1	3270 PC File Transfer Program	base

The ISPF User Interface

ISPF provides an action bar-driven interface that exploits many of the usability features of Common User Access (CUA) interfaces. Refer to *Object-Oriented Interface Design: IBM Common User Access Guidelines* for additional information.

The panels look different than in Version 3: all screens are in mixed case, and most have action bars at the top. These action bars give you a new way to move around in the product as well as access to command nesting. Command nesting allows you to *suspend* an activity while you perform a new one rather than having to end a function to perform another function.

This chapter primarily explains the action bar-driven interface and the use of ISPF's graphical user interface (GUI).

Some Terms You Should Know

The following terms are used in this book:

action bar. The area at the top of an ISPF panel that contains choices that give you access to actions available on that panel. When you select an action bar choice, ISPF displays a *pull-down menu*.

pull-down menu. A list of numbered choices extending from the selection you made on the action bar. The action bar selection is highlighted; for example, Utilities in Figure 1 on page xxxi appears highlighted on your screen. You can select an action either by typing in its number and pressing Enter or by selecting the action with your cursor. ISPF displays the requested panel. If your choice contains an *ellipsis* (...), ISPF displays a *pop-up window*. When you exit this panel or pop-up, ISPF closes the pull-down and returns you to the panel from which you made the initial action bar selection.

ellipsis. Three dots that follow a pull-down choice. When you select a choice that contains an ellipsis, ISPF displays a *pop-up* window.

pop-up window. A bordered temporary window that displays over another panel.

modal pop-up window. A type of window that requires you to interact with the panel in the pop-up before continuing. This includes cancelling the window or supplying information requested.

modeless pop-up window. A type of window that allows you to interact with the dialog that produced the pop-up before interacting with the pop-up itself.

point-and-shoot text. Text on a screen that is cursor-sensitive. See "Point-and-Shoot Text Fields" on page xxxiv for more information.

push button. A rectangle with text inside. Push buttons are used in windows for actions that occur immediately when the push button is selected (available only when you are running in GUI mode).

function key. In previous releases of ISPF, a programmed function (PF) key. *This is a change in terminology only*.

select. In conjunction with point-and-shoot text fields and action bar choices, this means moving the cursor to a field and simulating Enter.

mnemonics. Action bar choices can be defined with a underscored letter in the action bar choice text. In host mode you can access the action bar choice with the ACTIONS command and parameter 'x', where 'x' is the underscored letter in the action bar choice text. In GUI mode you can use a *hot key* to access a choice on the action bar; that is, you can press the ALT key in combination with the letter that is underscored in the action bar choice text.

© Copyright IBM Corp. 1980, 1999

How to Navigate in ISPF without Using Action Bars

If you use a non-programmable terminal to access OS/390 V2R8.0 ISPF and you do not want to take advantage of the command nesting function, you can make selections the same way you always have: by typing in a selection number and pressing Enter.

How to Navigate in ISPF Using the Action Bar Interface

Most ISPF panels have action bars at the top; the choices appear on the screen in white by default. Many panels also have point-and-shoot text fields, which appear in turquoise by default. The panel shown in Figure 3 on page xxxii has both.

Action Bars

Action bars give you another way to move through ISPF. If the cursor is located somewhere on the panel, there are several ways to move it to the action bar:

- Use the cursor movement keys to manually place the cursor on an action bar choice.
- Type ACTIONS on the command line and press Enter to move the cursor to the first action bar choice.
- Press F10 (Actions) or the Home key to move the cursor to the first action bar choice.

  If mnemonics are defined for action bar choices, you can:
  - In 3270 mode, on the command line, type ACTIONS and the mnemonic letter that corresponds to an underscored letter in the action bar choice text. This results in the display of the pull-down menu for that action bar choice.
  - In 3270 mode, on the command line enter the mnemonic letter that corresponds to an underscored letter in the action bar choice text, and press the function key assigned to the ACTIONS command. This results in the display of the pull-down menu for that action bar choice.
  - In GUI mode, you can use a *hot key* to access a choice on an action bar or on a pull-down menu; that is, you can press the ALT key in combination with the mnemonic letter that is underscored in the choice text to activate the text.

Use the tab key to move the cursor among the action bar choices. If you are running in GUI mode, use the right and left cursor keys.

Notes:

1. ISPF does not provide a mouse emulator program. This book uses *select* in conjunction with point-and-shoot text fields and action bar choices to mean moving the cursor to a field and simulating Enter.

   Note: Some users program their mouse emulators as follows:
   - Mouse button 1 – to position the cursor to the pointer and simulate Enter
   - Mouse button 2 – to simulate F12 (Cancel).
2. If you want the Home key to position the cursor at the first input field on an ISPF panel, type `SETTINGS` on any command line and press Enter to display the ISPF Settings panel. Deselect the **Tab to action bar choices** option.
3. If you are running in GUI mode, the Home key takes you to the beginning of the current field.

When you select one of the choices on the action bar, ISPF displays a pull-down menu. Figure 1 shows the pull-down menu displayed when you select Utilities on the ISPF Primary Option Menu action bar.

```
  Menu  Utilities 1 Compilers  Options  Status  Help
-----  ------------------------------------------------------------------------
         _ 1. Library                ary Option Menu
           2. Data set
0  Se      3. Move/Copy              arameters            User ID . : USERID
1  Vi      4. Data Set List           or listings         Time. . . : 15:08
2  Ed      5. Reset Statistics       urce data            Terminal. : 3278
3  Ut      6. Hardcopy               ctions               Screen. . : 1
4  Fo      7. Download...            e processing         Language. : ENGLISH
5  Ba      8. Outlist                uage processing      Appl ID . : ISR
6  Co      9. Commands...            ation commands       TSO logon : SERPROC
7  Di     *0. Reserved               ing                  TSO prefix: USERID
8  LM     11. Format                 or functions         System ID : VS1C
9  IB     12. SuperC                 ment products        MVS acct. : 76TD0B0Z
10 SC     13. SuperCE                brary Manager        Release . : ISPF
11 Wo     14. Search-For             Workplace
          15. Search-ForE
Enter                                defaults


Option ===>
 F1=Help      F2=Split     F3=Exit      F7=Backward  F8=Forward   F9=Swap
F10=Actions  F12=Cancel
```

1 The selected action bar choice is highlighted.

Figure 1. Panel with an Action Bar Pull-Down Menu

To select a choice from the Utilities pull-down menu, type its number in the entry field (underlined) and press Enter or select the choice. To cancel a pull-down menu without making a selection, press F12 (Cancel). For example, if you select choice 9, ISPF displays the Command Table Utility pop-up, as shown in Figure 2 on page xxxii.

Note: If you entered a command on the command line prior to selecting an action bar choice, the command is processed, and the pull-down menu is never displayed. The CANCEL, END, and RETURN commands are exceptions. These three commands are not processed and the cursor is repositioned to the first input field in the panel body. If there is no input field, the cursor is repositioned under the action bar area. If you are running in GUI mode and select an action bar choice, any existing command on the command line is ignored.

```
  Menu  Utilities  Compilers  Options  Status  Help
- ──────────────────────────────── Commands ────────────────────────────────
  │                          Command Table Utility                         │
  │                                                                        │
  │0    Specifications                          Command search order       │
  │1    Application ID . . ISR                  Application table  : ISR   │
  │2    Enter "/" to select option              User table . . . . :       │
  │3    __ Show description field               Site table . . . . :       │
  │4                                            System table . . . : ISP   │
  │5                                                                       │
  │6  If no application ID is specified, the current application ID will be│
  │7  used.  The name of the command table to be processed is formed by   │
  │8  prefixing the application id to the string 'CMDS'.  For example:     │
  │9  Application ID  . .  TST results in a command table name of 'TSTCMDS'.│
  │1                                                                       │
  │1  Command ===>                                                         │
  │    F1=Help       F2=Split      F3=Exit       F7=Backward   F8=Forward  │
  │E   F9=Swap      F12=Cancel                                             │
  └────────────────────────────────────────────────────────────────────────┘

  Option ===>
   F1=Help      F2=Split     F3=Exit      F7=Backward  F8=Forward   F9=Swap
  F10=Actions  F12=Cancel
```

Figure 2. Pop-Up Selected from an Action Bar Pull-Down

```
1 Menu  Utilities  Compilers  Options  Status  Help
 ------------------------------------------------------------------------------
     2                    ISPF Primary Option Menu                        3
 0  Settings      Terminal and user parameters            User ID . : USERID
 1  View          Display source data or listings         Time. . . : 09:31
 2  Edit          Create or change source data            Terminal. : 3278
 3  Utilities     Perform utility functions               Screen. . : 1
 4  Foreground    Interactive language processing         Language. : ENGLISH
 5  Batch         Submit job for language processing      Appl ID . : ISR
 6  Command       Enter TSO or Workstation commands       TSO logon : SERPROC
 7  Dialog Test   Perform dialog testing                  TSO prefix: USERID
 8  LM Facility   Library administrator functions         System ID : VS1C
 9  IBM Products  IBM program development products        MVS acct. : 522126,B
 10 SCLM          SW Configuration Library Manager        Release . : ISPF 4.2
 11 Workplace     ISPF Object/Action Workplace Shell

 Enter X to Terminate using log/list defaults


 Option ===> ____________________________________________________________
  F1=Help      F2=Split     F3=Exit      F7=Backward  F8=Forward   F9=Swap
 F10=Actions  F12=Cancel
```

1 Action bar. You can select any of the action bar choices and display a pull-down.

2 Options. The fields in this column are point-and-shoot text fields.

3 Dynamic status area. You can specify what you want to be displayed in this area.

Figure 3. Panel with an Action Bar and Point-and-Shoot Fields

Action Bar Choices

The action bar choices available vary from panel to panel, as do the choices available from their pull-downs. However, Menu and Utilities are basic action bar choices, and the choices on their pull-down menus are always the same.

Menu Action Bar Choice

The following choices are available from the Menu pull-down:

Settings	Displays the ISPF Settings panel
View	Displays the View Entry panel
Edit	Displays the Edit Entry panel
ISPF Command Shell	Displays the ISPF Command Shell panel
Dialog Test...	Displays the Dialog Test Primary Option panel
Other IBM Products...	Displays the Additional IBM Program Development Products panel
SCLM	Displays the SCLM Main Menu
ISPF Workplace	Displays the Workplace entry panel
Status Area...	Displays the ISPF Status panel
Exit	Exits ISPF.

Note: If a choice displays in blue (the default) with an asterisk as the first digit of the selection number (if you are running in GUI mode, the choice will be *grayed*), the choice is unavailable for one of the following reasons:

- Recursive entry is not permitted here
- The choice is the current state; for example, RefMode is currently set to Retrieve in Figure 4 on page xxxiv.

```
  Menu  RefList  RefMode  Utilities  LMF  Workstation  Help
--------------  ---------------------------------------------
                1  1. List Execute    ry Panel
                   *. List Retrieve
ISPF Library:
   Project . . . USERID
   Group . . . . PRIVATE  . . .          . . .          . . .
   Type  . . . . PANELS
   Member  . . .                 (Blank or pattern for member selection list)

Other Partitioned or Sequential Data Set:
   Data Set Name . . .
   Volume Serial . . .           (If not cataloged)

Workstation File:
   File Name . . . . .

Initial Macro  . . . .                  Options
Profile Name . . . . .                  /  Confirm Cancel/Move/Replace
Format Name  . . . . .                     Mixed Mode
Data Set Password  . .                     Edit on Workstation
Command ===>
 F1=Help      F2=Split     F3=Exit      F7=Backward  F8=Forward   F9=Swap
F10=Actions  F12=Cancel
```

Figure 4. An Unavailable Choice on a Pull-Down

Utilities Action Bar Choice

The following choices are available from the Utilities pull-down:

Library	Displays the Library Utility panel
Data Set	Displays the Data Set Utility panel
Move/Copy	Displays the Move/Copy Utility panel
Data Set List	Displays the Data Set List Options panel
Reset Statistics	Displays the Reset ISPF Statistics panel
Hardcopy	Displays the Hardcopy Utility panel
Download...	Displays the panel that enables you to download workstation clients and other files from the host.
Outlist	Displays the Outlist Utility panel
Commands...	Displays the Command Table Utility panel
Reserved	Reserved for future use by ISPF; an unavailable choice
Format	Displays the Format Specification panel
SuperC	Displays the SuperC Utility panel
SuperCE	Displays the SuperCE Utility panel
Search-for	Displays the Search-For Utility panel.
Search-forE	Displays the Search-ForE Utility panel.

Point-and-Shoot Text Fields

Point-and-shoot text fields are cursor-sensitive; if you select a field, the action described in that field is performed. For example, if you select Option 0, Settings, in Figure 3 on page xxxii, ISPF displays the ISPF Settings panel.

Note: If you have entered a command on the command line, this command is processed before any point-and-shoot command unless you are running in GUI mode.

The cursor-sensitive portion of a field often extends past the field name. Until you are familiar with this new feature of ISPF, you might want to display these fields in reverse video (use the PSCOLOR command to set Highlight to REVERSE).

Note: You can use the Tab key to position the cursor to point-and-shoot fields by selecting the **Tab to point-and-shoot fields** option on the ISPF Settings panel (Option 0).

Function Keys

ISPF uses CUA-compliant definitions for function keys F1–F12 (except inside the Edit function). F13–F24 are the same as in ISPF Version 3. By default you see the CUA definitions because your **Primary range** field is set to 1 (Lower - 1 to 12).

To use non-CUA-compliant keys, select the **Tailor function key display** choice from the Function keys pull-down on the ISPF Settings (option 0) panel action bar. On the Tailor Function Key Definition Display panel, specify 2 (Upper - 13 to 24) in the **Primary range** field.

The following function keys help you navigate in ISPF:

F1 **Help**. Displays Help information. If you press F1 (and it is set to Help) after ISPF displays a short message, a long message displays in a pop-up window.

F2 **Split**. Divides the screen into two logical screens separated by a horizontal line or changes the location of the horizontal line.

Note: If you are running in GUI mode, each logical screen displays in a separate window.

F3 **Exit** (from a pull-down). Exits the panel underneath a pull-down.

F3 **End**. Ends the current function.

F7 **Backward**. Moves the screen up the scroll amount.

F8 **Forward**. Moves the screen down the scroll amount.

F9 **Swap**. Moves the cursor to where it was previously positioned on the other logical screen of a split-screen pair.

F10 **Actions**. Moves the cursor to the action bar. If you press F10 a second time, the cursor moves to the command line.

F12 **Cancel**. Issues the Cancel command. Use this command to remove a pull-down menu if you do not want to make a selection. F12 also moves the cursor from the action bar to the Option ==> field on the ISPF Primary Option Menu. See *ISPF Dialog Developer's Guide and Reference* for cursor-positioning rules.

F16 **Return**. Returns you to the ISPF Primary Option Menu or to the display from which you entered a nested dialog. RETURN is an ISPF system command.

Selection Fields

OS/390 V2R8.0 ISPF uses the following CUA-compliant conventions for selection fields:

A single period (.)

Member lists that use a single period in the selection field recognize only a single selection. For example, within the Edit function you see this on your screen:

```
EDIT     USER1.PRIVATE.TEST                ROW 00001 of 00002
 Name         VV MM  Created   Changed  Size Init  Mod  ID
 . MEM1       01.00  94/05/12  94/07/22  40    0    0   USER1
 . MEM2       01.00  94/05/12  94/07/22  30    0    0   KEENE
```

You can select only one member to edit.

A single underscore (_)

Selection fields marked by a single underscore prompt you to use a slash (/) to select the choice. You may use any non-blank character. For example, the **Panel display CUA mode** field on the ISPF Settings panel has a single underscore for the selection field:

```
Options
  Enter "/" to select option
  _  Command line at bottom
  _  Panel display CUA mode
  _  Long message in pop-up
```

Note: If you are running in GUI mode, this type of selection field displays as a check box; that is, a square box with associated text that represents a choice. When you select a choice, a check mark (in OS/2) or an X (in Windows) appears in the check box to indicate that the choice is in effect. You can clear the check box by selecting the choice again.

An underscored field (____)

Member lists or text fields that use underscores in the selection field recognize multiple selections. For example, from the Display Data Set List Option panel, you may select multiple members for print, rename, delete, edit, browse, or view processing.

Command Nesting

Command nesting allows you to *suspend* an activity while you perform a new one rather than having to end a function to perform another function. For example, in previous versions of ISPF, if you are editing a data set and want to allocate another data set, you type `=3.2` on the command line and press Enter. ISPF *ends* your edit session before taking you to the Data Set Utility panel. When you have allocated the data set and want to return to your edit session, you type `=2` and press Enter; ISPF returns you to the Edit Entry Panel. With OS/390 V2R8.0 ISPF, from your edit session, select the Data set choice from the Utilities pull-down on the Edit panel action bar. ISPF suspends your edit session and displays the Data Set Utility panel. When you have allocated the new data set and end the function, OS/390 V2R8.0 ISPF returns you directly to your edit session rather than to the Edit Entry Panel.

Chapter 1. Introduction to ISPF

This chapter describes ISPF at an introductory level. It explains what ISPF is and what it does for you.

What Is ISPF?

Consider the Interactive System Productivity Facility (ISPF) program product an extension of the MVS Time Sharing Option (TSO) host system on which it runs. ISPF services complement those of the host system to provide interactive processing. ISPF is similar to a control program or access method in that it provides services to dialogs (applications) during their execution. The types of services provided by ISPF are:

- Display services
- File-tailoring services
- Variable services
- Table services
- Miscellaneous services
- Dialog test facility, including:
  - Setting breakpoints
  - Tracing usage of dialog services and dialog variables
  - Browsing trace output in the ISPF log data set
  - Examining and updating ISPF tables
  - Interactively invoking most dialog services.

A dialog receives requests and data from a user at a terminal. The dialog responds by using ISPF services to obtain information from, or enter information into, a computer system.

What Is a Dialog?

To understand the dialog interface, you must first understand what a dialog is. A dialog is the interaction between a person and a computer. It helps a person who is using an interactive display terminal to exchange information with a computer.

The user starts an interactive application through an interface that the system provides. The dialog with the user begins with the computer displaying a panel and asking for user interaction. It ends when the task for which the interactions were initiated is completed.

A dialog developer creates the parts of a dialog, called dialog elements. Each dialog application is made up of a command procedure or program, together with dialog elements that allow an orderly interaction between the computer and the application user.

The elements that make up a dialog application are:

- Functions
- Variables
- Command tables
- Panel definitions
- Message definitions
- File-tailoring skeletons

© Copyright IBM Corp. 1980, 1999

- Tables.

A dialog does not necessarily include all types of elements, particularly tables and skeletons, which certain kinds of applications do not use.

Functions

A function is a command procedure or a program that performs processing requested by the user. It can invoke ISPF dialog services to display panels and messages, build and maintain tables, generate output data sets, and control operational modes.

A function can be coded in a command procedure language using CLIST or REXX or in a programming language, such as PL/I, COBOL, FORTRAN, APL2*, Pascal, or C. The following compilers can be used:
- PL/I Optimizing Compiler (5668-909)
- OS/VS COBOL Compiler and Library (5740-CB1)
- VS COBOL II Compiler, Library and Debug (5668-958)
- VS FORTRAN Compiler and Library (5748-FO3)
- VS PASCAL (5668-767)
- APL2 (5668-899)
- C/370* Compiler (5668-040).

You can use more than one language in a dialog application. For example, within a single application containing three functions, each function could be written using a different language, such as PL/I, COBOL, or FORTRAN. One or more of the functions can be written using a command procedure language instead of a programming language.

A function coded in a programming language can be designed for cross-system use, to be processed by equivalent levels of ISPF running under Virtual Memory (VM) or MVS.

Notes:

1. ISPF functions written in PL/I should not be linked with the PL/I multitasking libraries.
2. ISPF functions written in FORTRAN should be linked in FORTRAN link mode. That is, include the VLNKMLIB library ahead of the VFORTLIB library in the SYSLIB concatenation. Refer to the *VS FORTRAN Programming Guide* for additional information.
3. ISPF functions written in the C language should be linked with the C$START load module. For more information, refer to the *C Compiler User's Guide*

Variables

ISPF services use variables to communicate information among the various elements of a dialog application. ISPF provides a group of services for variable management. Variables can vary in length from zero to 32K bytes and are stored in variable pools according to how they are to be used. A set of variables whose names begin with the character Z are system variables. Z variables are reserved for ISPF system-related uses.

Command Tables

A system command table (ISPCMDS) is distributed with ISPF in the table input library. An application can provide an application command table by including a table named *xxxx*CMDS in its table input library, where *xxxx* is a 1- to 4-character application ID. In addition, you can specify two other command tables, the User command table and the Site command table. The application IDs of both are specified in the ISPF Configuration table. You can also specify if the Site command table is searched before or after the system command table.

You can define an application command table either by using the Dialog Tag Language (DTL) and ISPF conversion utility, or by using ISPF option 3.9.

When a user enters a command, the dialog manager searches the application command table, if any, and then the system command table. If it finds the command, action is taken immediately. If it does not find the command in the application or system tables, the command is passed to the dialog, unaltered, in the command field. The dialog then takes appropriate action.

Panel Definitions

A panel definition is a programmed description of the panel. It defines both the content and format of a panel.

Most panels prompt the user for input. The user's response can identify which path is to be taken through the dialog, as on a selection panel. The response can be interpreted as data, as on a data-entry panel.

Message Definitions

Message definitions specify the format and text of messages to users. A message can confirm that a user-requested action is in progress or completed, or it can report an error in the user's input. Messages can be superimposed on the display to which they apply, directed to a hardcopy log, or both.

File-tailoring Skeletons

A file-tailoring skeleton, or simply a skeleton, is a generalized representation of sequential data. It can be customized during dialog execution to produce an output data set. After a skeleton is processed, the output data set can be used to drive other processes. File skeletons are frequently used to produce job data sets for batch execution.

Tables

Tables are two-dimensional arrays that contain data and are created by dialog processing. They can be created as a temporary data repository, or they can be retained across sessions. A retained table can also be shared among several applications. The type and amount of data stored in a table depends on the nature of the application.

Tables are generated and updated during dialog execution. The organization of each table is specified to ISPF using ISPF table services.

What Does a Dialog Do?

You can use ISPF to simplify the programming that provides interactive application operations. Operations performed during dialog execution include:

- Identifying to the user choices of available processing routines
- Invoking a requested routine, based on the user's choice
- Prompting the user to enter data
- Reading the data into a work area
- Checking the data to verify that it is appropriate for the application

  If the data is not appropriate for the application:
  - Identifying the error to the user
  - Prompting the user to enter new data and verifying that data

  If the entered data is in the proper form:
  - Displaying any information requested by the user
  - Processing or storing the user's data, then advising the user of its disposition
- Creating sequential output data sets or reports
- Providing online messages, help, and tutorial displays to help users understand application processing.

Developing a Dialog

A developer, using an editor such as the PDF editor in Option 2 of ISPF, develops a dialog by creating its various elements at a terminal and storing them in libraries. You can use any available editor when creating dialog elements.

However, in addition to an editor, ISPF provides special facilities to aid dialog development. Examples of these facilities are:

- A VIEW facility for displaying source data or output listings
- Utilities to simplify data handling
- Programming-language processing facilities
- Edit models for messages, file-tailoring skeletons, panels, and DTL source
- Library access services for accessing both ISPF libraries and other data sets.

Figure 5 on page 5 shows a developer using ISPF to create and test dialog elements. As shown in the figure, panel definitions, message definitions, and file-tailoring skeletons are created prior to running the dialog. These dialog elements are saved in libraries. The developer stores the program (after compilation) or command procedure in an appropriate system program library. During dialog testing, tables of data, log entries, and file-tailoring output data sets can be created by dialog processing. ISPF creates the log data set the first time the user performs some action that results in a log message, such as saving edited data or submitting a job to the batch machine. ISPF creates the list data set the first time a user requests a print function or executes a dialog that issues a LIST service request.

When the developer completes the functions, panel definitions, and any other dialog elements required by the application being developed, the dialog is ready to be processed under ISPF.

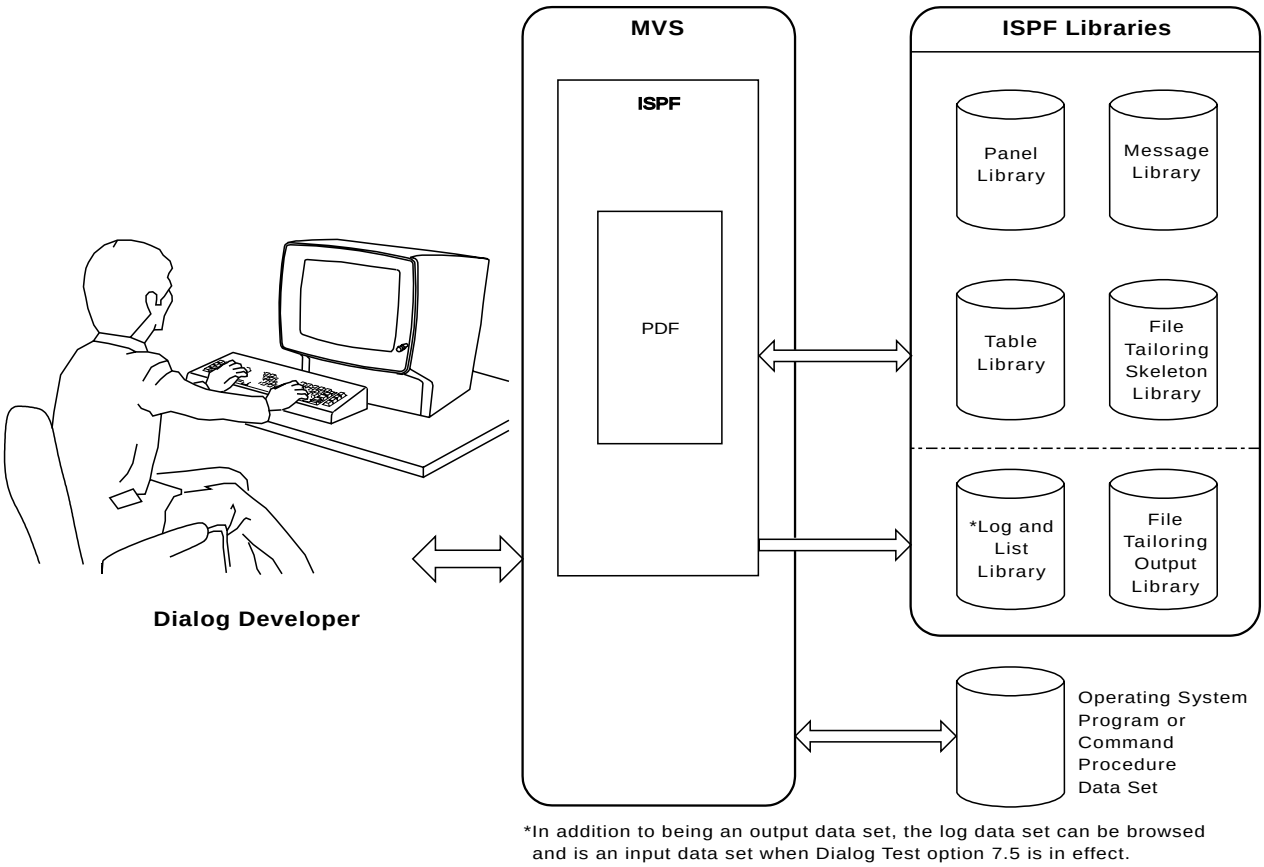


Figure 5. Using ISPF

How Dialog Elements Interact

A dialog can 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 6 on page 6, starts with display of the highest menu, called the primary option menu. User options selected from the primary option menu can result in the call of a function or the display of a lower-level menu. Each lower-level menu can also cause functions to receive control or still other menus to be displayed.

Eventually, a function receives control. The function can use any of the dialog services provided by ISPF. Typically, the function can continue the interaction with the user by means of the DISPLAY service. The function might also display data-entry panels to prompt the user for information. When the function ends, the menu from which it was invoked is redisplayed.

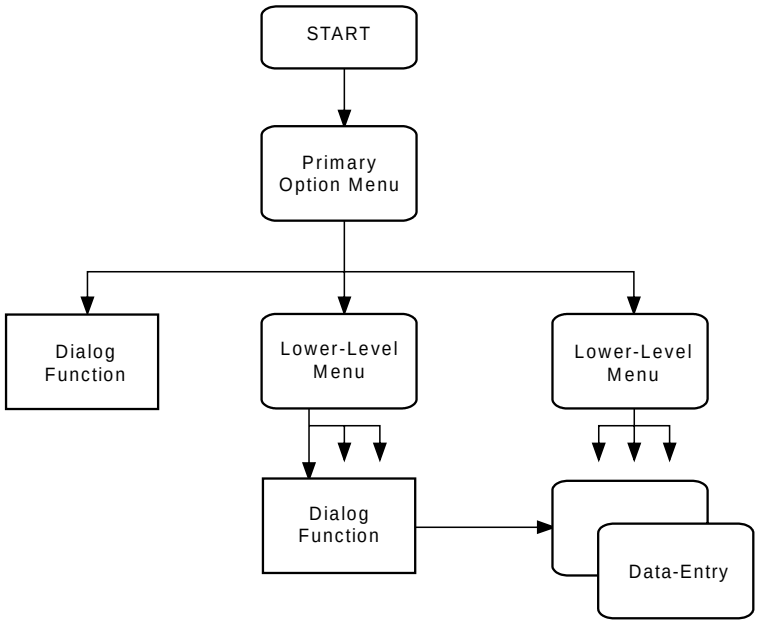


Figure 6. Typical Dialog Organization Starting with a Menu

Figure 7 on page 7 shows another type of dialog organization in which a dialog function receives control first, prior to the display of a menu. The function performs application-dependent initialization and displays data-entry panels to prompt the user for basic information. It then starts the selection process by using the SELECT service to display the primary option menu for the application.

As shown in Figure 7 on page 7, a function can also invoke another function without displaying a menu. It uses the SELECT service to do this, which 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.

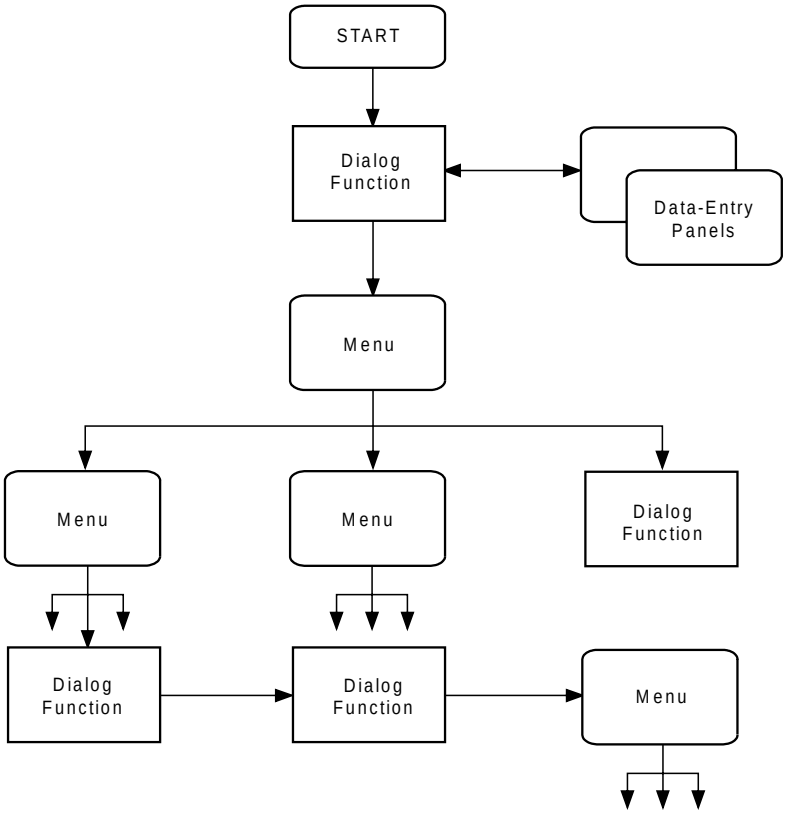


Figure 7. Typical Dialog Starting with a Function

To relate your application design to CUA design models and principles, refer to the *IBM* Common User Access Guidelines* It is recommended you use DTL to design CUA-based panels. Refer to the *ISPF Dialog Tag Language Guide* for more information.

Dialog Variables

ISPF uses dialog variables to communicate data between the dialog management services and the dialog elements. A dialog variable's value is a character string that can vary in length from 0 to 32K bytes. Some services restrict the length of dialog variable data.

Dialog variables are referred to symbolically. The name is composed of 1–8 characters (6 for FORTRAN). Alphanumeric characters A–Z, 0–9, #, $, or @ can be used in the name, but the first character cannot be numeric. APL variable names cannot contain #, $, or @.

Dialog variables can be used with panels, messages, and skeleton definitions, as well as within dialog functions. For example, a dialog variable name can be defined in a panel definition, and then referred to in a function of the same dialog. Or, the variable can be defined in a function, then used in a panel definition to initialize information on a display panel, then later used to store data entered by the user on the display panel.

For functions coded in a programming language other than APL2, the internal program variables that are to be used as dialog variables can be identified to ISPF

and accessed using the ISPF variable services. The use of STEM or COMPOUND variables within a REXX procedure is not supported by ISPF. For a function coded as CLIST or REXX command procedures or as an APL2 procedure, variables used in the procedure are automatically treated as dialog variables. In this case, no special action is required to define them to ISPF.

Chapter 2. Controlling ISPF Sessions

This chapter is intended to help you understand how to control ISPF sessions. It describes how to start and stop an ISPF session and how to use many of the ISPF facilities.

Dialog Control and Data Flow

Dialog control and data flow are illustrated in Figure 8. At the start of an ISPF session, you can use the ISPSTART command to either request a selection panel from which to choose the first task or call a dialog function. The figure also illustrates how the ISPF services interact with the various dialog elements.

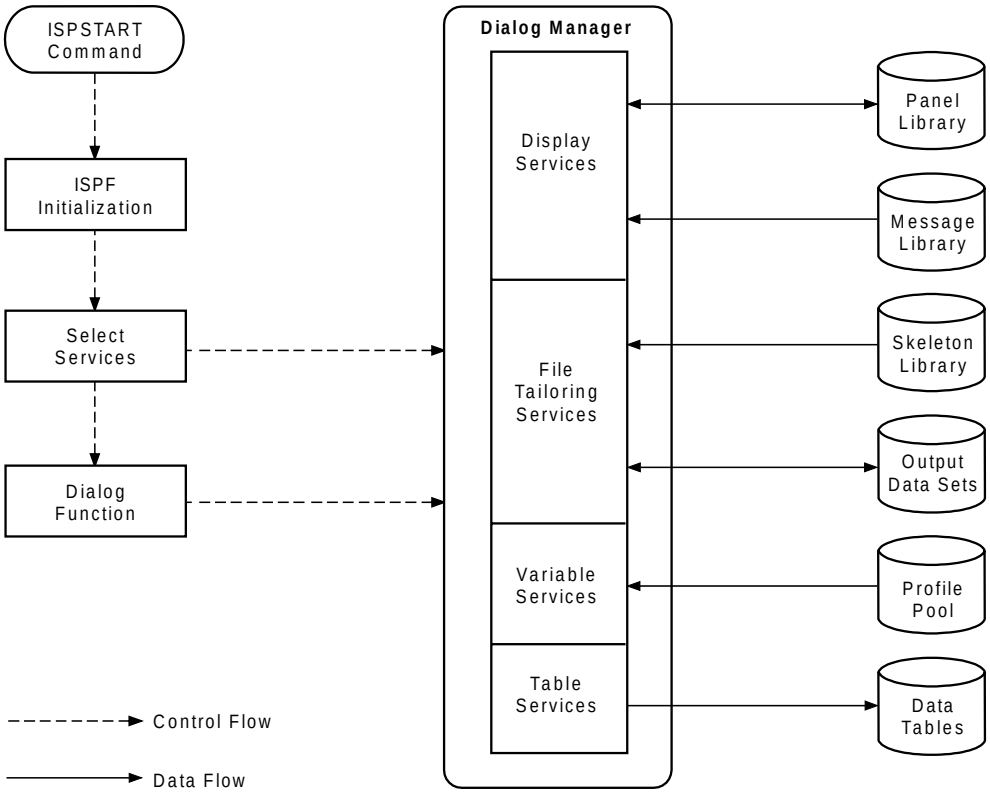


Figure 8. Control and Data Flow

Processing a Dialog

Figure 9 on page 10 shows a dialog being processed under ISPF. The figure shows that ISPF dialog services are available only to command procedures or programs running under ISPF. During dialog processing, the dialog requests specific ISPF services and identifies the panel and message definitions, skeletons, and tables to use. The figure also shows that entries in the log and list data sets, as well as the file-tailoring output data sets, can be generated during dialog processing.

© Copyright IBM Corp. 1980, 1999

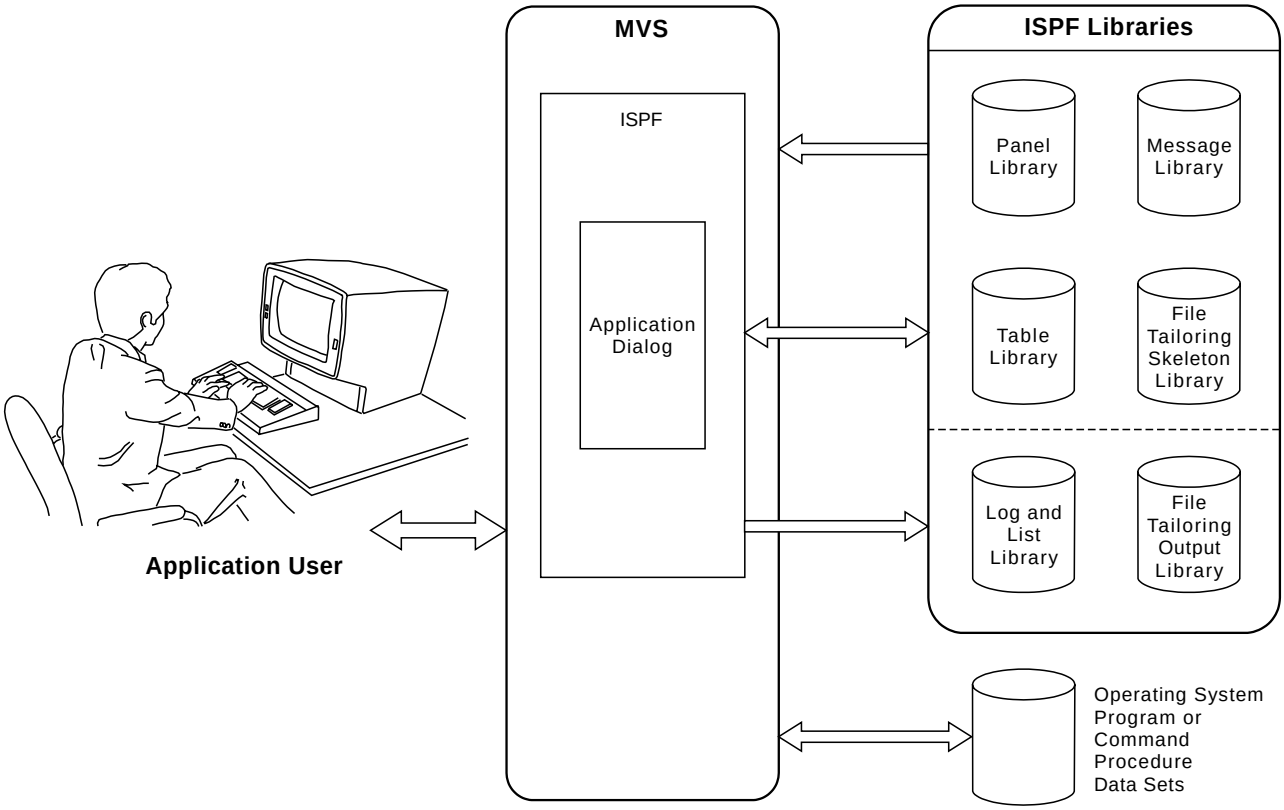


Figure 9. Application Dialog Running under ISPF

Dialog processing begins either with the display of a selection panel or with a function. In either case, you can invoke a dialog from a terminal running under control of TSO.

Starting a Dialog

You can use the ISPF, PDF, or ISPSTART command, with the CMD, PGM, or PANEL keyword, to invoke ISPF or other dialogs. These commands provide compatibility with the SPF licensed program. ISPF is a command procedure that runs under TSO. For example, it can be invoked by a:

- Terminal running under TSO
- Command procedure (CLIST or REXX).

Before starting a dialog, data sets referred to by the dialog must be defined to ISPF.

Syntax for Issuing the ISPSTART Command

You invoke ISPF by using the ISPSTART command. ISPSTART command parameters specify the first menu to be displayed or the first function to receive control prior to the display of a menu.

If no parameters are specified, the ISPSTART command defaults to PANEL(ISP@MSTR). If the PDF or ISPF command is used to start the product, the default is PANEL(ISP@PRIM) NEWAPPL(ISR). All the parameters below apply to the PDF and ISPF command as well as ISPSTART. For more information about the GUI parameters, refer to the *OS/390 ISPF User's Guide*.

```
ISPSTART
{PANEL(panel-name) [OPT(option)][ADDPOP]}
{CMD(command parm1 parm2) [LANG(APL|CREX)]}
{PGM(program-name) [PARM(parameters)]}
{WSCMD(workstation-command)
              [MODAL|MODELESS]
              [WSDIR(dir)]
              [MAX|MIN]
              [VIS|INVIS]}
{WSCMDV(var_name)
              [MODAL|MODELESS]
              [WSDIR(dir)]
              [MAX|MIN]
              [VIS|INVIS]            }
[GUI(LU:address:tpname | IP:address:port |,FI:) |,NOGUIDSP)] [TITLE(title)]
[GUISCRW(screen-width)]
[GUISCRD(screen-depth)]
[FRAME(STD|FIX|DLG)]
[CODEPAGE(codepage)] [CHARSET(character_set)]
[GUIWEB(application_name, token_id, maxwait_value)
[BKGRND(STD|DLG)]
[NEWAPPL[(application-id)]]
[SCRNAME(screen-name)]
[TEST|TESTX|TRACE|TRACEX]
[NOLOGO|LOGO(logo-panel-name)]
[BATSCRW(screen-width)]
[BATSCRD(screen-depth)]
[BDISPMAX(max-number-of-displays)]
[BREDIMAX(max-number-of-redisplays)]
[BDBCS]
[DANISH|ENGLISH|GERMAN|JAPANESE|
PORTUGUE|SPANISH|KOREAN|FRENCH|ITALIAN|CHINESET|
CHINESES|SGERMAN|UPPERENG]
```

where:

panel-name
Specifies the name of the first menu (that is, the primary option menu) to be displayed.

option
Specifies an initial option, which should be a valid option on the first menu. This causes direct entry to that option without displaying the primary option menu. (The primary option menu is processed in nondisplay mode, as though the user had entered the option.) If you specify an option that is not valid, the primary option menu displays an appropriate error message.

ADDPOP
Specifies that the panel displayed from a SELECT service appears in a pop-up window. An explicit REMPOP is performed when the SELECT PANEL has ended.

command
Specifies a command procedure (CLIST or REXX), an APL2 command, or a TSO command processor that is to be invoked as the first dialog function. For more information on invoking APL2 dialogs, see *ISPF Services Guide*

CLIST or REXX command parameters can be included within the parentheses. For example, the call format would be:

```
ISPSTART CMD(MYCLIST parm1 parm2 ...)
```

These parameters are passed to the command procedure. You can find information about specifying CLIST parameters in *TSO/E Version 2 CLISTs* and information about specifying REXX parameters in *TSO/E Version 2 REXX User's Guide*

You can type a percent sign (%) preceding the CLIST or REXX procedure name to:

- Improve performance
- Prevent ISPF from entering line-display mode when the procedure is started.

  Note: When starting a CLIST or REXX procedure or a program through the SELECT service, a MODE(LINE|FSCR) parameter is available for specifying either line- or full-screen mode. If you do not specify the mode parameter or do not use the % prefix, ISPF enters line-display mode.
- Ensure that the command procedure is invoked if ISPF has access to a program function that has the same name as the procedure. If you use the percent sign prefix, ISPF searches only for a procedure with the specified name. However, without the percent sign prefix, ISPF searches first for a program, then for a CLIST or REXX procedure.

On extended data stream terminals, using the percent sign causes the keyboard to remain in a locked condition. To avoid this condition, the CLIST or REXX procedure can issue output line I/O before issuing a READ.

LANG(APL|CREX)
Specifies special language invocations. LANG(APL) specifies to start the command specified by the CMD keyword, and to start an APL2 environment. LANG(CREX) specifies that the command specified by the CMD keyword is a REXX EXEC that has been compiled and link-edited into a LOAD module and that a CLIST/REXX function pool is to be used.

program-name
Specifies the name of a program that is to be invoked as the first dialog function. In PL/I, it must be a MAIN procedure. This parameter must specify the name of a load module that is accessible by use of the LINK macro.

However, if the program dialog consists of multiple tasks and if any of the subtasks use ISPF services, the CMD keyword, not the PGM keyword, must be used. Dialog developers should avoid using prefixes ISP and ISR, the ISPF component codes, in naming dialog functions. Special linkage conventions, intended only for internal ISPF use, are used to invoke programs named `ISPxxxxx` and `ISRxxxxx`.

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 half-word 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 PL/I, 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.

workstation-command
Specifies a fully-qualified workstation program with any of its parameters. To issue a command that is not a program (.exe, .com, .bat, or .cmd file in OS/2) the DOS and OS/2 environments allow the command to be prefaced with the DOS COMMAND command or the OS/2 CMD command. For example, in the DOS environment: SELECT WSCMD(COMMAND /C DIR C:), or in the OS/2 environment: SELECT WSCMD(CMD /C DIR C:).

Note: When using the ISPF Application Server and Workstation Agent Applet to make your workstation connection, the WSCMD is limited in its capabilities. The only commands supported with the server and the applet are:

BROWSER: httpaddress
This starts a new browser window with the specified http address.

EDIT: filename
The filename is the name of a file that you have transferred to the workstation and that now resides in dynamica memory. This command starts a JAVA editor to edit the *in memory* copy of the file. You can then transfer the file back to the host, it cannot be saved on the workstation.

DELETE: filename
The filename is the name of a file that you have in memory on the workstation.

MODAL
The MODAL parameter invokes the workstation command modally. It waits until the workstation command has completed and then returns to ISPF.

MODELESS
The MODELESS parameter invokes the command modelessly and is only valid when running in GUI mode. It is the default. It does not wait until the workstation command has completed. It always returns a return code of zero if the command was started, even if the command does not exist at the workstation.

WSDIR(dir)
The WSDIR parameter specifies the variable name containing the workstation current working directory. This directory is the directory from which the workstation command should be invoked.

MAX
The MAX parameter attempts to start the workstation command in a maximized window. The workstation command may override this request. MAX and MIN are mutually exclusive.

MIN
The MIN parameter attempts to start the workstation command in a minimized window. The workstation command may override this request. MAX and MIN are mutually exclusive.

VIS
The VIS parameter attempts to start the workstation command as a visible window. The workstation command may override this request. This is the default. VIS and INVIS are mutually exclusive.

INVIS
The INVIS parameter attempts to start the workstation command in an invisible (hidden) window. The workstation command may override this request. VIS and INVIS are mutually exclusive.

var_name
Specifies a variable name that contains the text string of a command and its parameters. Use this when the command path or parameters, or both, contain any of the following: imbedded blanks, quotation marks, or special characters that might not parse properly with the WSCMD service.

LU:*address:tpname*
Specifies the workstation's Advanced Program-to-Program Communication (APPC) network name.

Note: The variable ZGUI will be set to the workstation address (in character format) if ISPSTART is issued with the GUI parameter; ZGUI will be set to blank if ISPSTART is issued without the GUI parameter.

IP:*address:port*
Specifies the workstation's Transmission Control Protocol/Internet Protocol (TCP/IP) hardware-level IP address: a fully qualified machine name.

Note: The variable ZGUI will be set to the workstation address (in character format) if ISPSTART is issued with the GUI parameter; ZGUI will be set to blank if ISPSTART is issued without the GUI parameter.

FI:
Specifies that you want to search a file allocated to DD ISPDTPRF for the user's network protocol and workstation address to be used when initiating a workstation connection or GUI display. Refer to the *OS/390 ISPF User's Guide* for more information.

NOGUIDSP
Specifies that you want to make a connection to the workstation, but DO NOT want ISPF to display in GUI mode.

Note: This parameter is only valid if you have specified an **LU**, **IP**, or **FI**parameter. In other words, you can have any of the following situations:
- you specify **LU:address:tpname**, **IP:address:port**, *or* **FI:** without the NOGUIDSP parameter
- *or* you specify **LU:address:tpname, NOGUIDSP**
- *or* you specify **IP:address:port, NOGUIDSP**
- *or* you specify **FI:, NOGUIDSP**

TITLE(*title*)
Specifies the text displayed in the title bar unless a dialog has assigned a non-blank value to ZWINTTL or ZAPPTTL. The default value for the title bar

is the user ID. This value has a maximum length of 255 characters and will be truncated without notice to the user at display time if it does not fit on the panel.

GUISCRW(*screen-width***)**
Allows you to specify a screen width different than that of the emulator or real device from which you enter the ISPSTART command. If you do not specify GUISCRD, the depth will be that of the emulator or real device.

If GUISCRW is different than the emulator or real device, and GUI initialization fails, ISPF will not initialize. Dialogs started with dimensions other than those of the emulator or real device that use the GRINIT service will not display GDDM* screens.

Note: This parameter is usually only used with the **GUI** parameter, but you can specify it without using the GUI parameter. When you do this and use a valid value for this parameter, ISPF does nothing with the value. However, if you specify a value that is not valid for this parameter, ISPF might return an error condition.

GUISCRD(*screen-depth***)**
Allows you to specify a screen depth different than that of the emulator or real device from which you enter the ISPSTART command. If you do not specify GUISCRW, the width will be that of the emulator or real device.

If GUISCRD is different than the emulator or real device and GUI initialization fails, ISPF will not initialize. Dialogs started with dimensions other than those of the emulator or real device that use the GRINIT service will not display GDDM screens.

Note: This parameter is usually only used with the **GUI** parameter, but you can specify it without using the GUI parameter. When you do this and use a valid value for this parameter, ISPF does nothing with the value. However, if you specify a value that is not valid for this parameter, ISPF might return an error condition.

CODEPAGE(*codepage***) CHARSET(***character_set***)**

When running in GUI mode or connecting to the workstation, these values are used as the host codepage and character set in translating data from the host to the workstation, regardless of the values returned from the terminal query response.

When running in 3270 mode, if your terminal or emulator does not support codepages, these values are used as the host codepage and character set. Otherwise, these values are ignored.

GUIWEB(*application_name, token_id, maxwait_value, ISPF_restart***)**
The options for this parameter are:

application_name
The name by which this application is known to the application server and the Workstation Agent applet of the ISPF Application Server feature. Refer to the *OS/390 ISPF Application Server User's Guide and Reference* for more information.

token_id
The token used to specify the current instance of this application

identifying a batch mode TSO/ISPF session submitted by the ISPF Application Server. This parameter is not used for TSO/ISPF sessions started by other means.

maxwait_value
An integer specifying the number of seconds that this ISPF session waits for the Application Server to establish a connection to a Workstation Agent applet.

Note: This parameter is usually only used with the **GUI** parameter, but you can specify it without using the GUI parameter. When you do this and use a valid value for this parameter, ISPF does nothing with the value. However, if you specify a value that is not valid for this parameter, ISPF might return an error condition.

FRAME(STD|FIX|DLG)
Specifies that the first window frame displayed will be a standard (STD), fixed (FIX), or dialog (DLG) window frame, where:

Standard
A GUI window frame that can be resized and has max/min buttons. This is the default value.

Fixed A GUI window frame that has max/min buttons but cannot be resized.

Dialog
A GUI window frame that cannot be resized and does not have max/min buttons.

Note: Pop-up panels are displayed in dialog frames by default.

Note: This parameter is usually only used with the **GUI** parameter, but you can specify it without using the GUI parameter. When you do this and use a valid value for this parameter, ISPF does nothing with the value. However, if you specify a value that is not valid for this parameter, ISPF might return an error condition.

BKGRND(STD|DLG)
Specifies that the first window frame displayed will be a standard (STD) or dialog (DLG) background color. The colors are defined by the workstation. In OS/2 2.1, the colors are white for STD and gray for DLG. The default is DLG.

Note: This parameter is usually only used with the **GUI** parameter, but you can specify it without using the GUI parameter. When you do this and use a valid value for this parameter, ISPF does nothing with the value. However, if you specify a value that is not valid for this parameter, ISPF might return an error condition.

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 to 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.

SCRNAME(screen-name)
Specifies a screen name to be used with the SWAP command and the ISPF task list. The name can be any set of 2 to 8 characters, except LIST, PREV, or NEXT.

TEST
Specifies that ISPF is to be operated in TEST mode, described under "ISPF Test and Trace Modes" on page 25.

TESTX
Specifies that ISPF is to be operated in extended TEST mode, described under "ISPF Test and Trace Modes" on page 25.

TRACE
Specifies that ISPF is to be operated in TRACE mode, described under "ISPF Trace Modes" on page 27.

TRACEX
Specifies that ISPF is to be operated in extended TRACE mode, described under "ISPF Trace Modes" on page 27.

LOGO(logo-panel-name)
Specifies that ISPF displays the named panel prior to invoking the specified dialog object. Subsequent SELECT service requests that identify a LOGO panel will *not* result in the indicated panel being displayed. This includes a repeat of the first SELECT as a result of a split-screen request or a logical screen restart following a severe dialog error.

Applications can choose to display their own LOGO panel directly. These applications can determine whether the user specified the NOLOGO keyword on ISPSTART by retrieving the ISPF system variable ZLOGO. Applications that choose to display their own LOGO panel are responsible for controlling that display operation during split-screen operations and logical-screen restart situations.

NOLOGO
Specifies that ISPF is to bypass the display of the message pop-up window containing the product title and copyright statement.

screen-width
For batch mode, specifies screen width in character positions. The default value is 80. This parameter is ignored when not running in batch mode.

All screen sizes from 24 x 80 to 62 x 160 are valid.

screen-depth
For batch mode, specifies screen depth in lines. The default value is 32. This parameter is ignored when not running in batch mode.

max-number-of-displays
For batch mode, specifies the maximum number of displays that can occur during a session. This number includes the total of all SELECT PANEL calls, plus all DISPLAY and TBDISPL calls (with or without panel name). This number does not include redisplays related to the .MSG control variable. The largest number that can be specified is 999999999. The batch default value is 100. This parameter is ignored when not running in batch mode.

max-number-of-redisplays
For batch mode, specifies the maximum number of redisplays allowed for a .MSG-redisplay loop. The largest number that can be specified is 255. The batch default value is 2. This parameter is ignored when not running in batch mode.

BDBCS
For batch mode, specifies that Double-Byte Character Set (DBCS) terminal support is required. This parameter is ignored when not running in batch mode.

DANISH, ENGLISH, GERMAN, JAPANESE, PORTUGUE, SPANISH, KOREAN, FRENCH, ITALIAN, CHINESET, CHINESES, SGERMAN, UPPERENG
Specifies the national language that is to override the default language for this session. The JAPANESE keyword specifies that the KANJI character set is to be used. The CHINESET keyword stands for Traditional Chinese, CHINESES stands for Simplified Chinese, and SGERMAN stands for Swiss-German. The UPPERENG keyword specifies that the uppercase English character set is to be used. For information about establishing the default session language, refer to the *ISPF Planning and Customizing* manual.

Notes:

1. Attempting to run a dialog under a session language other than that for which it was intended may produce unexpected results.
2. When the Korean, French, Italian, Traditional Chinese, Simplified Chinese, Spanish, Brazilian-Portuguese, or Danish session language is specified, its respective literal module is used. However, the ISPF product panels and messages are displayed in English.
3. If you specify the CMD (command) or PANEL (panel) keyword more than once on an ISPSTART command line, ISPF uses the last value specified. For example, If you specify:

   ```
   ISPSTART PANEL(PANELA) PANEL(PANELX)
   ```

   ISPF interprets the command as:

   ```
   ISPSTART PANEL(PANELX)
   ```

Using the ISPSTART Command

ISPSTART command parameters specify the first menu to be displayed or the first function to receive control. For example:

```
ISPSTART PANEL(ABC)
```

invokes ISPF and specifies that dialog processing is to begin with display of a selection panel named ABC. Panel ABC is stored in the panel library.

Another example:

```
ISPSTART CMD(%DEF)
```

invokes ISPF and specifies that dialog processing is to begin with a CLIST command procedure function named DEF.

The following example:

```
ISPSTART PGM(GHI)
```

invokes ISPF and specifies that dialog processing is to begin with a program function named GHI.

The ISPSTART command is typically entered during logon or from a command procedure. For example, suppose you begin an application from a terminal by invoking a command procedure named ABC. Procedure ABC allocates the

appropriate libraries for the application, and then issues an ISPSTART command to begin ISPF processing. The ABC procedure cannot use ISPF dialog services, because it does not run under ISPF.

ISPF is a command processor that can be attached by another command processor as a subtask. You should always specify SZERO=NO in the MVS ATTACH macro, as ISPF does when it attaches a subtask, to assure that at ISPF termination the storage that was acquired by ISPF will be released. For more information on using MVS macros, refer to *MVS/XA Supervisor Services and Macros*

Invoking a Dialog from a Selection Panel

Figure 11 shows a selection panel on which the user has selected option 3. When the user presses Enter, option 3, the INVENTORY application, is given control.

```
------------------------------- BUILDING 661 ----------------------------
 SELECT OPTION ===> 3_

   1  PAYROLL    - Add, update, or delete employee records
   2  MAILING    - Add, delete, or change address of employee
   3  INVENTORY  - Status of stock
   4  SCHEDULE   - Building maintenance

 ENTER END COMMAND TO TERMINATE.
```

Figure 11. Sample Selection Panel

Invoking a Dialog from a Master Application Menu

If your installation provides an ISPF master application menu, you can invoke a dialog from that menu. A master application menu is one from which any of the installation's applications can be invoked. It generally is displayed at the beginning of each ISPF session. Figure 12 on page 20 is an illustration of the sample master application menu shipped with ISPF.

```
                    ISPF Master Application Menu

   1 Sample 1     Sample application 1                      Userid . : USERID
   2 .            (Description for option 2                 Time . . : 13:41
   3 .            (Description for option 3)                Terminal : 3278
   4              (Description for option 4)                PF keys  : 12
   5 .            (Description for option 5)                Screen . : 1
   X Exit         Terminate ISPF using list/log defaults    Language : ENGLISH
                                                            Appl ID  : ISR
                                                            Release  : ISPF

 Enter END command to terminate application

 5655-042 (C) Copyright IBM CORP 1982, 1996


  Licensed Materials - Property of IBM 5645-001, 5655-042 (C) Copyright IBM
  Corp. 1980, 1996. All rights reserved. US Government Users Restricted Rights
  - Use, duplication or disclosure restricted by GSA ADP Schedule Contract
  with IBM Corp.

 Option  ===> _____________________________________________________________
  F1=Help      F2=Split     F3=Exit      F9=Swap     F10=Actions  F12=Cancel
```

Figure 12. ISPF Master Application Menu (ISP@MSTR)

You usually invoke the master menu by using the ISPSTART command with no operands. ISPSTART can be issued automatically as part of a user's logon procedure or from a CLIST or REXX command procedure.

What the SELECT Service Does

The SELECT service initiates dialog execution. 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 on how the SELECT service was originally invoked.

SELECT is both a control facility and a dialog service. ISPF uses SELECT during its initialization to invoke the function or selection panel that begins a dialog. During dialog processing, SELECT displays selection panels and invokes program functions or command procedure functions.

The principal SELECT parameters are:
- PANEL(panel-name)
- CMD(command)
- PGM(program-name)

See *ISPF Services Guide* for a full description of the SELECT service syntax.

The panel-name parameter specifies the name of the next selection panel to be displayed. You must use the ISPF panel definition statements (described in "Chapter 5. Panel Definition Statement Guide" on page 103) to define the panel.

The command and program-name parameters specify a function, coded as a CLIST command procedure or program, respectively, to receive control. Input parameters

can be passed to the function as part of the command specification or, for programs, by the use of the PARM parameter.

Figure 13 shows how the SELECT service is used when invoking or processing a dialog. After SELECT starts a dialog, the dialog uses it as a service to invoke a function or to display a selection panel. In turn, that function or menu can use SELECT to invoke another function or to display another menu. This function or menu can, in turn, using SELECT, invoke still another function or menu. This process can continue for many levels and establishes a hierarchy of invoked functions and menus. There is no restriction on the number of levels allowed in this hierarchy.

Subtasks attached by the SELECT service do not share subpools. ISPF specifies SZERO=NO when issuing the ATTACH macro to assure that at SELECT termination the storage that was acquired by ISPF is released.

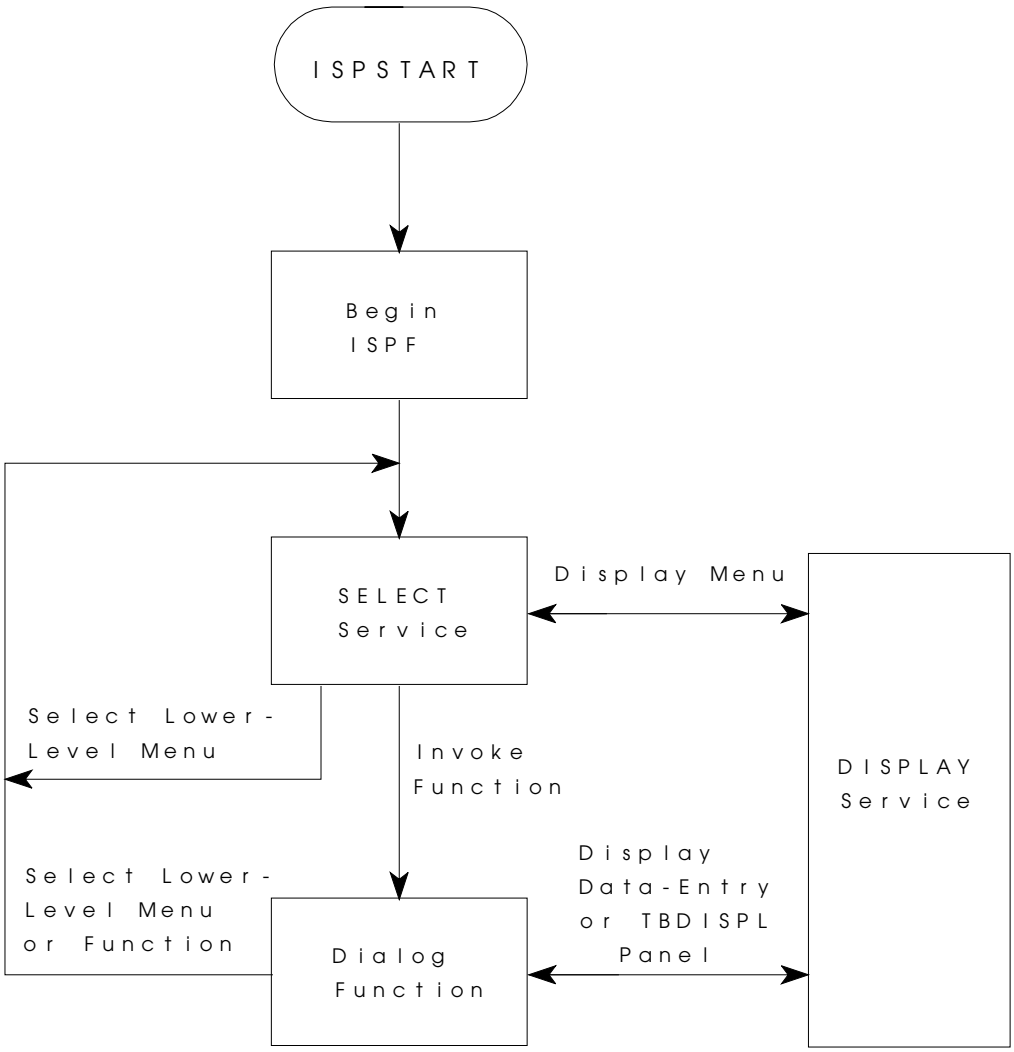


Figure 13. SELECT Service Used to Invoke and Process a Dialog

When a lower-level function or menu in the hierarchy completes its processing, control returns 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. This hierarchy determines the sequence in which functions and menus are processed, including the sequence in which they are terminated.

Dialog functions written as command procedures can directly invoke other functions written as command procedures without using the SELECT service. They are *not* treated as new functions by ISPF.

Dialog functions written as programs 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 function by ISPF.

Invoking the SELECT Service

The SELECT service can be invoked in the following ways:

- During initialization, the dialog manager automatically invokes the SELECT service to start 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 is fully initialized. ISPF is considered to be fully initialized when the Enter key on the primary option menu has been processed without a severe error occurring.
- If you enter 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 new logical screen.
- The SELECT service recursively invokes itself when you select 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 can be invoked from a dialog function. In this case, the selection keywords are passed as calling sequence parameters.

Terminating a Dialog

The action taken at dialog termination is as follows:

- If a dialog function invoked the SELECT service, control returns to that function and the function continues execution.
- If a menu invoked the SELECT service, that menu is redisplayed, including execution of the INIT section in the panel definition.
- If you are terminating split-screen mode, the original dialog ends on that logical screen, and the other logical screen expands to the full size of the viewing area.
- If you are terminating ISPF, which can be done only in single-screen mode, either the ISPF termination panel is displayed or the ISPF SETTINGS defaults for list/log processing are used.

ISPF displays the termination panel if:

- The dialog started with the display of a menu and you 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 you entered the RETURN command or selected the EXIT option.
- The dialog started with the execution of a function and the function ended with a return code of 4 or higher. A return code other than 0 or 4 causes an error message to be displayed.

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

Return Codes from Terminating Dialogs

The return code from ISPSTART for a successful dialog completion is either 0 or a value returned by the executing dialog in shared-pool system variable ZISPFRC. ZISPFRC is a shared pool input variable of length 8. The dialog can set ZISPFRC to any value in the range of 0 to 16777215, except the values reserved for ISPF use (900 through 999). This value must be left-justified and padded with blanks.

At termination, ISPF copies the value from ZISPFRC and passes it to the invoking application (or Terminal Monitor Program) in register 15. If the value in ZISPFRC is not within the valid range or is otherwise not valid, such as a value that is not numeric, ISPF issues an appropriate line message and passes a return code of 908. If the dialog has not set ZISPFRC to a value, ISPF returns a value of 0.

Notes:

1. CLIST procedures that invoke ISPSTART can check the CLIST variable LASTCC for the ISPF return code. In REXX, check the variable *rc* after an ISPF function.
2. Even though ISPF restricts the return code value to the range 0 to 16777215, other products or subsystems, such as JES when processing JCL condition codes, can be more restrictive on return code values. See documentation for the affected product for more information.
3. ZISPFRC should not be confused with the normal dialog return code set by the function; it has no effect on ISPF log/list termination processing.

ZISPFRC is intended to be used by applications that invoke a dialog dedicated to a single task or function. However, it is valid to set ZISPFRC from a selection panel invoked by the ISPSTART command.

ISPF checks for the existence of ZISPFRC only at ISPF termination. If ZISPFRC is set by any dialog other than the one invoked by the ISPSTART command, ISPF ignores the value.

Return Codes from Termination Dialogs

Error codes that ISPF can return in register 15 to an application are:

908 ZISPFRC value not valid

920 ISPSTART command syntax not valid

985 An attempt was made to start a GUI in batch mode, but no workstation connection was made.

987 An attempt was made to start GUI with GUISCRW or GUISCRD and the GUI intitialization failed.

988 An error occurred intializing IKJSATTN

989 The ISPF C/S component window was closed while still running ISPF in GUI mode

990 An error occurred running in batch mode. If ZISPFRC has not been set previously, and ISPF encounters a severe error that terminates the product, then 990 is set.

997 Uncorrectable TPUT error

998 ISPF initialization error. A 998 error code can result from:
- Required ISPF data element library not preallocated
- Error opening ISPF data element library
- ISPF data element library has invalid data set characteristics
- Error loading literals module
- Recursive ISPF call

ISPF issues a line message that indicates which of these errors caused the 998 return code.

999 ISPF environment not valid. A 999 error code can result from:
- TSO/MVS environment not valid
- Unsupported screen size

ISPF issues a line message that indicates which of these errors caused the 999 return code.

An Example Using the ZISPFRC Return Code

Figure 14 on page 25 shows a portion of a background job that invokes ISPF. The final job step executes only if the job step that invoked the ISPF dialog terminates with a return code of 8 or less.

```
⋮

//*************************************************
//*                                               *
//*  INVOKE ISPF TO EXECUTE DIALOG "DIALOG1".     *
//*  DIALOG1 PASSES BACK A RETURN CODE OF         *
//*  20 IF IT DID NOT PROCESS SUCCESSFULLY.       *
//*                                               *
//*************************************************
//ISPFSTEP EXEC PGM=IKJEFT01,DYNAMNBR=30,REGION=2048K
//*
//*  ALLOCATE DIALOG AND ISPF PRODUCT LIBRARIES, *
//*  ISPF LOG DATA SET, AND TSO OUTPUT DATA SET. *
//*                                               *
//ISPPROF  DD DSN=USER1.ISPF.TABLES,DISP=SHR
⋮

//*  ALLOCATE TSO INPUT DATA SET.                 *
//*                                               *
//SYSTSIN  DD *
  PROFILE PREFIX(USER1)     /* ESTABLISH PREFIX  */
  ISPSTART CMD(%DIALOG1)    /* INVOKE DIALOG1    */
/*
//*************************************************
//*                                               *
//*  EXECUTE NEXT JOB STEP ONLY IF THE ISPF STEP *
//*  ENDED WITH A RETURN CODE LESS THAN OR EQUAL *
//*  TO 8.  THAT IS, BYPASS THE STEP IF 8 IS      *
//*  LESS THAN THE ISPF RETURN CODE.              *
//*                                               *
//*************************************************
//NEXTSTEP EXEC PGM=IKJEFT01,DYNAMNBR=30,REGION=2048K,
//              COND=(8,LT,ISPFSTEP)
⋮
```

Figure 14. Sample Background ISPF Job

The portion of the invoked dialog, DIALOG1, that establishes the value in system variable ZISPFRC is shown in Figure 15.

```
PROC 0
⋮

IF &MAXCC > 8 THEN +
  DO
    SET &ZISPFRC = 20
    VPUT (ZISPFRC) SHARED
  END
EXIT CODE(0)
```

Figure 15. Sample Dialog Using System Variable ZISPFRC

ISPF Test and Trace Modes

The testing modes of ISPF provide special processing actions to help debug a dialog. Consider using the Dialog Test (option 7) facility.

You can specify any one of four mutually exclusive keyword parameters on the ISPSTART command to control the operational mode when testing a dialog:

TEST Test mode

TESTX
Extended test mode; logged messages are displayed

TRACE
Trace mode; ISPF service calls are logged

TRACEX
Extended trace mode; ISPF service calls are logged and displayed

Test Modes

In TEST mode, ISPF operates differently from normal mode in that:

- Panel and message definitions are fetched again from the panel and message files when 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. If you have modified the panel or message file, use of TEST mode ensures that the latest version of each panel or message is accessed during a test run.

  Using an editor to modify a panel, message, or skeleton can result in an additional DASD extent being required for the associated data set. DASD rarely (if ever) gains new extents as the result of the execution of software (with the possible exception of DASD formatting software). It can also be caused by link editing a module. When a new extent is allocated, you can access the modification only by first terminating and then invoking ISPF again.
- Tutorial panels are displayed with current panel name, previous panel name, and previous message ID on the bottom line of the display screen. This assists you in identifying the position of the panel in the tutorial hierarchy.
- Screen printouts, obtained through use of the PRINT or PRINT-HI commands, include line numbers, current panel name, and message ID.
- In PDF, the index listing (option 3.1) for a partitioned data set includes TTR data for each member of the data set.
- 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 ISPF can ABEND.
- Other than the situation described in the previous item, and if you have an ABEND from a SELECTed command as described under the SELECT service in *ISPF Services Guide* any ISPF-detected error, ABEND, or program interrupt forces an ABEND of ISPF. You can also force an ABEND by entering ABEND or CRASH in the command line of any panel.
- The PA1 key causes an immediate exit from ISPF.

The ISPF controller task attaches one ISPF subtask for each logical screen. Any additional logical screens are created by the SPLIT command and there can be up to four screens on a 3290 terminal.

If an ISPF subtask ABENDs, pressing Enter after the ABEND message appears generates a dump, provided that a SYSUDUMP, SYSMDUMP, or SYSABEND data set has been allocated.

Dialogs invoked with the SELECT CMD(XXX) cause an attach of a new subtask under the ISPF subtask. If an ABEND occurs under the new subtask, an immediate dump is taken.

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

ISPF provides the ENVIRON command, which allows you to cause a dump following an ABEND condition, even if ISPF is not running in TEST mode. See "Using the ENVIRON System Command" on page 334 for a description of using the ENVIRON command.

ISPF Trace Modes

In TRACE mode, ISPF operates as it does in TEST mode, except that a message is written to the ISPF log file when any ISPF service is invoked, even if CONTROL ERRORS RETURN has been issued, and when any error is detected by an ISPF service. Note that only CLIST, APL2, and CALL ISPEXEC service requests are recorded. This does not include service requests issued under Dialog Test option 7.6. CALL ISPLINK 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. If the length of the message text exceeds the width of the terminal screen, the message will be truncated.

Invoking Authorized Programs

You can invoke authorized programs by using the SELECT service, a selection panel, a command table, or by using the TSO CALL command under ISPF. ISPF uses the TSO Service Facility IKJEFTSR to invoke authorized commands and programs. Authorized programs are invoked under the TSO TMP (Terminal Monitor Program) and therefore should not reside in the ISPLLIB library. Authorized programs cannot issue dialog service requests. Refer to *TSO/E Version 2 Customization* for information on adding authorized programs and commands to the list maintained by your installation.

Invoking TSO Commands

TSO commands can be initiated by use of the SELECT dialog service (with the CMD keyword), from a selection panel, from a command table, by entering the ISPF TSO system command in the command field of any panel, or be contained in a CLIST or REXX command procedure that is invoked under ISPF.

You can invoke authorized TSO commands by using the SELECT service, a selection panel, or a command table. Authorized commands are attached under the TSO TMP (Terminal Monitor Program) and, therefore, should not reside in the ISPLLIB library. Authorized commands cannot issue dialog service requests.

You can execute most TSO commands under ISPF. The following commands are not allowed:
- LOGON
- LOGOFF
- SPF
- ISPF
- PDF
- ISPSTART
- TEST
- Commands that are restricted by TSO or PCF (Program Control Facility).

Note: The LOGON, LOGOFF, and TEST commands can be executed within ISPF if the TSOEXEC interface is used, for example, TSO TSOEXEC LOGOFF. In that case, the LOGON and LOGOFF commands are processed upon ISPF termination, instead of returning to TSO READY. When the TEST command is being executed, TSO TEST is entered immediately. However, because TSOEXEC executes commands in a parallel TMP structure, ISPF dialogs cannot be run under TSO TEST in this situation.

Compiled REXX Requirements

ISPF now supports Compiled REXX Load Modules through ISPSTART and the SELECT Service. The REXX program must be compiled with the OBJECT option of the IBM Compiler for REXX/370, product number 5695-013. This OBJECT output needs to be link edited with the CPPL stub that is a part of the IBM Library for REXX/370, product number 5695-014.

The SELECT service and ISPSTART command contains a new value, CREX, for the LANG parameter on the CMD keyword. The specification of LANG(CREX) when using the CMD keyword indicates that it is a Compiled REXX load module and that a REXX function pool is to be used for variable manipulation.

The CPPL stub takes the parameters that are passed by the SELECT CMD service, or the ISPSTART invocation, and converts them into arguments for the REXX program. For complete details on how to create a REXX load module, see *IBM Compiler and Library for REXX/370 User's Guide and Reference*, SH19-8160.

Compiled REXX programs, compiled with the CEXEC option, should be executed using the CMD option of the SELECT service or ISPSTART command and should NOT use the LANG(CREX) parameter.

CLIST Requirements

A CLIST cannot invoke any of the restricted TSO commands. TERMIN command procedure statements can cause unpredictable results.

Note: If a CLIST contains CONTROL MAIN, the TSO input stack is not flushed after an ISPF severe error.

When a CLIST command procedure is executing under ISPF, the ATTN statement in the procedure defines how attention interrupts are to be handled. You can find information about using attention exits in *TSO/E Version 2 CLISTs* and *TSO/E Version 2 Programming Services*

To use attention exits in a CLIST ISPF dialog, your system must be MVS SP2.2.0 or later.

Restrictions to Using Attention Exits from CLISTs

Restrictions that apply to using attention exits from a CLIST dialog are:

- CLIST attention exits are not supported when running in ISPF TEST or TRACE modes. This is because the ISPF attention exit routine is not established in TEST or TRACE modes.

- The CLIST must issue a null command to return from an attention exit. If the dialog issues a TSO command to terminate the exit routine, ISPF discards the command. The ISPF dialog then resumes execution as if CONTROL MAIN NOFLUSH were in effect for this CLIST.
- You can stack CLIST attention exits only within one SELECT CMD level. An exit applies only to the logical screen from which the CLIST owning the attention exit was invoked. Therefore, when you are operating in split-screen mode, invoking a CLIST attention exit from one logical screen has no effect on the other logical screen(s).
- You should not invoke an ISPF dialog service from a CLIST attention exit routine. If you do, results are unpredictable.
- Attention interrupts initiated while an exit routine is executing are not honored.

Examples of CLIST Attention Exit Process Flow

Single CLIST with One Attention Exit

1. From a selection panel, select a CLIST procedure named CLIST1. CLIST1 has one attention exit routine, named ATTN1.
2. CLIST1 displays PANEL1.
3. Press the attention key.
4. Exit routine ATTN1 executes and PANEL1 redisplays.

Nested CLISTs with Two Attention Exits (One SELECT Level)

1. From a selection panel, select a CLIST procedure named CLIST1. CLIST1 has one attention exit routine, named ATTN1.
2. CLIST1 invokes procedure CLIST2 by using the TSO EXEC command. CLIST2 has one attention exit routine, named ATTN2.
3. CLIST2 displays PANEL2.
4. Press the attention key.
5. Exit routine ATTN2 executes and PANEL2 redisplays.
6. Press Enter to return control to CLIST2. CLIST2 then terminates processing and control returns to CLIST1.
7. CLIST1 displays PANEL1.
8. Press the attention key.
9. Exit routine ATTN1 executes and PANEL1 redisplays.

Nested CLISTs with One Attention Exit

1. From a selection panel, select a CLIST procedure named CLIST1. CLIST1 has one attention exit routine, named ATTN1.
2. CLIST1 invokes procedure CLIST2 by using the TSO EXEC command. CLIST2 has no attention exit routine.
3. CLIST2 displays PANEL2.
4. Press the attention key.
5. Exit routine ATTN1 executes and PANEL2 redisplays.
6. Press Enter to return control to CLIST2. CLIST2 then terminates processing and control returns to CLIST1.
7. CLIST1 displays PANEL1.
8. Press the attention key.
9. Exit routine ATTN1 executes and PANEL1 redisplays.

Nested CLISTs and SELECT Levels with One Attention Exit

1. From a selection panel, select a CLIST procedure named CLIST1. CLIST1 has one attention exit routine, named ATTN1.
2. CLIST1 invokes procedure CLIST2 by using the ISPEXEC SELECT CMD(CLIST2) command. CLIST2 has no attention exit routine.
3. Press the attention key.
4. Because CLIST2 has no attention exit routine, and ISPF does not propagate attention exits across SELECT levels:
   - An error message indicates that a CLIST was interrupted by an attention condition.
   - The logical screen terminates and restarts, causing the primary option menu to redisplay.

Using APL2

ISPF permits the use of APL2, as follows:

- ISPF dialogs can be written in an APL2 workspace.
- APL2 can be selected as a command, initializing an ISPF-APL2 environment.
- APL2 functions can be selected as options (from a selection panel), as ISPF commands (from an application command table), or from another dialog function, once the ISPF-APL2 environment has been established.
- All dialog manager services available to the command language dialog writer are executable from the APL2 workspace after the ISPF-APL2 environment has been established.
- ISPF views the APL2 workspace variables as the dialog function pool whenever an ISPF dialog service is executing.
- ISPF supports APL on a DBCS device with an APL keyboard.

The ISPF/GDDM* interface is not available to an APL2 dialog. However, the APL2 dialog can interface directly with GDDM and interleave the ISPF and GDDM services.

Invoking APL2

You can invoke APL2 by specifying the APL2 command and its appropriate keywords as the value of the CMD keyword of the SELECT service. In addition, you must code the SELECT keyword and value “LANG(APL)” on the SELECT statement. The LANG(APL) information provides the basis for establishing an ISPF-APL2 environment, and is required if any ISPF dialog services are to be used.

You can code any of the APL2 command keywords. However, the following can be of special interest:

APNAMES

ISPF and APL2 communicate through an APL2 Auxiliary Processor (AP), ISPAPAUX, which is released with the ISPF product. This AP, number 317, must be made available to APL2 when APL2 is invoked, as follows:

- The dialog writer can specify ISPAPAUX in the APNAMES list of auxiliary processors to be dynamically loaded.

  When APL2 is invoked, ISPAPAUX must exist as a load module in a system library, or in a private library named by the LOADLIB keyword.

LOADLIB

Keep in mind that if this keyword is used, the dialog must be changed or accept this keyword's value dynamically (for example, through a variable), if the name of the private library containing the AP is changed.

TERMCODE (code)

The user is prompted to enter an appropriate character if this keyword is not coded. This allows APL2 to identify the terminal type that is currently being used.

Typically, a dialog ensures that the user does not have to perform this extra step by identifying the terminal type through the TERMCODE keyword.

ISPF system variable ZTERM contains this information. However, ISPF terminal types are different from those of APL2. For those dialog writers who wish to make use of currently available ISPF information, program dialog ISPAPTT can be selected before the call of APL2. ISPAPTT expects one parameter, which is the ISPF variable name into which the corresponding APL2 terminal type is returned. The variable is created in the shared variable pool.

For a CLIST, the use of ISPAPTT can look as follows:

```
   .
   .
   .
ISPEXEC SELECT PGM(ISPAPTT) PARM(APLTT)
ISPEXEC VGET APLTT
ISPEXEC SELECT CMD(APL2.....TERMCODE(&APLTT)) LANG(APL)
   .
   .
   .
```

The following ISPF to APL2 mappings are supported:

ISPF (ZTERM)	APL2
3277	3277
3278	3279
3277A	32771
3278A	32791
3278T	32791
3278CF	3279
3277KN	3277
3278KN	3279

If ISPF is executing in the background, then ISPAPTT will return a terminal code of 1.

If ZTERM contains a value other than those listed above, the specified variable is set to a value of 3277 in the shared variable pool.

FREESIZE, WSSIZE

Some combination of these keywords should be coded to accommodate the user's storage requirements; however, remember that ISPF and the ISPF-APL2 AP require storage (beyond that currently allocated) to execute, especially if ISPF split-screen facilities are to be used.

INPUT

A user dialog can specify the INPUT keyword to load a given workspace, start an APL2 dialog function, and terminate APL2. This allows a user to enter APL2, use APL2 dialog capabilities, and leave APL2 without needing special APL2 expertise.

For example, to start a dialog named EMPLOY in workspace MYWS:

```
.....INPUT(')LOAD MYWS' 'EMPLOY' ')OFF HOLD').....
```

Note that a dialog function can also be started through the latent function definition in the workspace. In addition, the Alternate Input Auxiliary Processor, AP101, can be used to stack commands for execution.

If INPUT is coded and QUIET and PROFILE are not coded (see below), the first ISPF panel can be refreshed before the keyboard is unlocked.

QUIET
A dialog can specify the QUIET keyword to suppress the APL2 entry and exit information, so that the user does not see non-dialog APL2 messages.

PROFILE
A dialog can specify the PROFILE keyword with a value of null to suppress any entry and exit APL2 session manager screens, so that the user does not see any non-dialog panels.

Executing APL2 Functions

While it is possible to start an APL2 function dialog by using the INPUT keyword (as described above), many applications find it necessary to invoke additional APL2 functions as options (from a selection panel), as commands (from an application command table), or from other dialog functions. Such functions are selected by specifying the function request as the value of the SELECT CMD keyword, and once again, specifying LANG(APL). Since APL2 has already been started, and the APL2 environment established, the string is passed back to the APL2 workspace, and an APL2 “EXECUTE” function is performed on the string. For example, option 5 on a selection panel can be defined to APL2 function AVG (assuming that APL2 has already been started) as follows:

```
    .
    .
    .
5,'CMD(AVG 1 2 3 4 5) LANG(APL)'
    .
    .
    .
```

The return code for the selected function is passed back as a fullword of 0 (zero) if no terminating (to a quad-EA) APL2 errors have occurred. Otherwise, a fullword consisting of the quad-ET values in the two halfwords is returned.

APL2 cannot be invoked more than once, either within the same screen or on more than one screen. ISPF does nothing to prevent the second call. If APL2 is invoked a second time while running under ISPF, the results are unpredictable. Note that a user can utilize ISPF split-screen capabilities as long as APL2 is not invoked on a second screen.

Invoking ISPF Dialog Services in the APL2 Environment

A dialog service can be invoked by using the *function form* of ISPEXEC:

```
[n]   lastrc←ISPEXEC   character-vector
```

lastrc
Specifies the name of an APL2 variable in which the return code from the service is to be stored.

character-vector
Specifies a vector of characters that contains parameters to be passed to the dialog service. The format of the vector is the same as that for dialog service statements for command procedures written in CLIST.

A workspace containing the ISPEXEC function is provided with ISPF. All dialog writers must use this ISPEXEC function, as it contains the interface to ISPF and handles the implementation of commands (through the APL2 "EXECUTE" function); otherwise, results are unpredictable.

For example:

```
[1]     ⍝OPEN THE TABLE
[2]     LASTCC←ISPEXEC 'TBOPEN TABLE NOWRITE'
[3]     →(LASTCC = 0)/NORMALCONT
[4]     ⍝PROCESS ERRORS HERE
          .
[15]  NORMALCONT:.........
```

APL2 Workspace as the ISPF Function Pool

When an APL2 function invokes an ISPF dialog service, the APL2 workspace is considered to be the ISPF function pool. The dialog writer need not do anything special to make use of this mechanism. However, the following restrictions apply:

- Any variable retrieved or set is the most local to the currently executing APL2 function.
- The dialog writer should not use variables whose names begin with the three characters *ISP*; these names are reserved for ISPF. All variables used in the ISPEXEC function have names that start with these three characters.
- Only those variables whose names and formats fit both ISPF and APL2 protocols can be used for ISPF entities such as panels or tables:
  - All variable names must be 1 to 8 characters in length, composed of alphanumeric characters (A–Z, 0–9), of which the first must not be numeric. Note that #, $, and @ are not allowed.
  - All variable values must be simple character strings; APL2 general data types are not allowed. Note that the only acceptable null vector is that for character strings (‘’).
  - If an attempt is made to use a name or format incompatible with ISPF for an ISPF entity, a severe error occurs. Any APL2 name or format can be used within a dialog function, as long as that variable is not used for an ISPF entity.
  - Whenever an APL2 function is selected after APL2 is started, the original APL2 function pool (the APL2 workspace) is used. This implies that information can remain in the function pool from previous SELECTs, and the dialog writer must handle any such cases. Moreover, this rule is unaffected by SELECTs where new shared or profile pools are created; it is the responsibility of the dialog writer to ensure that the integrity of the workspace is maintained.
  - If the PDF component is installed, and the Dialog Test Variables option is requested, only those variables that have the correct name and format are displayed; if an attempt is made to enter a variable with a name that is not valid (to ISPF or APL2), an error occurs. The variables displayed are the most local to the currently executing function.

– A maximum of 64K bytes can be retrieved from the APL2 workspace during the execution of a DM service.

Interface between ISPF and APL2

The interface between ISPF and APL2 is like a telephone call. If one side of the communication is broken, any attempt to use the interface causes error messages to be generated. The link between the two products can be broken by:

- The APL2 user "hanging up". For example, if a new workspace is loaded and there are still ISPF service requests that have not completed (for example, options in the selection panel process), the ISPF Auxiliary Processor (ISPAPAUX) issues an error message, informs ISPF and waits for the process to begin again (by "hanging up" until another ISPF request is made). ISPF issues a severe error message telling the user that the link has been damaged.

  If the user is in ISPF TEST mode, then, on user request, ISPF attempts to reshow all panels traversed in an effort to unnest all service requests. When all requests have been unnested, ISPF will again wait for the ISPF Auxiliary Processor to make a request. During the unnesting process, any attempts to invoke APL2 functions are rejected, severe error messages are issued, and any requests for APL2 variables are logged.
- The APL2 user "cutting the line". For example, if the user terminates APL2 while there are still outstanding APL2 function requests from ISPF (for example, options in the selection panel process), the ISPF Auxiliary Processor (ISPAPAUX) issues an error message, informs ISPF, and terminates. ISPF issues a severe error message telling the user that the link has been damaged, and if in TEST mode, proceeds to unnest as described above. When all requests have been unnested, APL2 will be terminated. During the unnesting process, any attempts to invoke APL2 functions are rejected, severe error messages are issued, and any requests for APL2 variables are logged.
- An APL2 failure. This is handled as if the line were cut, assuming APL2 performs recovery and returns to ISPF.
- An ISPF failure. In this case, ISPF or the logical screen can fail, causing APL2 termination.

Subtasking Support

A dialog attached by ISPF, as described in "Invoking TSO Commands" on page 27, can invoke a dialog service. It does this by a call to either the ISPLINK or ISPEXEC interfaces from any subtask level. For subtasks to issue ISPF services, the program that attaches these subtasks must be invoked with the SELECT(*cmd*) service.

In addition, ISPF allows a task to detach its subtask at any time, even if an ISPF service invoked by that subtask is processing. The SUBTASK keyword of the CONTROL service, described in *ISPF Services Guide* , provides additional information. Multiple dialog services issued from multiple tasks executing asynchronously are not supported, and results will be unpredictable.

ESTAE Restrictions

Programs that code their own ESTAE routines should not issue ISPF services within the MVS ESTAE routine. Unpredictable results can occur. For more information on using MVS macros, refer to *MVS/XA Supervisor Services and Macros*

ISPF Services in Batch Mode

When initiated in a batch environment, ISPF services execute as a background command. Background calls are generally used to invoke ISPF table and file tailoring services. However, access to other dialog services is also available.

Command Processors in the TSO Batch Environment

TSO provides facilities for executing command processors in the batch environment. The JCL stream provides for data sets to be preallocated prior to the call of any command. Invoke the Terminal Monitor Program (TMP) using the EXEC statement to establish the necessary control blocks for the TSO environment. The command input stream is accessed from the SYSTSIN DD statement. All terminal line I/O outputs issued by the TSO I/O service routines are directed to the SYSTSPRT DD statement definition. Allocate ISPF libraries by using DD statements. Panel, message, 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.

To invoke ISPF, place the ISPSTART command in the SYSTSIN input stream with the PANEL, CMD, or PGM keywords that name the dialog to be invoked.

Note: When running on MVS with TSO/E Version 2 Release 1, ISPF does not read and execute the CLIST statements that follow the ISPSTART command. With ISPF running in batch (background) mode in the MVS environment with TSO/E Version 2 Release 1, you can select a CLIST procedure.

A user ID is selected for the background job as follows:

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

Although the user ID defaults to BATCH, the prefix used by ISPF when dynamically allocating a data set has no default. Therefore, a prefix should always be supplied on the TSO PROFILE command. At various times, ISPF attempts dynamic allocation and if no prefix has been supplied, allocation will fail and the job will ABEND. Multiple jobs executing concurrently must have unique prefixes.

The contents of positions 17–24 in system variable ZENVIR indicate whether ISPF is running interactively (TSO followed by five blanks) or background (BATCH followed by three blanks).

Sample Batch Job

Figure 16 on page 36 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.

```
* * * Top of File * * *
//ISPPLIB  DD DSN=ISP.SISPPENU,DISP=SHR
//ISPMLIB  DD DSN=ISP.SISPMENU,DISP=SHR
//ISPSLIB  DD DSN=ISP.SISPSENU,DISP=SHR
//         DD DSN=ISP.SISPSLIB,DISP=SHR
     .
     .
     .
//ISPTLIB  DD DSN=ISP.SISPTENU,DISP=SHR
//         DD DSN=ISP.SISPTLIB,DISP=SHR
     .
     .
     .
//SYSPROC  DD DSN=ISP.SISPEXEC,DISP=SHR
//         DD DSN=ISP.SISPCLIB,DISP=SHR
     .
     .
     .
//USERAA JOB (AA04,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.SISPPENU,DISP=SHR
//ISPMLIB  DD DSN=ISP.SISPMENU,DISP=SHR
//ISPSLIB  DD DSN=ISP.SISPSENU,DISP=SHR
//         DD DSN=ISP.SISPSLIB,DISP=SHR
//ISPLOG   DD DSN=USERAA.ISPF.LOG,DISP=SHR
//*
```

Figure 16. MVS Batch Job (Part 1 of 2)

```
//*- - - ALLOCATE TABLE DATA SETS - - - - - - - - - - - - */
//ISPTLIB  DD DSN=ISP.SISPTENU,DISP=SHR
//         DD DSN=ISP.SISPTLIB,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=ISP.SISPEXEC,DISP=SHR
//         DD DSN=ISP.SISPCLIB,DISP=SHR
//*
//*- - - ALLOCATE TSO BACKGROUND OUTPUT AND INPUT DS- - - */
//SYSTSPRT DD DSNAME=USERAA.ISPF.ISPFPRNT,DISP=SHR
//SYSTSIN  DD *
  PROFILE PREFIX(USERAA)         /* ESTABLISH PREFIX      */
  ISPSTART CMD(%TBUPDATE)        /* INVOKE CLIST DIALOG   */
/*
```

Figure 16. MVS Batch Job (Part 2 of 2)

Processing Errors

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

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.

Batch Display Facility for Background Panel Processing

The Batch Display Facility allows applications to simulate full-screen write operations while ISPF is executing in the background. This requires that dialogs provide the input to ISPF that would normally be supplied by the user or by information associated with the type of terminal being used. Much of this is done by having the dialog assign values to panel input variables, and by supplying screen size information through keywords on the ISPSTART command.

Batch execution has traditionally not allowed the use of services that require user interaction. Any full-screen write operation would result in an error condition.

The Batch Display Facility overcomes these limitations. Although there is no user interaction during execution; the Batch Display Facility does allow background execution of interactive services. These services include:
- DISPLAY
- TBDISPL
- SELECT PANEL
- SETMSG
- PQUERY.

These services are issued for batch just as they are issued for dialogs running in interactive mode. ISPF GDDM services do not run in the background, and thus, cannot be requested in a batch environment.

All ISPF commands except SPLIT and SPLITV can be executed in dialogs running in batch mode.

Installations can easily convert current interactive applications that use these services so they run in a batch environment. When you are running in a batch environment, you cannot direct your display to a workstation; that is, the GUI parameter on the ISPSTART command is not supported in a batch environment.

Supplying Input in Lieu of Interactive Users

When an application is running in batch, there is no user to respond to panel input operations. Therefore, the primary requirement for running interactive applications in batch is to supply expected input data by an alternate means. For example, panel variables can be given values by dialog function statements or by the processing specified in the panel's executable sections. This processing is begun in the batch environment as though a user had pressed Enter. In the absence of an alternative action on the dialog's part, ISPF assumes an ENTER condition following a panel display.

A dialog can override the ENTER condition and establish an END condition by doing one of the following:
- Using the .RESP control variable

- Setting the panel command field to END
- Issuing a CONTROL NONDISPL END prior to the display operation.

Supplying Batch Terminal Characteristics

In a batch environment there is no terminal from which ISPF can get screen width and screen depth values, so you must supply to ISPF data related to terminal type. You can include two optional keywords, BATSCRW and BATSCRD, on the ISPSTART command line to specify, respectively, screen width and screen depth values. The default values, if you do not include these keywords, are a screen width of 80 characters and a screen depth of 32 lines. The width and depth values, whether specified on the ISPSTART command or through the default values, establish the values in system variables ZSCREENW, ZSCREEND, ZSCRMAXW, and ZSCRMAXD.

In addition to the display services, use of the PQUERY service requires that the screen width and depth values be supplied to ISPF, either through default values or as defined on the ISPSTART command.

When running batch, terminal characteristics cannot be changed during a session, although some characteristics can be changed during an interactive session. For example, when ISPF is running interactively you can specify 3278 Model 5 and 3290 screen formatting. In batch mode, a dialog does not interact with a physical screen. Therefore, screen size, specified by including the BATSCRW and BATSCRD keywords on the ISPSTART command, is fixed for the duration of the batch session.

When running in batch mode, you can include the BDBCS keyword on the ISPSTART command. ISPF then processes the dialog as though it were running on a DBCS terminal.

The value in system variable ZCOLORS defines the number of colors (either 1 or 7) that a terminal can display. In batch mode, ISPF sets ZCOLORS to 1.

The value in system variable ZHILITE (YES or NO) determines if a terminal is to have extended highlighting capability, including underscore, blinking, and reverse video. In batch mode, ISPF sets ZHILITE to NO.

Message Processing in the Batch Environment

In an interactive environment ISPF displays two types of messages:

- Informational messages, normally those resulting from the MSG keyword specified on the SETMSG, DISPLAY, or TBDISPL service
- Error messages, including those resulting from the .MSG control variable in an executable panel section.

When running in a batch environment, ISPF writes any informational or error messages to the ISPF log data set at the processing point that the messages would normally be displayed to a user. The information logged includes the name of the panel associated with the message, followed by the short message and the long message.

A .MSG-initiated error message plus an ENTER condition causes a panel redisplay. In a batch environment, there is no interactive user to correct the error, so it must be handled by statements in the panel's)REINIT or)PROC sections. This leads to the possibility of a .MSG-redisplay loop if the error condition is not corrected.

Some panel language functions that can lead to this problem are VER, TRANS, ADDSOSI, DELSOSI, .MSG, and PANEXIT. To prevent this loop, a BREDIMAX keyword on the ISPSTART command is available to specify the maximum number (default 2) of redisplays. If this number of redisplays is exceeded, a severe error condition (return code 20) results and the related error message is written to SYSTSPRT.

Command Processing in the Batch Environment

ISPF processes most commands when running in the batch environment in the same way it processes them when running interactively, except the following:

- The SPLIT and SPLITV commands are disabled
- The ENVIRON, LOG, LIST, ISPPREP, KEYS, ZKEYS, and PFSHOW TAILOR commands can result in display loops.

Display Error Processing in the Batch Environment

When ISPF is running interactively with CONTROL ERRORS CANCEL in effect, a return code of 12 or higher causes the ISPF error panel to display. These same conditions in the batch environment cause the error panel message to be written to the SYSTSPRT data set, after which ISPF terminates. In the interactive or batch environment with CONTROL ERRORS RETURN in effect, control returns to the dialog for error processing following a return code of 12 or higher.

How ISPF Handles Log and List Data Sets in the Batch Environment

If ISPF allocates a log or list data set in the batch environment, it is always kept at termination, regardless of the disposition specified on SETTINGS Option 0.

Avoiding Panel Loop Conditions in the Batch Environment

When writing new dialogs or altering existing dialogs to run in the batch environment, dialog developers must be very careful not to create functions that result in looping situations since user input is expected and none is supplied. See "Supplying Input in Lieu of Interactive Users" on page 37 for more information. For example, issuing the ISPPREP command causes ISPF to call an interactive ISPPREP dialog, which will cause a loop condition in a batch environment. Therefore, you should invoke the non-interactive ISPPREP facility directly by using the SELECT PGM(ISPPREP) service request as described for batch mode under Figure 54 on page 156.

The KEYS command can cause a loop condition because its processing termination depends on an END or RETURN command. An ENTER condition, which ISPF assumes in absence of an END or RETURN being forced, results only in another panel display, which leads to a loop condition.

To help deal with possible looping situations, the BDISPMAX keyword on the ISPSTART command is available for specifying the maximum number of panel displays that can occur during a session. The default value is 100. If the number specified in BDISPMAX is exceeded, a severe error condition (return code 20) results and an error message, stating that the maximum number of displays has been exceeded, is written to the SYSTSPRT data set.

ISPF Graphical User Interface in Batch Mode

OS/390 Version 2 Release 5.0 provides the capability to run the ISPF Client/Server (C/S) component in a batch environment. You can start ISPF using the GUI parameter to enable a C/S session to run on a specific workstation without tying up the invoking session.

This function also enables you to capture REXX trace output (in SYSTSPRT), and to invoke ISPF without a 3270 terminal, such as through an icon on the workstation through APPC or TCP/IP, or through a Telnet linemode session.

Restrictions

When using the batch mode capabilities of the C/S, be sure to consider these restrictions:

- The number of initiators on the batch machine. Because the JCL remains resident for the duration of the session, you should be aware that you have reduced the number of available initiators for other uses.
- The limitation of the C/S Server to 30 sessions.

  Note: Because it is based on JAVA, this 30 session limit does not apply when using the ISPF Application Server and Workstation Agent Applet.
- Each batch session must have a unique profile (just like each user ID).
- The PA1 key is not available within the GUI environment.
- Full-screen TPUT function is not supported.
- Because you are in batch mode, and therefore you are *not* using a TSO emulator, GDDM is disabled and TSO line-mode output is not available.
- Other TSO batch limitations. Some commands might not be supported in GUI Batch mode because of the inability to provide terminal input to them (such as RECEIVE). Refer to the *TSO User's GUIDE* for more information about running TSO in batch mode.

Example JCL

The JCL that follows does not take into consideration the conditions present when you use the ISPF Application Server and Workstation Agent Applet to make your GUI connections. For more information about the server and applet, refer to the *OS/390 ISPF Application Server Guide and Reference*.

The JCL job that follows can be run from your MVS session to invoke ISPF running the Client/Server in batch mode. Before submitting this JCL job, you must update it with this information:

- The jobcard information in line 1 must be furnished, and a unique jobname must be used.
- Update "userid.PRIVATE" with your private libraries, if needed. If you do not use private libraries, remove those data sets from the concatenation.
- If your ISPF product data sets are not named "ISP.SISxxxx", update the data set names.
- Update the TSO profile.
- Update the ISPSTART invocation with the session title.
- Update the GUI() keyword for either TCPIP (your_ip_address) or APPC (your_lu_name), and remove the other keyword.

```
//userid0    your jobcard information here
//* JCL TO RUN ISPF IN BATCH
//WSGUI   EXEC PGM=IKJEFT01,REGION=4096K,TIME=1439,DYNAMNBR=200
//STEPLIB  DD  DSN=ISP.SISPLPA,DISP=SHR
//         DD  DSN=ISP.SISPLOAD,DISP=SHR
//         DD  DSN=ISP.SISPSASC,DISP=SHR
//         DD  DSN=userid.PRIVATE.LOAD,DISP=SHR
//ISPLLIB  DD  DSN=ISP.SISPLPA,DISP=SHR
//         DD  DSN=ISP.SISPLOAD,DISP=SHR
//         DD  DSN=ISP.SISPSASC,DISP=SHR
//         DD  DSN=userid.PRIVATE.LOAD,DISP=SHR
//SYSEXEC  DD  DSN=userid.PRIVATE.EXEC,DISP=SHR
//         DD  DSN=ISP.SISPEXEC,DISP=SHR
//SYSPROC  DD  DSN=userid.PRIVATE.CLIST,DISP=SHR
//         DD  DSN=ISP.SISPCLIB,DISP=SHR
//ISPMLIB  DD  DSN=userid.PRIVATE.MSGS,DISP=SHR
//         DD  DSN=ISP.SISPMENU,DISP=SHR
//ISPPLIB  DD  DSN=userid.PRIVATE.PANELS,DISP=SHR
//         DD  DSN=ISP.SISPPENU,DISP=SHR
//ISPSLIB  DD  DSN=userid.PRIVATE.SKELS,DISP=SHR
//         DD  DSN=ISP.SISPSLIB,DISP=SHR
//         DD  DSN=ISP.SISPSENU,DISP=SHR
//SYSIN    DD  DUMMY,DCB=(LRECL=120,BLKSIZE=2400,DSORG=PS,RECFM=FB)
//SYSUADS  DD  DSN=SYS1.UADS,DISP=SHR
//SYSHELP  DD  DSN=SYS1.HELP,DISP=SHR
//ISPCTL0  DD  UNIT=SYSDA,
//             SPACE=(TRK,(5,5)),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=6160),
//             DISP=(,DELETE,DELETE)
//ISPCTL1  DD  UNIT=SYSDA,
//             SPACE=(TRK,(5,5)),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=6160),
//             DISP=(,DELETE,DELETE)
//ISPCTL2  DD  UNIT=SYSDA,
//             SPACE=(TRK,(5,5)),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=6160),
//             DISP=(,DELETE,DELETE)
//ISPWRK0  DD  UNIT=SYSDA,
//             SPACE=(TRK,(5,5)),
//             DCB=(RECFM=FB,LRECL=256,BLKSIZE=2560),
//             DISP=(,DELETE,DELETE)
//ISPWRK1  DD  UNIT=SYSDA,
//             SPACE=(TRK,(5,5)),
//             DCB=(RECFM=FB,LRECL=256,BLKSIZE=2560),
//             DISP=(,DELETE,DELETE)
//ISPWRK2  DD  UNIT=SYSDA,
//             SPACE=(TRK,(5,5)),
//             DCB=(RECFM=FB,LRECL=256,BLKSIZE=2560),
//             DISP=(,DELETE,DELETE)
//ISPLST0  DD  UNIT=SYSDA,
//             SPACE=(TRK,(5,5)),
//             DCB=(RECFM=FBA,LRECL=121,BLKSIZE=1210),
//             DISP=(,DELETE,DELETE)
//ISPLST1  DD  UNIT=SYSDA,
//             SPACE=(TRK,(5,5)),
//             DCB=(RECFM=FBA,LRECL=121,BLKSIZE=1210),
//             DISP=(,DELETE,DELETE)
//ISPLST2  DD  UNIT=SYSDA,
//             SPACE=(TRK,(5,5)),
//             DCB=(RECFM=FBA,LRECL=121,BLKSIZE=1210),
//             DISP=(,DELETE,DELETE)
//ISPTABL  DD  DSN=userid.PRIVATE.TABLES,DISP=SHR
//ISPTLIB  DD  DSN=userid.PRIVATE.TABLES,DISP=SHR
//         DD  DSN=ISP.SISPTENU,DISP=SHR
//SYSUDUMP DD  DUMMY
//ISPLOG   DD  SYSOUT=T,
//             DCB=(RECFM=VA,LRECL=125,BLKSIZE=129)
```

```
//ISPPROF  DD  DSN=userid.PRIVATE.TABLES,DISP=SHR
//SYSTSPRT DD  DSN=userid.PRIVATE.TSOOUT,DISP=SHR
//*SYSTSPRT DD  SYSOUT=(*)
//SYSPRINT DD  SYSOUT=(*)
//SYSOUT   DD  SYSOUT=(*)
//SYSTSIN  DD  *
  PROFILE PREFIX(profile)
  PROFILE NOPROMPT
  ISPSTART PANEL(ISR@PRIM) NEWAPPL(ISR)  TITLE(your_session_title) +
    GUI(IP:your_ip_address)  or   GUI(LU:your_lu_name)
```

Chapter 3. Introduction to Writing Dialogs

This chapter provides introductory information that explains how to use the ISPF display, variable, table, file-tailoring, PDF, and other miscellaneous services to write dialogs.

For more detailed information on using these services, refer to the *ISPF Services Guide*

Using the Display Services

The display services allow a dialog to display information and interpret responses from users. The display services are:

ADDPOP Start pop-up window mode. The ADDPOP service specifies that the following panel displays are to be in a pop-up window. It also identifies the location of the pop-up window on the screen in relation to the underlying panel or window.

DISPLAY Display a panel. The DISPLAY service reads a panel definition from the panel files, initializes variable information in the panel from the corresponding dialog variables in the function, shared, or profile variable pools, and displays the panel on the screen. Optionally, the DISPLAY service might superimpose a message on the display.

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

The COMMAND option on the DISPLAY service allows a dialog to pass a chain of commands to ISPF for execution. This option is explained fully in the *ISPF Services Guide* Use of the DISPLAY service is illustrated in a function example later in this chapter.

LIBDEF Define optional search libraries. The LIBDEF service allows users to define an optional, application-level set of libraries containing, for example, messages or panels, to be searched before the IBM-supplied ISPF libraries. Refer to the *ISPF Services Guide* for more information.

REMPOP Remove a pop-up window. The REMPOP service call removes a pop-up window from the screen.

SELECT Select a panel or function. The SELECT service is used to display a hierarchy of selection panels or invoke a function.

SETMSG Display a message on the next panel. The SETMSG service constructs a specified message from the message file in an ISPF system save area. The message will be superimposed on the next panel displayed by any DM service. The optional COND parameter allows you to specify that the message is to be displayed on the next panel only if there is no SETMSG request pending.

TBDISPL Display a table. The TBDISPL service combines information from

© Copyright IBM Corp. 1980, 1999

panel definitions with information stored in ISPF tables. It displays selected rows from a table, and allows the user to identify rows for processing.

Panel definitions used by the TBDISPL service contain nonscrollable 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. For more information on TBDISPL, see “Defining Table Display Panels” on page 134 and the description of the TBDISPL service in *ISPF Services Guide*

Example: Creating a Display with TBDISPL

The TBDISPL service displays information from an ISPF table on a panel formatted by information on a panel definition. Table 1 illustrates an ISPF table named TAB1.

Table 1. TBDISPL – ISPF Table

RANK	ID	CITY	STATE	POPCH	ROW
1	FLO621	Fort Myers	fl	+95.1	r1
2	NV1235	Las Vegas	nv	+69.0	r2
3	FL1972	Sarasota	fl	+68.0	r3
4	COO649	Fort Collins	co	+66.0	r4
5	FL2337	West Palm Beach	fl	+64.3	r5
6	FLO608	Fort Lauderdale	fl	+63.6	r6
7	TXO231	Bryan	tx	+61.5	r7
8	NV1833	Reno	nv	+60.0	r8
9	UT1656	Provo	ut	+58.4	r9
10	TX1321	McAllen	tx	+56.1	r10

Figure 17 illustrates a panel definition named PAN1.

```
************************************************************
* )Attr                                                    *
*   @ Type(output) Intens(low) Just(asis)   Caps(off)      *
* )Body                                                    *
* -------------------- Population Change ----------------- * ---┐
* +Command ==>Cmdfld                    +Scroll ==>_samt+  *    │
* +                                                        *    │
* This table shows selected metropolitan areas which had a *    │---> (A,Figure 18)
*                                                          *    │
* large relative increase in population from 1970 to 1980. *    │
*                                                          *    │
* +Metro area      State     Change                        *    │
* +                         (Percent)                      * ---┘
* )Model                                                   *
* @City            @State  @popchg+                        * -------> (B,Figure 18)
*                                                          *
* )Init                                                    *
*   &samt=page                                             *
* )Proc                                                    *
* )End                                                     *
************************************************************
```

Figure 17. TBDISPL Panel Definition

The)BODY section of PAN1 defines the fixed portion of the display, area “A” in Figure 17 on page 44. The)MODEL section of PAN1 produces the scrollable portion of the display, area “B” in Figure 17 on page 44.

There can be up to eight model lines. Panel PAN1 has only one. The scrollable portion of the display is formed by replicating the model lines to fill the screen. Each of these replications, as well as the original, is known as a model set. Each model set corresponds to a table row. Table rows are then read to fill in the appropriate fields in the model set replications.

PAN1 displays only three (city, state, and popchg) of the five columns of table TAB1. The model lines can include any number of the KEYS, NAMES, and extension variables of the table. They can also include fields that are not variables in the table. Figure 18 shows the effect of displaying information from TAB1 on panel PAN1.

```
                 +-------------------------------------------------------+
        +------- |------------------- Population Change ------ ROW 4 OF 10 |
        |        | Command ==>                             Scroll ==> Page   |
        |        |                                                           |
        |        |                                                           |
-- (A)|        |This table shows selected metropolitan areas which had a |
        |        |large relative increase in population from 1970 to 1980. |
        |        |                                                           |
        |        |  Metro area        State        Change                    |
        -------- |                                (Percent)                  |
        +--r4--  |  Fort Collins       co          +66.0                    |
        |  r5--  |  West Palm Beach    fl          +64.3                    |
        |  r6--  |  Fort Lauderdale    fl          +63.6                    |
-- (B)|  r7--  |  Bryan              tx          +61.5                    |
        |  r8--  |  Reno               nv          +60.0                    |
        |  r9--  |  Provo              ut          +58.4                    |
        |  r10-  |  McAllen            tx          +56.1                    |
        +------- |********************** BOTTOM OF DATA ****************** |
                 +-------------------------------------------------------+
```

Figure 18. TBDISPL Display

When the TBDISPL service is invoked with the panel name specified, the scrollable portion begins with the current row. That is, the current row is the top row displayed. In this example, the current row pointer (CRP) for table TAB1 has been set to row 4. Table rows are read starting with row 4 to fill in the appropriate fields in the model set replications. If there were any non-table variables in the model line, they would be filled in with their current values. Because there aren’t enough rows in the table to fill the screen, the bottom-of-data marker is placed in the display after the last row. The “empty” model sets beyond this marker are not displayed.

In Table 1 on page 44, the symbols r1 through r10 label the 10 rows in the table TAB1. The highlighted rows, r4 through r10, indicate that these rows provide the information for the scrollable portion of the display (marked as area B in Figure 18).

Figure 18 is the result of using the TBDISPL service with panel definition PAN1 (Figure 17 on page 44) and ISPF table TAB1 (Table 1 on page 44). Portion A is the fixed portion defined by the)BODY section of PAN1. Portion B is the scrollable portion defined by the)MODEL section of PAN1. The table information in the display is the specified columns from row 4 to row 10.

Processing Selected Rows

When a user changes data in a model set, the corresponding table row is said to be selected for processing. More than one row can be selected in a single interaction. Before the TBDISPL service returns control to the dialog function, the CRP is positioned to the first of the selected rows. *First* means the row closest to the top of the table, not the row that was selected first. The other selected rows are called pending selected rows.

When the CRP is positioned at a selected row, the row is retrieved, meaning the values from that row are stored in the appropriate dialog variables. Then, all input fields in the selected model set on the display are stored in the corresponding dialog variables.

The dialog function can then process the row in any manner it chooses. For example, the function can invoke the TBPUT service to update the row, or it can invoke the BROWSE service to examine a file specified in that row.

A call of the TBDISPL service is required to position the CRP to each pending selected row. For these calls, neither the PANEL nor MSG parameter should be specified.

The system variable ZTDSELS contains the number of selected rows. It can be tested by the dialog function or in the)PROC section of the table display panel to determine if any rows were selected. For example:

```
)PROC
  . . .                    /* Process fixed portion fields   */
  IF (&ZTDSELS ¬= 0000)    /* Any selected rows?             */
    . . .                  /* Process scrollable portion flds*/
)END
```

The interpretation of this variable is as follows:

0000 No selected rows

0001 One selected row (now the current row)

0002 Two selected rows, consisting of the current row and a pending selected row

0003 Three selected rows, consisting of the current row and two pending selected rows

. .

. .

. .

n “n” selected rows, consisting of the current row and “n-1” pending selected rows.

As TBDISPL is reinvoked without the PANEL and MSG parameters (to process any pending selected rows), ZTDSELS is decremented by one. An example is shown in Table 2 on page 47.

Table 2. ZTDSELS Decrementation

DM Service	User Action	Value of ZTDSELS
TBDISPL TAB1 PANEL(PAN1)	Selects 3 rows	0003 (current row plus two pending selected rows)
TBDISPL TAB1	None	0002 (current row plus one pending selected row)
TBDISPL TAB1	None	0001 (current row; no pending selected rows)

Adding Table Rows Dynamically during Table Display Scrolling

Assume that you have access to a large amount of related data that might be built into a single table. However, you need to interface with only a subset of that data during an ISPF session, but you are not sure just how extensive that subset is. Normally, you would have to initially construct a table that included all possible data that you might wish to access during a session before you began scrolling and update activity on the table. This could lead to a great deal of unnecessary overhead because you might include a lot of data in your table that you never access.

By interacting with a set of function system variables, an ISPF function can dynamically expand the table as you scroll through it during a session. The function can specify that the table is to be expanded upward when the user has scrolled past the top, expanded downward when the user has scrolled past the bottom, or both. In this way, the function adds only the table rows that satisfy your needs as you need them.

System Variables Are the ISPF-Function Interface

Eight system variables in the function pool are the vehicle for passing, between ISPF and the function, values that control table expansion. These variables and the functions they perform are:

ZTDRET (input; length 8)
The function sets variable ZTDRET in the function pool to a value (UP, DOWN, or VERTICAL) that indicates to ISPF when control is to return to the function so that more rows can be added to the table being processed.

ZTDADD (output; length 3)
ISPF sets this variable to either YES or NO before returning control to the function. A value of YES indicates that the function needs to add more rows to the table being processed. ZTDADD is normally set to NO, indicating that no more rows need to be added to the table.

ZTDSCRP (input/output; length 6)
This variable is set to the row pointer (number of the row relative to the top of the table) of the row that is to be at the top of the panel's scrollable area after the scroll request is processed. If ISPF cannot determine this value, this variable is set to zero.

ZTDSRID (output; length 6)
ISPF sets this variable to the row ID of the row pointed to by the value in variable ZTDSCRP. During table processing, the row pointer value for a given row can change. However, the row ID of that row does not change.

ZTDAMT (output; length 4)
When ISPF returns control to the function with the value of variable ZTDADD set to YES, the value that ISPF has set in variable ZTDAMT tells the function how many rows, based on the information available, ISPF calculates should be added to the table to satisfy the current scroll request.

ZTDSIZE (output; length 4)
ISPF sets the value of ZTDSIZE to the total number of model sets; that is, the number of table rows that fill the scrollable area of the panel. This is not necessarily the same as the number of lines displayed in the panel's scrollable area.

ZTDLTOP (input; length 6)
The function can optionally set this variable to a value for ISPF to use in calculating the value *x* (top-row-displayed) in the indicator 'ROW x OF y', which ISPF displays on a TBDISPL screen.

ZTDLROWS (input; length 6)
The function can optionally set this variable to a value for ISPF to use as the value *y* (total rows in the logical table) in the indicator 'ROW x OF y'.

You can define variables ZTDAMT, ZTDSCRP, ZTDSRID, ZTDSIZE, ZTDLTOP, and ZTDLROWS as fullword fixed binary in a program function. If you do not, the default for each of these variables is character with lengths as specified in the system variable charts in the "Appendix E. System Variables" on page 361.

Dynamic Table Building: To put the dynamic table building concept into practice, a function first builds a basic table structure. The initial size of this table is determined by balancing the minimum amount of table data that would satisfy most anticipated user needs against the overhead of including a large amount of table data to cover more contingencies. As more table rows are needed to satisfy scroll requests, ISPF returns control to the function so that it can add those rows.

When a user issues a scroll request, there might be input fields in a panel that have been typed into (selected for processing). In that case, the dialog first processes all selected rows and then issues a TBDISPL request, without panel name, to cause the panel to redisplay. If no table rows are needed to fill the scroll request, ISPF completes the scroll and redisplays the panel. If more table rows are needed to fill the scroll request, ISPF returns control to the function to add table rows. Keep in mind that each time control returns to the function, the)PROC section of the panel from which the table display was requested is executed. After adding the table rows, the function issues a TBDISPL without a panel name to complete the scroll and redisplay. Remember, specifying a panel name on a TBDISPL request nullifies any pending selected rows or request for scrolling.

The values of a set of system variables in the function pool are the parameters used in the interchange between ISPF and a function when dynamically increasing the table size.

Using Variable ZTDRET

The need for expanding a table occurs when a user scrolls beyond the top or bottom of the table while using the TBDISPL service. The function must set variable ZTDRET to a value that tells ISPF when to return control so the function can expand the table. The function sets ZTDRET to one of three possible values:

UP Control returns to the function when the top of the scrollable data

is reached. This applies when you are building the table upward from the bottom. The value UP has no effect when the bottom of the scrollable data is reached.

DOWN Control returns to the function when the bottom of scrollable data is reached. This applies when you are building the table downward from the top. The value DOWN has no effect when the top of the scrollable data is reached.

VERTICAL Control returns to the function when the top or bottom of the scrollable data is reached.

The value in ZTDRET must be left-justified (no leading blanks). ISPF evaluates the value of ZTDRET only when the function issues a TBDISPL request *with a panel name specified*. This is true, even though in the interim, the function might change the value of ZTDRET and issue TBDISPL requests without a panel name specified. A TBDISPL request with a panel name specified also nullifies processing of any pending selected table rows and any pending scroll request.

When a scroll request is pending, a TBDISPL request with a message ID specified (but without a panel name specified) causes the panel to be redisplayed with the message, but the scroll request is nullified.

Using Variable ZTDADD

Before returning control to a function from a TBDISPL request, ISPF sets function variable ZTDADD to YES or NO, indicating to the function whether rows are to be added to the table. The function normally receives a return code of 0 from the TBDISPL service. It can then interrogate variable ZTDADD. If its value is 'YES', then ZTDSCRP, ZTDSRID, ZTDAMT, and ZTDSIZE contain valid values.

ISPF normally returns control to the function for reasons other than to add table rows. In those cases, ISPF sets the value of ZTDADD to NO. For example, the function might need to interact with table rows that have been selected for processing during a table display.

Using Variable ZTDAMT

When ISPF returns control to a function with variable ZTDADD set to YES, the function must add rows to the table. If rows must be added to the table to satisfy a scroll request, ISPF calculates, when possible, the number of rows that need to be added to the table and returns that value to the function in variable ZTDAMT. The function should use this value for determining the number of rows to add.

For some scroll requests, such as UP MAX or DOWN MAX, ISPF cannot determine the number of rows to be added to the table. In those cases, ISPF returns a value of 0 to the function in ZTDAMT.

Using Variables ZTDSCRP and ZTDSRID

When ZTDSCRP contains a value other than 0, that value is the number of the table row that is to be at the top of the panel's scrollable area when the panel is redisplayed. ISPF sets ZTDSCRP to a non-zero value if a user has requested a downward scroll such that, when ISPF redisplays the panel following the scroll, the top row displayed in the scrollable area existed in the table at the time of the scroll request.

When the user requests an UP MAX or DOWN MAX, ISPF does not require the ZTDSCRP value to position the table when it is redisplayed following the scroll. It simply positions the table in the scrollable display area relative to the top table row (UP MAX) or the bottom-of-data marker (DOWN MAX).

For other scroll requests that require that rows be added to the table, ISPF may not be able to determine what the value of ZTDSCRP should be. In other words, one of the table rows to be added by the function will be the new top row displayed. ISPF has no way of knowing what the number of that row will be. In those cases, ISPF returns a value of 0 to the function.

If a function receives a value of 0 in ZTDSCRP (other than for UP MAX or DOWN MAX), it must set the variable's value to the number of the new table row that should display at the top of the panel's scrollable area. When the function sets the value of ZTDSCRP, the developer must take into account that the number specified is the number of the top displayed table row relative to the top of the table as the user who issued the scroll requests will see it. The developer must also take into account any processing that takes place from the time the user requests a scroll to the time the scroll is processed. For example, assume that variable ZTDRET is set to UP. A user issues:

```
UP 10
```

but there are only eight table rows above the top one currently displayed. ISPF returns control to the function with variable ZTDAMT having a value of 2, indicating that two lines must be added to the table to satisfy the current scroll request. ISPF has set variable ZTDSCRP to 0 because the new top displayed row did not exist in the table when the scroll was requested. Assume that, instead of adding only the two required table rows at the top of the table to satisfy this scroll request, the function adds 20 rows as a cushion against additional scrolling. Therefore, the function must set ZTDSCRP to 19 so that ISPF will redisplay the panel with the table positioned as the user wants it.

In addition to the row pointer in variable ZTDSCRP, ISPF returns to the function in variable ZTDSRID the identification (rowid) of the row that is to be displayed at the top of the scrollable area. As just described for ZTDSCRP, if ISPF cannot determine which is to be the top row displayed, it returns a value of 0 in ZTDSRID.

Using Variable ZTDSIZE

When ISPF returns control to the function to add more rows to a table, variable ZTDSIZE contains the *total* number of table rows that can fit into the entire panel scrollable area. Changes made to the panel structure, such as by PFSHOW ON or split-screen mode, do not affect this value. The value is the total number of scrollable area rows.

Using Variables ZTDLTOP and ZTDLROWS

ISPF displays in the upper-right corner of a TBDISPL panel a default top-row-displayed indicator, 'ROW x OF y', where *x* is the current row pointer of the top row displayed, and *y* is the total number of rows in the physical table being displayed. By assigning a message ID to system variable ZTDMSG, a function can specify a message whose short message text is to replace the top-row-displayed indicator. However, keep in mind that in the following text all references to the top-row-displayed indicator refer to the default supplied by ISPF, not an alternate indicator specified by the application.

Because the dimensions of only the physical table are available, ISPF has no way of assuring what the x and y values for the top-row-displayed indicator should be. Therefore, it is the application's responsibility to pass to ISPF the logical table positioning in variables ZTDLTOP and ZTDLROWS, respectively, any time control returns to the function to add table rows. If the function does not set these variables to a value, ISPF calculates the x and y values according to the size and position of the table being displayed.

For example, assume that, to satisfy scroll requests, an application is adding records dynamically to a table from a 1000-record file. The application initially builds the table with records 500 through 520. To pass these values to ISPF for use as the x and y values in the top-row-displayed indicator, the application function sets ZTDLTOP to 500 and sets ZTDLROWS to 1000. This causes the indicator text 'ROW 500 OF 1000' to be displayed initially on the TBDISPL panel. Then assume that the user scrolls down 10 rows. ISPF, using the value in ZTDLTOP plus the 10 rows scrolled, changes the indicator to 'ROW 510 OF 1000'.

In the example just described, assume that the user initially scrolled up 10 rows instead of down 10 rows. Because the top row displayed was the top table row, control returns to the application function to add rows to the top of the table so the scroll request can be completed. As mentioned above, it is the application's responsibility to change the values of ZTDLTOP and ZTDLROWS as needed to provide ISPF an accurate base for generating the top-row-displayed indicator. Therefore, after adding rows to the top of the table, the function sets variable ZTDLTOP to 490 before issuing the TBDISPL request to redisplay the table. The text of the top-row-displayed indicator on the displayed panel is now 'ROW 490 OF 1000'.

Example: Dynamic Table Expansion

This example illustrates how you can use dynamic expansion to reduce the initial overhead of creating a large table for display.

Assume that you are given the task of creating an ISPF dialog that allows a user to browse through a list of invoices for a given year. The list is maintained in a sequential file, and it contains information (invoice number, transaction date, part number, quantity, and customer name) for each transaction made during the year.

The file is fixed-block with a logical record length of 80 and a block size of 6160. The first record in the file contains the year and the number of invoices that follow in the file.

The format of this record is as follows:

Positions	Format
1–4	Year
5–10	Number of invoices
11–80	Reserved

The format of each of the invoice records is as follows:

Positions	Format
1–6	Invoice number
7–14	Transaction date (format mm/dd/yy)
15–18	Part number
19–21	Quantity (right justified)
22–46	Customer name (left justified)

47–80 Reserved

For example, the file might look something like this:

```
1986010000
00000101/06/867071100Acme Auto
00000201/06/860015 15Parts City
00000301/07/861023340Cary Auto Center
00000401/08/860231  1Parts Unlimited
00000501/08/863423805Bosworth's Parts
00000601/08/862341165Acme Parts
00000701/08/867653 20Acme Parts
00000801/08/863353100Bosworth's Parts
00000901/08/860003325Bosworth's Parts
00001001/08/863322  1Bosworth's Parts
⋮

00999912/15/860325 43ABC Parts
01000012/18/864234340ACME Parts
```

As you can see, the file is in no form to be browsed as it is. One way to implement the dialog is to transfer the invoice file to a temporary ISPF table, and then display the table with the TBDISPL service. However, since the number of invoices can be relatively high (in this example, there are 10 000 invoices), the initial overhead of reading every record and adding it to the table is unacceptable. As an alternative, the dialog uses dynamic table expansion instead. Using this method, it adds only the first 60 invoices to the table initially. Other invoices are added on an as-needed basis as the user scrolls through the table. The user sees no evidence that only a portion of the invoices are in the table.

Figure 19 shows the definition for panel INVPANEL, which the dialog uses to display table rows.

```
)Attr
 @ Type(Output) Intens(Low)
)Body Expand(//)
+-/-/-%&year TRANSACTIONS+-/-/-
%Command ====>_cmd                                  %Scroll ===>_amt +
+
+
%Invoice    Transaction    Part
%Number        Date        Number    Quantity    Customer
%-------    -----------    ------    --------    --------
)Model
@inv        @date         @part       @qty      @cust                +
)Init
 &amt = PAGE
)End
```

Figure 19. Panel Definition Dynamic Table Expansion

The PL/I dialog function, INVOICE (shown in Figure 20 on page 53), requires that the invoice file be allocated to ddname INVFILE prior to execution of the dialog. The intent of this example is to illustrate the dynamic expansion function. Normal error checking and error processing is not shown, but should be included in all dialogs.

```
INVOICE: PROC OPTIONS(MAIN);
 /*******************************************************************/
 /*  THIS PROGRAM ILLUSTRATES THE USE OF DYNAMIC EXPANSION WITH     */
 /*  THE TABLE DISPLAY SERVICE.  THE PROGRAM READS RECORDS FROM A   */
 /*  SEQUENTIAL FILE CONTAINING A LIST OF INVOICES AND ADDS THE     */
 /*  INVOICE INFORMATION TO A TEMPORARY ISPF TABLE (INVTABLE).      */
 /*  THE TABLE IS THEN DISPLAYED SO THAT THE USER CAN BROWSE        */
 /*  THROUGH THE INVOICES.  THE FOLLOWING STEPS ARE PERFORMED BY    */
 /*  THE PROGRAM:                                                   */
 /*                                                                 */
 /*    1. DEFINE THE FUNCTION POOL VARIABLES FOR THE TEMPORARY      */
 /*       TABLE, THE TBDISPL SYSTEM VARIABLES, AND MISCELLANEOUS    */
 /*       VARIABLES.                                                */
 /*                                                                 */
 /*    2. ISSUE A TBCREATE SERVICE CALL FOR TEMPORARY TABLE,        */
 /*       INVTABLE.                                                 */
 /*                                                                 */
 /*    3. OPEN FILE INVFILE AND READ THE HEADER RECORD INTO THE     */
 /*       HEADER_RECORD STRUCTURE.                                  */
 /*                                                                 */
 /*    4. READ EACH OF THE FIRST 60 INVOICE RECORDS FROM INVFILE    */
 /*       INTO THE INVOICE_RECORD STRUCTURE AND ADD THEM TO TABLE   */
 /*       INVTABLE.  USE THE TBADD MULT PARAMETER TO OPTIMIZE       */
 /*       TBADD ROW STORAGE MANAGEMENT.                             */
 /*    5. ISSUE A TBTOP SERVICE CALL TO POSITION THE CRP AT THE     */
 /*       TOP OF INVTABLE.                                          */
 /*    6. INITIALIZE SYSTEM VARIABLE ZTDRET TO "DOWN"               */
 /*       AND SYSTEM VARIABLE ZTDLROWS TO THE NUMBER OF INVOICES    */
 /*       IN THE FILE.                                              */
 /*    7. ISSUE A TBDISPL SERVICE CALL THAT REFERS TO TABLE         */
 /*       INVTABLE AND PANEL INVPANEL.                              */
 /*    8. LOOP WHILE THE TBDISPL SERVICE RETURN CODE IS LESS THAN   */
 /*       8 (WHILE THE USER HAS NOT ISSUED THE END COMMAND AND      */
 /*       WHILE THERE HAVE BEEN NO SEVERE ERRORS).  ON RETURN       */
 /*       FROM THE TBDISPL SERVICE, DO THE FOLLOWING:               */
 /*                                                                 */
 /*        - CHECK TO SEE IF ADDITIONAL ROWS ARE NEEDED TO          */
 /*          SATISFY A SCROLL REQUEST.                              */
 /*        - IF ADDITIONAL ROWS ARE NEEDED, READ THE APPROPRIATE    */
 /*          NUMBER OF INVOICES FROM INVFILE AND ADD THEM TO        */
 /*          INVTABLE AGAIN USING THE TBADD MULT PARAMETER.         */
 /*        - IF NECESSARY, SET THE SYSTEM VARIABLE ZTDSCRP TO       */
 /*          THE CRP OF THE NEW TOP ROW.                            */
 /*        - FINALLY, ISSUE A TBDISPL SERVICE CALL (WITHOUT A       */
 /*          PANEL NAME) TO REDISPLAY INVTABLE.                     */
 /*    9. PERFORM SOME FINAL CLEANUP BEFORE EXITING THE DIALOG:     */
 /*                                                                 */
 /*        - ISSUE A TBEND SERVICE CALL TO CLOSE AND DELETE         */
 /*          INVTABLE.                                              */
 /*        - CLOSE INVFILE.                                         */
 /*        - ISSUE A VDELETE SERVICE CALL TO DELETE ALL FUNCTION    */
 /*          POOL VARIABLES CREATED BY THE DIALOG.                  */
 /*******************************************************************/
```

Figure 20. PL/I Dialog Function Example Program (Part 1 of 7)

```
DECLARE                                  /*                       */
  1 HEADER_RECORD,                       /* HEADER RECORD FIELDS  */
    3 YEAR      CHAR(4),                 /*   YEAR OF INVOICES    */
    3 NUM_RECS  CHAR(6),                 /*   NUMBER OF INVOICES  */
    3 FILLER    CHAR(70);                /*   ** RESERVED **      */
                                         /*                       */
DECLARE                                  /*                       */
  1 INVOICE_RECORD,                      /* INVOICE RECORD FIELDS */
    3 INV       CHAR(6),                 /*   INVOICE NUMBER      */
    3 DATE      CHAR(8),                 /*   TRANSACTION DATE    */
    3 PART      CHAR(4),                 /*   PART NUMBER         */
    3 QTY       CHAR(3),                 /*   QUANTITY            */
    3 CUST      CHAR(25),                /*   CUSTOMER NAME       */
    3 FILLER    CHAR(34),                /*   ** RESERVED **      */
  INVOICE_FORMAT  (5) CHAR(8)            /* FORMAT ARRAY FOR      */
                INIT((5) (1)'CHAR    '), /*  INVOICE_RECORD VDEF  */
  INVOICE_LENGTH   (5) FIXED BIN(31,0)   /* LENGTH ARRAY FOR      */
                INIT(6,8,4,3,25);        /*  INVOICE_RECORD VDEF  */
DECLARE                                  /*                       */
  1 SCROLL_VARS,                         /* TBDISPL SCROLL FIELDS */
    3 ZSCROLLA  CHAR(4),                 /*   SCROLL AMOUNT       */
    3 ZTDRET    CHAR(8),                 /*   RETURN ON EOD       */
    3 ZTDSCRP   FIXED BIN(31,0),         /*   TOP ROW CRP         */
    3 ZTDAMT    FIXED BIN(31,0),         /*   #ROWS TO ADD        */
    3 ZTDSIZE   FIXED BIN(31,0),         /*   SCROLLABLE AREA SIZE*/
    3 ZTDLROWS  FIXED BIN(31,0),         /*   #ROWS IN LOGICAL TBL*/
    3 ZTDADD    CHAR(3),                 /*   NEED TO ADD ROWS?   */
  SCROLL_FORMAT (7) CHAR(8)              /* FORMAT ARRAY FOR      */
                INIT((2) (1)'CHAR    ',  /*  SCROLL_VARS VDEFINE  */
                     (4) (1)'FIXED   ',  /*                       */
                     'CHAR    '),        /*                       */
  SCROLL_LENGTH (7) FIXED BIN(31,0)      /* LENGTH ARRAY FOR      */
                INIT(4,8,4,4,4,4,3);     /*  SCROLL_VARS VDEFINE  */
                                         /*                       */
```

Figure 20. PL/I Dialog Function Example Program (Part 2 of 7)

```
DECLARE                                  /*                       */
  I             FIXED BIN(31,0),         /* WORK INDEX            */
  L4            FIXED BIN(31,0),         /* VDEFINE LENGTH PARM   */
  TBDISPL_RC    FIXED BIN(31,0),         /* TBDISPL RETURN CODE   */
  BOTTOM        FIXED BIN(31,0),         /* CRP OF BOTTOM ROW     */
  NEW_BOTTOM    FIXED BIN(31,0),         /* CRP OF NEW BOTTOM ROW */
  REQUESTED_TOP FIXED BIN(31,0),         /* TOP ROW REQUESTED BY  */
                                         /*  END USER SCROLL      */
  ADD_NUMBER    FIXED BIN(31,0);         /* #ROWS TO ADD          */
                                         /*                       */
DECLARE                                  /*                       */
  MIN     BUILTIN,                       /* PL/I BUILTIN          */
  PLIRETV BUILTIN,                       /*  FUNCTIONS            */
  ISPLINK EXTERNAL ENTRY                 /* ISPF SERVICE          */
          OPTIONS(ASM INTER RETCODE);    /*  INTERFACE            */
                                         /*                       */
DECLARE                                  /*                       */
  INVFILE  FILE INPUT RECORD SEQUENTIAL  /* INVOICE FILE          */
     ENV(FB BLKSIZE(6160) RECSIZE(80));  /*                       */
                                         /*                       */
/*******************************************************************/
/*                                                                 */
/*  ISSUE VDEFINE SERVICE CALLS TO DEFINE THE TABLE VARIABLES,     */
/*  SCROLL SYSTEM VARIABLES, AND OTHER MISCELLANEOUS FIELDS TO     */
/*  ISPF.                                                          */
/*                                                                 */
/*******************************************************************/
                                         /*                       */
CALL ISPLINK('VDEFINE ',                 /* DEFINE TABLE VARS     */
             '(INV DATE PART QTY CUST)', /*                       */
             INVOICE_RECORD,             /*                       */
             INVOICE_FORMAT,             /*                       */
             INVOICE_LENGTH,             /*                       */
             'LIST    ');                /*                       */
                                         /*                       */
CALL ISPLINK('VDEFINE ',                 /* DEFINE SCROLL VARS    */
          '(ZSCROLLA ZTDRET ZTDSCRP ZTDAMT ZTDSIZE ZTDLROWS ZTDADD)',
             SCROLL_VARS,                /*                       */
             SCROLL_FORMAT,              /*                       */
             SCROLL_LENGTH,              /*                       */
             'LIST    ');                /*                       */
L4 = 4;                                  /*                       */
CALL ISPLINK('VDEFINE ',                 /* DEFINE BOTTOM ROW CRP */
             '(BOTTOM)',                 /*                       */
             BOTTOM,                     /*                       */
             'FIXED   ',                 /*                       */
             L4);                        /*                       */
                                         /*                       */
```

Figure 20. PL/I Dialog Function Example Program (Part 3 of 7)

```
CALL ISPLINK('VDEFINE ',                /* DEFINE PANEL VAR YEAR */
             '(YEAR)',                  /*                       */
             YEAR,                      /*                       */
             'CHAR    ',                /*                       */
             L4);                       /*                       */
                                        /*                       */
/*******************************************************************/
/*                                                                 */
/*  ISSUE TBCREATE SERVICE CALL TO CREATE TEMPORARY TABLE          */
/*  INVTABLE.  MAKE EACH OF THE TABLE VARIABLES NAME VARIABLES.    */
/*                                                                 */
/*******************************************************************/
                                        /*                       */
CALL ISPLINK('TBCREATE',                /*                       */
             'INVTABLE',                /*                       */
             ' ',                       /*                       */
             '(INV DATE PART QTY CUST)'); /*                     */
                                        /*                       */
/*******************************************************************/
/*                                                                 */
/*  OPEN FILE INVFILE AND READ THE HEADER RECORD.                  */
/*                                                                 */
/*******************************************************************/
                                        /*                       */
OPEN FILE(INVFILE);                     /* OPEN INVOICE FILE     */
READ FILE(INVFILE)                      /* READ HEADER RECORD    */
  INTO(HEADER_RECORD);                  /*                       */
                                        /*                       */
/*******************************************************************/
/*                                                                 */
/*  READ THE FIRST 60 RECORDS FROM INVILE, ADDING EACH TO THE      */
/*  TABLE.                                                         */
/*                                                                 */
/*******************************************************************/
                                        /*                       */
ADD_NUMBER = 60;                        /*                       */
DO I = 1 TO ADD_NUMBER;                 /*                       */
  READ FILE(INVFILE)                    /* READ NEXT RECORD      */
    INTO(INVOICE_RECORD);               /*                       */
  CALL ISPLINK('TBADD   ',              /* ADD INVOICE TO TABLE  */
               'INVTABLE',              /*                       */
               ' ',                     /*                       */
               ' ',                     /*                       */
               ADD_NUMBER);             /*                       */
END;                                    /*                       */
```

Figure 20. PL/I Dialog Function Example Program (Part 4 of 7)

```
/*****************************************************************/
/*                                                               */
/*  SKIP BACK TO THE TABLE TOP, INITIALIZE THE ZTDRET AND        */
/*  ZTDLROWS SYSTEM VARIABLES, AND ISSUE A TBDISPL SERVICE CALL  */
/*  TO DISPLAY THE TABLE.                                        */
/*                                                               */
/*****************************************************************/
                                        /*                       */
CALL ISPLINK('TBTOP   ',                /* SKIP TO TABLE TOP     */
             'INVTABLE');               /*                       */
ZTDRET = 'DOWN    ';                    /* RETURN ON BOTTOM OF   */
                                        /*  DATA                 */
ZTDLROWS = NUM_RECS;                    /* SET LOGICAL #ROWS     */
CALL ISPLINK('TBDISPL ',                /* PUT UP TABLE          */
             'INVTABLE',                /*                       */
             'INVPANEL');               /*                       */
TBDISPL_RC = PLIRETV();                 /*                       */
                                        /*                       */
/*****************************************************************/
/*                                                               */
/*  LOOP WHILE USER HAS NOT ISSUED THE END COMMAND, CHECK TO     */
/*  SEE IF ADDITIONAL ROWS ARE NEEDED TO SATISFY SCROLL, ADD ROWS */
/*  IF APPROPRIATE, AND THEN REDISPLAY TABLE.                    */
/*                                                               */
/*****************************************************************/
                                        /*                       */
DO WHILE(TBDISPL_RC < 8);               /* LOOP WHILE NOT END    */
  IF ZTDADD = 'YES' THEN                /* NEED TO ADD ROWS?     */
    DO;                                 /*                       */
                                        /*                       */
      CALL ISPLINK('VGET    ',          /*  CHECK TO SEE IF MAX  */
                   '(ZSCROLLA)',        /*   SCROLL              */
                   'SHARED  ');         /*                       */
      IF ZSCROLLA = 'MAX' THEN          /*  IF SO, ADD ALL       */
        ZTDAMT = 999999;                /*    REMAINING INVOICES */
      ELSE;                             /*  ELSE, ADD ZTDAMT ROWS*/
                                        /*                       */
```

Figure 20. PL/I Dialog Function Example Program (Part 5 of 7)

```
      CALL ISPLINK('TBBOTTOM',            /*  SKIP TO TABLE BOTTOM */
                   'INVTABLE',            /*  TO ADD ROWS          */
                   ' ',                   /*                       */
                   ' ',                   /*                       */
                   ' ',                   /*                       */
                   'BOTTOM  ');           /*  SAVE CRP OF BOTTOM   */
                                          /*                       */
      ADD_NUMBER = MIN(ZTDAMT,            /*  ADD ZTDAMT ROWS OR   */
                       ZTDLROWS-BOTTOM);  /*  UNTIL INVFILE EOF    */
      DO I = 1 TO ADD_NUMBER;             /*                       */
                                          /*                       */
        READ FILE(INVFILE)                /*  READ RECORD          */
          INTO(INVOICE_RECORD);           /*                       */
                                          /*                       */
        CALL ISPLINK('TBADD   ',          /*  ADD IT TO TABLE      */
                     'INVTABLE',          /*                       */
                     ' ',                 /*                       */
                     ' ',                 /*                       */
                     ADD_NUMBER);         /*                       */
      END;                                /*                       */
      IF ZSCROLLA ¬= 'MAX' THEN           /*  IF NOT MAX SCROLL,   */
        IF ZTDSCRP = 0 THEN               /*   MAY NEED TO SET     */
          DO;                             /*   ZTDSCRP             */
                                          /*                       */
            NEW_BOTTOM = BOTTOM +         /*  CALCULATE NEW BOTTOM */
              ADD_NUMBER;                 /*                       */
            REQUESTED_TOP = BOTTOM +      /*  CALCULATE TOP ROW    */
              ZTDAMT - ZTDSIZE + 1;       /*   REQUESTED BY SCROLL */
                                          /*                       */
            IF NEW_BOTTOM <               /*  IF HIT EOF BEFORE    */
               REQUESTED_TOP THEN         /*    REACHING TOP ROW   */
                                          /*    REQUESTED, DISPLAY */
              ZTDSCRP = NEW_BOTTOM + 1;   /*    ONLY BOTTOM OF     */
                                          /*    DATA MARKER        */
            ELSE                          /*  ELSE                 */
              ZTDSCRP = REQUESTED_TOP;    /*    ADDED REQUESTED    */
                                          /*    TOP, SET ZTDSCRP   */
                                          /*    TO NEW TOP ROW     */
          END;                            /*                       */
        ELSE;                             /* NO NEED TO SET        */
      ELSE;                               /*  ZTDSCRP              */
                                          /*                       */
    END;                                  /*                       */
  ELSE;                                   /* DON'T NEED TO ADD ROWS*/
                                          /*                       */
  CALL ISPLINK('TBDISPL ',                /* REDISPLAY TABLE       */
               'INVTABLE');               /*                       */
  TBDISPL_RC = PLIRETV();                 /*                       */
END;                                      /*                       */
                                          /*                       */
```

Figure 20. PL/I Dialog Function Example Program (Part 6 of 7)

```
/*****************************************************************/
/*                                                               */
/*  PERFORM FINAL CLEANUP.                                       */
/*                                                               */
/*****************************************************************/
                                        /*                       */
CALL ISPLINK('TBEND   ',                /* CLOSE AND DELETE      */
             'INVTABLE');               /*  TABLE                */
CLOSE FILE(INVFILE);                    /* CLOSE INVOICE FILE    */
CALL ISPLINK('VDELETE ',                /* DELETE FUNCTION POOL  */
             '*       ');               /*  VARIABLES            */
                                        /*                       */
RETURN (0);                             /*                       */
END INVOICE;                            /*                       */
```

Figure 20. PL/I Dialog Function Example Program (Part 7 of 7)

Now, assume that a user is running the invoice dialog on a terminal with 24 lines. The initial display of the table is shown in Figure 21.

```
------------------------------ 1986 TRANSACTIONS ------ ROW 1 OF 10000
Command ====>                                          Scroll ===> PAGE


Invoice    Transaction    Part      Quantity   Customer

 Number        Date        Number
-------    -----------    ------    --------   --------
0000001     01/06/86      7071        100       Acme Parts
0000002     01/06/86      0015         15       Parts City
0000003     01/07/86      1023        340       Cary Auto Center
0000004     01/08/86      0231          1       Parts Unlimited
0000005     01/08/86      3423        805       Bosworth's Parts
0000006     01/08/86      2341        165       Acme Parts
0000007     01/08/86      7653         20       Acme Parts
0000008     01/08/86      3353        100       Bosworth's Parts
0000009     01/08/86      0003        325       Bosworth's Parts
0000010     01/08/86      3322          1       Bosworth's Parts
0000011     01/10/86      2344         23       Parts Unlimited
0000012     01/10/86      4333         55       Cary Auto Center
0000013     01/10/86      3079         65       Parts Company of NC
0000014     01/10/86      4763        340       Cary Auto Center
0000015     01/10/86      0956         70       Cary Auto Center
0000016     01/10/86      4536         52       ABC Parts
0000017     01/10/86      0973        330       ABC Parts
```

Figure 21. Initial Display for Dynamic Table Expansion Example

Notice that even though the table actually contains only 60 rows, the top row displayed indicator shows “ROW 1 OF 10000”. This was accomplished by setting the ZTDLROWS variable in the function pool to a value of 10 000. TBDISPL will pick up this value and use it when ZTDRET has been properly set.

Assume that the user enters the command “DOWN 50” on the command line. This should result in rows 51–67 being displayed. Remember though that only rows 1–60 are currently in the table. Because there are not enough rows in the table to fill the screen, control will return to function INVOICE. Upon return from TBDISPL, the system variables used by the dialog have the following values:

ZSCROLLA	0050
ZTDADD	YES
ZTDSCRP	51

ZTDAMT	7
ZTDSIZE	17

ZTDAMT contains the number of rows that must be added to satisfy the scroll request and fill a full screen. ZTDSCRP has the CRP of the row that will be at the top of the screen after the scroll. Because it is non-zero, function INVOICE does not need to set it. In fact, all that the function has to do is skip to the table bottom, read and add the next 7 invoices to the table, and then issue a TBDISPL service request to redisplay the table. When the table is displayed again, it appears as shown in Figure 22.

```
------------------------------ 1986 TRANSACTIONS ------ ROW 51 OF 10000
Command ====>                                          Scroll ===> PAGE


Invoice    Transaction    Part      Quantity   Customer
Number        Date        Number
-------    -----------    ------    --------   --------
0000051     01/15/86      7536          6      Parts Unlimited
0000052     01/15/86      0546         54      ABC Parts
0000053     01/15/86      3349         65      Parts Company of NC
0000054     01/15/86      4234        340      Cary Auto Center
0000055     01/15/86      0342         70      Cary Auto Center
0000056     01/18/86      4544         52      ABC Parts
0000057     01/19/86      0763        330      Cary Auto Parts
0000058     01/19/86      0841        540      Bosworth's Parts
0000059     01/19/86      0445        560      ABC Parts
0000060     01/19/86      4542        450      ACME Parts
0000061     01/25/86      7071        100      Acme Parts
0000062     01/25/86      0015         15      Parts City
0000063     02/27/86      1023        340      Cary Auto Center
0000064     02/04/86      0231          1      Parts Unlimited
0000065     02/04/86      3423        805      Bosworth's Parts
0000066     02/04/86      2341        165      Acme Parts
0000067     02/04/86      7653         20      Acme Parts
```

Figure 22. Second Display for Dynamic Table Expansion Example

Now assume that the user issues:

```
DOWN 5000
```

This should result in rows 5051–5067 being displayed. As before, there are not enough rows in the table to handle the scroll request, so control returns to function INVOICE with the following information in the system variables:

ZSCROLLA	5000
ZTDADD	YES
ZTDSCRP	0
ZTDAMT	5000
ZTDSIZE	17

Notice that this time ZTDSCRP has a value of 0. This indicates that the new top row, as requested by the user scroll, is not in the physical table. After adding the 5000 rows indicated by the ZTDAMT system variable, function INVOICE must set ZTDSCRP to the CRP of the row that should be displayed at the top after the scroll (row 5051). This is accomplished in the dialog by adding ZTDAMT to the number of rows in the current table, and then subtracting out the size of the scrollable area (ZTDSIZE). When the table is redisplayed, it appears as shown in Figure 23 on page 61.

```
------------------------------ 1986 TRANSACTIONS ------ ROW 5051 OF 10000
Command ====>                                        Scroll ===> PAGE


Invoice    Transaction    Part      Quantity   Customer
Number        Date        Number
-------    -----------    ------    --------   --------
0005051     07/12/86      7326        436      Parts Unlimited
0005052     07/12/86      0516         54      ABC Parts
0005053     07/21/86      3549          5      Parts Company of NC
0005054     07/24/86      4243        350      Cary Auto Center
0005055     07/25/86      0342        540      Cary Auto Center
0005056     07/31/86      4544        444      ABC Parts
0005057     07/11/86      0653         30      Cary Auto Parts
0005058     08/29/86      0821        450      Bosworth's Parts
0005059     08/01/86      6445        460      ABC Parts
0005060     08/01/86      4942        850      ACME Parts
0005061     08/01/86      7021        180      Acme Parts
0005062     08/01/86      6026        945      Parts City
0005063     08/07/86      1523         30      Cary Auto Center
0005064     08/07/86      0531        451      Parts Unlimited
0000065     08/07/86      3263        455      Bosworth's Parts
0005066     08/07/86      2771          5      Acme Parts
0005067     08/07/86      7453        576      Acme Parts
```

Figure 23. Third Display for Dynamic Table Expansion Example

Finally, assume that the user again issues:

```
DOWN 5000
```

A scroll of 5000 would display rows 10051–10067, if there were that many invoices in the file. However, because there are only 10 000 invoices, function INVOICE can add only rows 5068–10000 to the table and then redisplay the table. On return from TBDISPL, the system variables again contain the following information.

ZSCROLLA	5000
ZTDADD	YES
ZTDSCRP	0
ZTDAMT	5000
ZTDSIZE	17

After adding all of the invoices to the table (end of file is reached), the dialog must set system variable ZTDSCRP. Because the scroll amount has caused the user to scroll past the end of data, the dialog sets ZTDSCRP to a value that will cause only the bottom of data marker to be displayed. That is, ZTDSCRP is set to a value greater than the number of rows in the table. When the table is redisplayed it appears as shown in Figure 24 on page 62.

```
------------------------------ 1986 TRANSACTIONS ----------------------
Command ====>                                        Scroll ===> PAGE


Invoice    Transaction    Part      Quantity   Customer
Number        Date        Number
-------    -----------    ------    --------   --------
***************************** BOTTOM OF DATA ****************************
```

Figure 24. Fourth Display for Dynamic Table Expansion Example

One case not illustrated above is that of the user issuing a DOWN MAX scroll request. In this case ZTDAMT and ZTDSCRP would each have a value of 0 when control returns to the dialog. ZSCROLLA would have a value of MAX. The dialog would add all remaining invoices to the table and then redisplay the table. It is not necessary in a MAX scroll case to set ZTDSCRP before redisplaying the table because ISPF automatically positions the table so that a full screen plus the bottom of data marker are displayed.

In this example the program has been written so that control continues to return to the dialog after all of the invoice file records have been added to the table. To further improve performance, it may be desirable for the dialog to disable the return after the end of file has been reached. This can be done by setting the ZTDRET function pool variable to some value other than DOWN, UP, or VERTICAL, and then issuing a TBDISPL service request with the panel name specified. Be aware that when a panel name is specified, ISPF clears any pending scroll requests. So it is up to the dialog to position the table CRP to the appropriate row to simulate the scroll. For example, assume that a DOWN MAX scroll request has been issued and the dialog has added all remaining invoices to the table. The dialog then sets ZTDRET to blank and prepares to issue the TBDISPL service request, with a panel name, to display the table. To simulate the user scroll the dialog issues a TBSKIP service request to position the CRP to the row that will cause a full screen plus the bottom of data marker to be displayed. When the TBDISPL request is subsequently issued, ISPF will position the table based on the CRP, thereby simulating the scroll.

Using the Variable Services

Dialog variables are the main communication vehicle between the components of a dialog and ISPF services. Program modules, command procedures, panels, messages, tables, and skeletons can all refer to the same data through the use of dialog variables. Variable services allow you to define and use dialog variables.

Some variable services require that ISPF search through the variable pools to locate requested variables. ISPF searches the pools in the following order:
1. Function pool (defined variables)
2. Function pool (implicit variables)
3. Shared pool
4. Application profile pool (profile pool).

Searching Variable Pools

Dialog variables are organized into groups, or pools, according to the dialog and application with which they are associated. An application is one or more dialogs, each of which has been started using the same application ID.

A pool can be thought of as a list of variable names that enables ISPF to access the associated values. When a DM 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 listed below:

Function pool
: Contains variables accessible only by that function. A variable that resides in the function pool of the function currently in control is called a function variable.

Shared pool
: Contains variables accessible only by dialogs belonging to the same application. A variable that resides in the shared pool of the current application is called a shared variable.

Profile pool
: Contains variables that are automatically retained for the user from one session to another. A variable that resides in the profile pool is called an application profile variable or profile variable. Profile variables are automatically available when an application begins and are automatically saved when it ends.

The number of shared, function, and profile variables that can exist at any one time depends on the amount of storage available.

SELECT Service and Variable Access

Figure 25 on page 64 shows how the SELECT service can be used to pass control within a dialog and illustrates the resulting pool structures. Menus A and B access variables from the shared and profile pools, because menus are not part of any function. The dialog invokes Function X, which uses the VPUT service to copy one of the variables from its function pool into the shared pool. Next, the dialog invokes Function Y, which uses the VGET service to copy a dialog variable from the shared pool to its function pool. Then it uses the SELECT service for further menu processing.

Figure 25 on page 64 also shows how the SELECT service controls access to dialog variable pools from both functions and menus.

When you define a variable as an input variable on a selection panel, the following actions occur during processing of the menu:

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

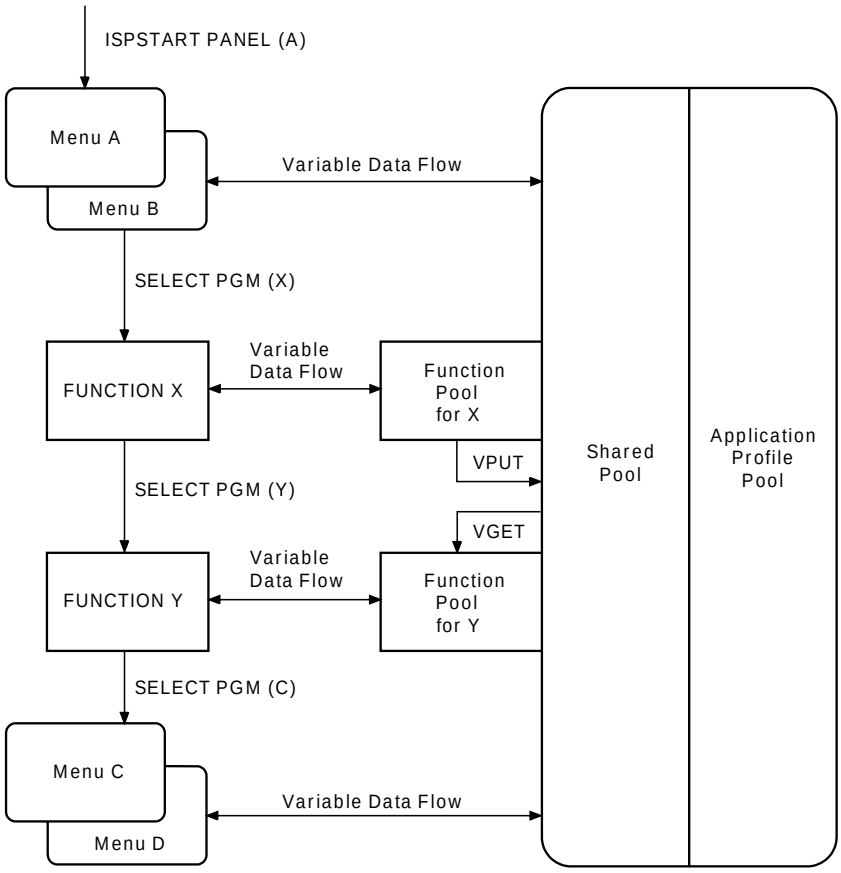


Figure 25. Control and Data Flow in a Dialog

Function Pools and Dialog Functions

Each function has its own unique pool of dialog variables. This is illustrated in Figure 25. These function pools are maintained by ISPF on behalf of each respective function. A function uses these dialog variables to communicate with the various DM services. A function pool's variables can be accessed only by the function for which the pool was created. To make these variables available to other functions, you must use variable services to copy any variables to be shared into the shared pool.

Dialog variables associated with one function can have the same names as dialog variables associated with another function, but they reside in different function pools, and therefore, are not the same variables.

When a new function begins, ISPF creates a function pool for it. Variables can 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.

Command Procedures, Program Functions, and Function Pools

When the function in control is a command procedure, the list of variable names kept by the command language processor and the list of function variables kept by ISPF is the same list. Thus, a variable created by the command procedure during its execution is automatically a dialog variable. Likewise, the command procedure can automatically access a dialog variable entered in the function pool by ISPF. However, ISPF variable names cannot exceed 8 characters.

Any CLIST or REXX variable such as SYSDATE and SYSTIME, which are dynamically evaluated when referred to, can be used in a CLIST or REXX exec running under ISPF; however, it cannot be used in panels, messages, skeletons, or tables. For SYSDATE and SYSTIME, use ISPF system variables ZDATE and ZTIME, respectively, which contain similar information.

ISPF makes available two other system variables, ZDATEF and ZDATEFD, to support date representation in various national languages. ZDATEF contains the date represented by the characters YY, MM, and DD plus delimiters. These characters are never translated; however, they can be in any order. For example, the date could be expressed MM/DD/YY, YY/MM/DD, and so on, depending on how a date is expressed in a given national language. ZDATEFD contains the same date format, translated into the session national language.

TSO global variables, in effect 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 CLIST command procedure example illustrates that ISPF treats command procedure variables as dialog variables.

Assume that the definition for panel XYZ contains two dialog variable input fields, AAA and BBB. 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 procedure:

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

is executed, variable AAA is set to the value 1. The procedure then invokes the DISPLAY service to display panel XYZ. The value of AAA is 1 on the displayed panel. ISPF creates the variable BBB in the function pool and displays it as a blank.

Now, in response to the panel display, you type 100 in the first field (AAA) and 20 in the second field (BBB). When you press Enter, 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. When the next statement executes, it creates variable CCC and sets it to 120, the sum of AAA and BBB.

When the function in control is a program, the associated function pool is not shared with ISPF. This is because a program is compiled, not interpreted as command procedures are. ISPF maintains a list of variables that belong to the function so that DM services can use dialog variables for communication of data.

ISPF makes two types of entries in the program function pool, defined and implicit.

Use a Variable Service to Create or Delete Defined Variables

Use the VDEFINE service to create a defined dialog variable name in the function pool and associate it with the corresponding program variable. This association enables ISPF to directly access and modify that program variable. Otherwise, the program's variables are not available to ISPF. Use the VDELETE service to end this association and remove ISPF's ability to access that program variable.

The following program coded in PL/I specifies that field PA of the program can be accessed by ISPF by using a dialog variable named FA. Then, the DISPLAY service is called to display panel XYZ.

```
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 through dialog variable FA. If dialog variable FA is specified as an input field on panel XYZ, then "OLD DATA" displays in field FA, and ISPF stores any data entered in that field into the program variable PA.

Creating Implicit Variables

ISPF places implicit variables in the function pool when an ISPF service:

- Refers to a dialog variable name that is not found in the standard search reference
- Must store data in a dialog variable that does not already exist in the function pool.

Let's illustrate how ISPF creates an implicit variable. Assume that panel XYZ, in the preceding example, allows the user to enter a second value and that this value is to be stored in dialog variable IA. This is the first reference to IA; therefore, it does not yet exist in the function pool. Because variable IA does not exist when it is referred to, ISPF creates it in the function pool. ISPF then stores into IA the value entered on the panel. Thus, IA is an implicit dialog variable.

Any DM service invoked by a program function can access an implicit variable directly by referencing the variable name. However, implicit variables cannot be accessed directly from a program function. Programs access implicit variables only through the use of the VCOPY and VREPLACE services.

When you are using APL2, variables in the current APL2 workspace that follow APL2 and ISPF naming rules become function pool variables. ISPF treats these as implicit variables. The VDEFINE service is not used with APL2 dialogs.

Naming Defined and Implicit Variables

A defined variable and an implicit variable can have the same name. This occurs when, using the VDEFINE service, a defined variable is created that uses 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, only the defined entry can be accessed. You can make the implicit entry accessible by using the VDELETE service to remove any defined entries for that variable name made through the VDEFINE service. The implicit entries are not affected.

You can define a given dialog variable name many times within a given function. Each definition can associate a different program variable with the dialog variable name. This is referred to as *stacking.* Only the most recent definition of that dialog variable is accessible. A previous definition of that variable can be made accessible by using the VDELETE service to delete the more recent definitions of that name.

For example, the main routine 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 &DMacro services invoked after the second VDEFINE would have access to only the subroutine's program variable. The subroutine would use the VDELETE service to delete that dialog variable before returning, thereby uncovering the earlier definition set up in the main routine. To avoid a possible program error, each VDEFINE processed within a function for a given dialog variable name should have a VDELETE using the same name or an asterisk (*) as the operand. When an asterisk is used as the operand, the VDELETE service removes all dialog variable names previously defined by the program module from the function pool.

The VRESET service allows a program to remove its function pool variables as though VDELETEs had been done. Any implicit variables are also deleted.

Sharing Variables among Dialogs

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

The SELECT service creates shared pools when it processes the ISPSTART or ISPF command, and when you specify the NEWAPPL or NEWPOOL keywords with the SELECT service. When SELECT returns, it deletes the shared pool and reinstates any previous shared pool.

A function can copy dialog variables from its function pool to the shared pool by using the VPUT service. In addition, another function can directly copy these variables to its function pool by means of the VGET service. Because 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 pool.

Saving Variables across ISPF Sessions

Like the shared pool, the application profile pool contains variables that are accessible to dialogs within an application. But, unlike the shared pool, the profile variables are saved across sessions.

When a new application is started, it has access to a profile pool. If an application is restarted by split screen, for example, both calls of the application access exactly the same profile pool. The profile pool 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 pool is used.

When accessing an application profile pool that is not currently active, ISPF first searches the user's profile files for a profile named xxxxPROF. ISPF finds the profile if the user previously ran the application, and thus, had a copy of the profile pool.

If ISPF cannot find the profile, it searches the table input file. The application developer can provide a profile pool with the table files. A profile pool contains variable names and values initialized for the application.

If ISPF cannot find the member in either the user's profile pool or table input library, it initializes the application profile pool with the contents of the default profile pool, ISPPROF, which is read from the table input library. If the dialog manager application ID "ISP" is active, the currently active copy of ISPPROF is used as the default, rather than reading ISPPROF from ISPTLIB. ISPPROF is distributed with ISPF. It contains a set of default Function key values. An installation can modify this table to change these settings or to include other variables that will be copied to initialize new profile pools.

Upon completion of the application, ISPF saves the contents of the application profile pool, under the name xxxxPROF, in the user's profile library. ISPF deletes the profile pool from storage when the last call of the application terminates.

You must use the VPUT service to enter variables in the profile pool. Functions can copy variables from the profile pool into function pools by using the VGET variable services. Selection panels automatically update existing profile variables.

Removing Variables from the Shared or Profile Pool

As mentioned in an earlier topic, you can use the VDELETE or VRESET service to remove variables only from the function pool. However, if you wish to do some housekeeping in the other variable pools, you can use the VERASE service. The VERASE service allows you to remove variable names and values from the shared pool, the profile pool, or both. You can specify on the VERASE service request a list of one or more variable names to be removed from the shared pools or both. For example:

```
ISPEXEC VERASE (AGE ADDRESS SOCSEC) PROFILE
```

might be used to remove variable values for age, address, and social security number from the profile pool.

For detailed information about VERASE and other services, refer to the *ISPF Services Guide*

Read-Only Profile Pool Extension Variables

ISPF provides for a read-only extension of the application profile variable pool. This allows installations to maintain better control over application default profile variables. It also results in conservation of disk storage because a copy of these variables need not exist in the application profile of every application user.

To use the read-only extension, you do two things:

1. First you must define the read-only extension. The read-only extension is actually a table, which you can create by using the ISPF TBCREATE table service. You add variables to this table as extension variables; that is, variables not specified when the table is created. This is illustrated in the CLIST procedure below, using the SAVE keyword on the TBADD table service.

   You need to create the extension table only once. After the table is saved, you must define it to ISPF by using an ALLOCATE command or a LIBDEF service request.
2. You then use DM variable services to put the name of the read-only extension table into system variable ZPROFAPP in the profile variable pool.

An example of a CLIST to create a read-only extension table named ROTABLE is shown in Figure 26. The table is to contain variables RDONLY1, RDONLY2, and RDONLY3 set to values of LKHFC, FLIST, and SPOOLFUL, respectively. After the procedure closes the table, it sets system variable ZPROFAPP to the table name, ROTABLE. The procedure then puts ZPROFAPP into the profile variable pool.

```
/* Example of creating a read-only extension table */
SET ROV1 = LKHFC
SET ROV2 = FLIST
SET ROV3 = SPOOLFUL
SET ROVLIST = &STR(ROV1 ROV2 ROV3)
ISPEXEC TBCREATE ROTABLE
ISPEXEC TBADD ROTABLE SAVE(&ROVLIST)
ISPEXEC TBCLOSE ROTABLE
SET &RC = &LASTCC
IF &RC = 0 THEN -
DO
  /* Put extension table name into system variable ZPROFAPP. */
  SET ZPROFAPP = ROTABLE
  ISPEXEC VPUT ZPROFAPP PROFILE
END
```

Figure 26. CLIST to Create a Read-Only Extension Table

When a new application that uses the NEWAPPL keyword on the SELECT service is specified, ISPF interrogates variable ZPROFAPP in the new application's profile pool. If the variable value is not null, it is assumed to be the name of the profile extension table. ISPF searches the table input files for a table with the name specified by ZPROFAPP. The set of variables in this table becomes the read-only extension for the profile pool of the application just selected.

Although variable services are not effective for updating the read-only extension, you can create or update the table used as the extension by using DM table services. Updating the extension may be done only when the application is not active, because the table is open in nowrite mode while the application is active.

If a variable name is referred to and exists in both the profile pool and the read-only extension table, ISPF uses the variable from the user's profile pool. In other words, the search order is: first the profile pool, and then the read-only extension.

If a VPUT PROFILE is issued for a variable in the read-only extension, the update for that variable is made to the user area of the profile pool, not to the read-only extension. Only the profile pool variable update is saved and accessed, not the extension variable value.

Variables Owned by ISPF

A second level of profile pool, the system profile pool (ISPSPROF), is always active. The dialog manager owns the dialog variables within the system profile pool, and the variables cannot be modified by an application. They can be read, however, because the system profile pool is included in the standard search sequence after the profile pool. All system variable names begin with “Z”, such as “ZTERM”, and supply information such as terminal type and list and log defaults.

If a system profile pool variable is used on a selection panel, a corresponding field is created in the profile pool (ISPPROF). Subsequently, when that variable is referred to by the dialog, the profile pool value is used rather than the system profile pool value. The dialog can use the VERASE service to delete variables from the profile (ISPPROF) pool.

Variable Formats

Information entered on a panel is in character string format. All dialog variables remain in character string format when stored:
- As implicit variables in a function pool
- In the shared pool
- In the profile pool
- In ISPF tables.

Defined variables, however, can be translated to a fixed binary, bit, hexadecimal, float, packed, or binary string, or to a user-defined format when stored internally in a program module. 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 accessed.

The VMASK service is used to validate input into a VDEFINEd dialog variable. Refer to the *ISPF Services Guide* for more information.

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

Truncation
If the current length of the variable is greater than the defined length within the module, the remaining data is lost.

Translation
If the variable is defined as something other than a character string, and the external representation has invalid characters, the contents of the defined variable are lost.

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

System Variables Communicate between Dialogs and ISPF

System variables are used to communicate special information between the dialog and the dialog manager (ISPF). System variable names are reserved for use by the system. They begin with the letter “Z”. Therefore, avoid names that begin with “Z” when choosing dialog variable names.

The types of system variables are input, output, non-modifiable, and input-output. Their type depends on their usage.

To access and update system variables, use variable services according to which pool the variables are in. System variables in the function pool can be accessed and updated directly from a command procedure. Those in the shared or profile pools can be accessed by using the VGET service, and updated by using the VPUT service.

A program function can access and update system variables in the function pool using the VDEFINE service. Dialog variables can be accessed by using the VCOPY service and updated by using the VREPLACE service.

The system variables in the shared or profile pools can be accessed by using the VCOPY service. They can be updated by first updating the variable in the function pool by using the VDEFINE or VREPLACE service and then moving that value to the shared or profile pool by using the VPUT service.

Using VDEFINE, VDELETE, VRESET, VCOPY, VMASK, and VREPLACE

For functions coded in a programming language other than APL2, you can manage the availability to ISPF of the internal program variables that are to be used as dialog variables through the ISPF VDEFINE, VDELETE, and VRESET services.

Variables used in a program function are not automatically put into that function's variable pool. Therefore, those variables are not initially available to ISPF for processing function requests. A function can use the VDEFINE service to make function variable names available to ISPF through the function pool.

The VDELETE and VRESET services are used to cancel the effect of using VDEFINE service requests. VDELETE can be used to delete access by ISPF to selected defined variables by removing them from the function pool. VRESET removes all defined and implicit variables from the function pool.

A program function can obtain a copy of dialog variables by using the VCOPY service. The service request can specify that either the variable data address or the data itself be returned.

The VMASK service is used to validate the data of a variable defined with the VDEFINE service. VMASK associates a specified user or predefined mask with a variable previously defined with VDEFINE. The VEDIT statement must be used to indicate VMASKed variables on a panel.

A program function can update the contents of dialog-defined or implicit variables in the function pool by using the VREPLACE service. The names of the variables to be updated and the new contents are specified with the VREPLACE service request.

The VDEFINE, VDELETE, VRESET, VCOPY, VMASK, and VREPLACE variable services are not used with functions coded as procedures. For a function coded as a CLIST or APL2 procedure, variables used in the procedure are automatically treated as dialog variables. No special action is required to define them to ISPF. Any trailing blanks in CLIST variables are not truncated; they remain as part of the variables.

Using the VGET, VPUT, and VERASE Services

The VGET, VPUT, and VERASE services can be used by both program and procedure functions. Functions use the VGET and VPUT services to control movement of variables between function pools and shared or profile pools.

Each function has its own function variable pool. The variables in a given function's pool are not available to other functions, and vice versa. To overcome this, a function can use the VGET service to copy into its function pool variables from the shared or profile pools. The function can make variables in its function pool available to other functions in the same application by copying them to the shared or profile pool by using the VPUT service.

You can use the VERASE service to remove variable names and values from the shared and/or profile pool. The VDELETE and VRESET services are available for removing function pool variables.

Summary of Variable Services

The variable services are:

All Functions

VERASE	Remove variables from shared pool and/or profile pool
VGET	Retrieve variables from shared pool or profile pool
VPUT	Update variables in shared pool or profile pool

Program Functions Only

VCOPY	Copy data from a dialog variable to the program
VDEFINE	Define function program variables to ISPF
VDELETE	Remove definition of function variables
VMASK	Associate a mask with a dialog variable
VREPLACE	Update dialog variable with program data specified in the service request
VRESET	Reset function variables

Using the Table Services

Table services let you use and maintain sets of dialog variables. A table is a two-dimensional array of information in which each column corresponds to a dialog variable, and each row contains a set of values for those variables.

Contents for a table are shown in Table 3 on page 78. In that example, the variables that define the columns are as follows:

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

Where Tables Reside

A table can be either temporary or permanent. A temporary table exists only in virtual storage. It cannot be written to disk storage.

Permanent tables are maintained in one or more table libraries. A permanent table, while created in virtual storage, can be saved on direct access storage. It can 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 is complete. In the MVS/XA* environment, some table services data areas reside above the 16-megabyte line when that storage is available, and TSO/E Version 2 Release 1 (or later) is installed.

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

When a permanent table is opened for processing, it is read from a table input library. A table to be saved can be written to a table output library that is 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.

Accessing Data

You specify the variable names that define table columns when the table is created. Specify each variable as either a KEY field or a NAME (non-key) field. You can specify one or more columns (variable names) as keys for accessing the table. For the table shown in Table 3 on page 78, 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 can also be accessed by one or more “argument” variables that need not be key variables. You can define the variables that constitute the search argument dynamically by using the TBSARG and TBSCAN services.

In addition, a table can be accessed by use of the current row pointer (CRP). The table can be scanned by moving the CRP forward or backward. A row can be retrieved each time the CRP is moved. When a table is opened, the CRP is automatically positioned at TOP, ahead of the first row. Table services, such as TBTOP, TBBOTTOM, and TBSKIP are available for positioning the CRP.

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

When a row is stored, a list of “extension” variables can be specified by name. These extension variables, and their values, are added to the row. Thus, variables that were not specified when the table was created can be stored in the row. A list of extension variable names for a row can 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.

ISPF Table Services treat blank data and NULL (zero-length) data as equal. For example, the following VDEFINES are executed:

```
"ISPLINK('VDEFINE ','(V1)',VAL1,'CHAR ',L8,' NOBSCAN ')"
"ISPLINK('VDEFINE ','(V2)',VAL2,'CHAR ',L8)"
```

If L8 = 8, VAL1 = 'ABCD ' and VAL2 = 'ABCD ', V1 will have a length of 8 and a value of 'ABCD ', and V2 will have a length of 4 and a value of 'ABCD'. To ISPF, V1 and V2 will be equal because when ISPF compares two values, it pads the shorter value with blanks so that the lengths are equal, and then it does the compare.

If the same VDEFINES are done with VAL1 = ' ' and VAL2 = ' ', V1 will have a length of 8 and a value of ' ' (8 blanks), and V2 will have a length of 0 (NULL value). To ISPF, V1 is EQUAL to V2, since ISPF will pad V2 with 8 blanks before doing the compare to V1.

Services That Affect an Entire Table

The following services operate on an entire table:

TBCLOSE Closes a table and saves a permanent copy if the table was opened
TBCREATE Creates a new table and opens it for processing
TBEND Closes a table without saving
TBERASE Deletes a permanent table from the table output file
TBOPEN Opens an existing permanent table for processing
TBQUERY Obtains information about a table
TBSAVE Saves a permanent copy of a table without closing
TBSORT Sorts a table
TBSTATS Provides access to statistics for a table

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

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

Services That Affect Table Rows

The following services operate on a row of the table:

TBADD Adds a new row to the table.
TBBOTTOM Sets CRP to the last row and retrieves the row.
TBDELETE Deletes a row from the table.
TBEXIST Tests for the existence of a row (by key).
TBGET Retrieves a row from the table.
TBMOD Updates an existing row in the table. Otherwise, adds a new row to the table.
TBPUT Updates a row in the table if it exists and if the keys match.
TBSARG Establishes a search argument for use with TBSCAN. Can also be used in conjunction with TBDISPL.

TBSCAN Searches a table for a row that matches a list of "argument" variables, 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.

TBTOP Sets CRP to TOP, ahead of the first row.

TBVCLEAR Sets to null dialog variables that correspond to variables in the table.

Protecting Table Resources

Table services provide a resource protection mechanism designed to prevent concurrent updating of the same table by more than one user. This protection mechanism assumes that for all users having update access to a given table, the same library name is used in the first statement defining the table for the table library. This can be ISPTLIB or another specified library.

When a table is opened or created in write mode, an exclusive enqueue is requested for a resource name consisting of the first library name in the ISPTLIB, or other specified file, 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, or other specified library, 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 or specified library name is protected from concurrent output operations from any ISPF function through a separate mechanism not specific to table services.

Example: Create and Update a Simple Table

The following series of commands demonstrates the use of table services:

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
Pauly John	W590	Jones Beach
Clark Joan	Y200	Bar Harbour

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 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.

2. Save table DALPHA for later use by writing it to the table output library.

   ```
   ISPEXEC TBCLOSE DALPHA
   ```

   The table DALPHA is written from virtual storage to the file specified by ISPTABL.

Determining Table Size

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

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

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

The maximum number of rows allowed in a table is 16 777 215. However, dialog variables later used in processing can only keep a value of 999 999 as the maximum number of table rows. The total table size is the sum of the row lengths, plus the length of the data table control block (DTCB), plus the sort information record for sorted tables. The length of the DTCB can be computed as follows:

```
DTCB length = 152 + 16d
```

where:
d = total number of columns in the table, not including extension variables

The length of the sort information record can be computed as follows:

```
sort-information length = 12 + 8e
```

where:
e = number of sort fields

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

Example: Function Using the DISPLAY, TBGET, and TBADD Services

This topic describes the use of the DISPLAY, TBGET, and TBADD services in a dialog function that allows a user to add data to a table. A user can start the function by using the ISPSTART command. If the user has already started ISPF, the function can be started from:

- A menu
- The command field in any display with an application command that is defined in the current command table to have the SELECT action
- Another function by using the SELECT service.

During function processing, the DISPLAY service controls displays requesting the user to enter data about new employees. The data consists of:
- Employee serial number, entered on panel SER
- Name and phone number, entered on panel DATA.

Entered information is added to the table, as a row, through use of the TBADD service.

If the user enters an employee serial number for which an employee record already exists in the table, a DUPLICATE NUMBER short message displays on line 1 of panel SER. If the user enters the HELP command or presses the HELP Function key to get further explanation of this short message, the long message:

```
EMPLOYEE RECORD ALREADY EXISTS FOR THIS NUMBER. ENTER ANOTHER
```

displays on line 3 of the panel.

When the user successfully enters data for an employee, the short message NEW RECORD INSERTED is displayed on line 1 of panel SER. Then the user can enter the serial number of the next employee for which table data is to be added.

The user ends function processing by entering the END or RETURN command on any displayed panel or by pressing the END Function key or RETURN Function key.

The following section lists the complete function, followed by each statement with supporting text and figures.

Command Procedure Function

1. CONTROL ERRORS CANCEL
2. TBOPEN TAB1 WRITE
3. DISPLAY PANEL(SER)
4. if return code = 0, go to 6
5. if return code = 8, go to 21
6. TBGET TAB1
7. if return code = 0, go to 9
8. if return code = 8, go to 12
9. DISPLAY PANEL(SER) MSG(EMPX210)
10. if return code = 0, go to 6
11. if return code = 8, go to 21
12. Set dialog variables to blanks
13. DISPLAY PANEL(DATA)
14. if return code = 0, go to 16
15. if return code = 8, go to 21
16. TBADD TAB1
17. if return code = 0, go to 18
18. DISPLAY PANEL(SER) MSG(EMPX211)
19. if return code = 0, go to 6
20. if return code = 8, go to 21
21. TBCLOSE TAB1
22. End the function

Description of Function Steps

1. CONTROL ERRORS CANCEL

   This DM service request specifies that the function is to be terminated for a return code of 12 or higher from a DM service request.

2. TBOPEN TAB1 WRITE

   Open table TAB1 in update (WRITE) mode. Read table contents, shown in Table 3, into virtual storage. TAB1 is referred to by Steps 2, 6, 16, and 21.

Table 3. Five Rows in Table TAB1

EMPSER	LNAME	FNAME	I	PHA	PHNUM
598304	Robertson	Richard	P	301	840-1224
172397	Smith	Susan	A	301	547-8465
813058	Russell	Richard	L	202	338-9557
395733	Adams	John	Q	202	477-1776
502774	Kelvey	Ann	A	914	555-4156

3. DISPLAY PANEL(SER)

   This DISPLAY operation uses the panel definition SER, shown in Figure 27, to control the format and content of the panel display, shown in Figure 28 on page 79.

```
)BODY
%---------------------- EMPLOYEE SERIAL ----------------------------------%
%COMMAND ===>_ZCMD                                                        %
+
%ENTER EMPLOYEE SERIAL BELOW:
+
+
+   EMPLOYEE SERIAL%===>_EMPSER+   (MUST BE 6 NUMERIC DIGITS)
+
+
+
+PRESS%ENTER+TO DISPLAY NEXT SCREEN FOR ENTRY OF EMPLOYEE DATA.
+
+PRESS%END KEY+(PF3) TO END THIS SESSION.

)PROC
  VER (&EMPSER,NONBLANK,PICT,NNNNNN)

)END
```

Figure 27. Panel Definition SER

```
-------------------------  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 (PF3) TO END THIS SESSION.
```

Figure 28. Panel Display SER

Both the panel definition and the display are referred to in Steps 3, 9, and 18. The display requests that a serial number be entered for an employee. The user enters the serial number in the field labeled EMPLOYEE SERIAL NUMBER. The DISPLAY service then stores it in function pool variable EMPSER, and verifies it as specified on the panel definition. The verification is specified in a VER statement in the)PROC section of the panel definition, as shown in Figure 27 on page 78:

```
VER (&EMPSER,NONBLANK,PICT,NNNNNN)
```

This statement specifies that EMPSER must be nonblank and must consist of six digits, each in the range of 0–9.

When the input passes the verification, the DISPLAY service returns control to the function.

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. For example, if the user fails to enter the required employee serial number, the ENTER REQUIRED FIELD message is displayed, as shown in Figure 29 on page 80, and referred to in Steps 3 and 18.

```
---------------------  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 (PF3) TO END THIS SESSION.
```

Figure 29. Panel Display SER with an ISPF-provided Message Superimposed on Line 1

After the user re-enters the information, it is stored again in function pool variable EMPSER and reverified. The process is repeated until the information passes the verification tests.

4. if return code = 0, go to 6

   If the return code is 0, the display operation is successfully completed. Go to step 6 to verify that no record exists for this employee number.

5. if return code = 8, go to 21

   If the return code is 8, the END or RETURN command was entered on the display by the user. Go to step 21 to end processing.

6. TBGET TAB1

   This TBGET uses the employee serial number, stored in EMPSER in step 3 or 18, to attempt retrieval of an employee record from the TAB1 table. The table is a keyed table and has been created in another dialog by the service request:

   ```
   TBCREATE TAB1 KEYS(EMPSER) NAMES(LNAME FNAME I PHA PHNUM)
   ```

7. if return code = 0, go to 9

   A return code of 0 means that the record is found. Therefore, a record already exists for the employee serial number entered by the user. Go to step 9 to display the DUPLICATE NUMBER message.

8. if return code = 8, go to 12

   A return code of 8 means that no record is found. Go to step 12 to request the user to enter employee data.

9. DISPLAY PANEL(SER) MSG(EMPX210)

   This DISPLAY operation uses panel definition SER (Figure 27 on page 78) and message EMPX210, shown in Figure 30 on page 81 to control the format and content of the display. Figure 30 is referred to by steps 9, 13, and 18.

```
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 000-000'

EMPX213  'ENTER FIRST NAME'
'A FIRST NAME OR FIRST INITIAL IS REQUIRED.'

EMPX214  'ENTER LAST NAME'
'A LAST NAME IS REQUIRED.'
```

Figure 30. Message EMPX21

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 on line 1 of the panel display, shown in Figure 31.

```
---------------------  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 (PF3) TO END THIS SESSION.
```

Figure 31. Panel Display SER—Short Form of Message EMPX210 Superimpose Line 1

While viewing this message, the user can request the long form of the message by pressing the HELP Function key. The long form of the message

```
EMPLOYEE RECORD ALREADY EXISTS FOR THIS NUMBER. ENTER ANOTHER.
```

is superimposed on line 3 of the display. See Figure 32 on page 82.

```
---------------------  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 (PF3) TO END THIS SESSION.
```

Figure 32. Panel Display SER—Long Form of Message EMPX210 Superimposed on Line 3

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 3. After the input passes verification, the DISPLAY service returns control to the function.

10. if return code = 0, go to 6

    If the return code is 0, the display operation is successfully completed. Go to step 6 to verify that no record already exists for this employee number.

11. if return code = 8, go to 21

    If the return code is 8, the END or RETURN command was entered on the display by the user. Go to step 21 to end processing.

12. Set dialog variables to blanks

    These function pool variables are set to blank to prepare to receive data for a new employee record.

13. DISPLAY PANEL(DATA)

    The DISPLAY operation uses panel definition DATA, shown in Figure 33 on page 83, to control the format and content of the display shown in Figure 34 on page 84.

```
)BODY
%----------------------------  EMPLOYEE RECORDS  -----------------------%
%COMMAND ===>_ZCMD
+
%   EMPLOYEE SERIAL: &EMPSER
+
+   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+(PF3) 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 33. Panel Definition DATA

```
---------------------  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 (PF3) TO END THIS SESSION.
```

Figure 34. Panel Display DATA

The variables set to blank in step 12 are displayed, along with the new employee serial number entered in step 3 or 18. The user is asked to enter, in the blank 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 (Figure 33 on page 83).

If the input fields pass the verification tests, the DISPLAY service returns control to the function.

If the input fields fail the verification tests, a short-form message is displayed on line 1.

The message can be provided by ISPF, 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 33 on page 83. When 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 on line 3.

The messages request that information be re-entered. When re-entered, this information is stored again in function pool variables and reverified. The process is repeated until the verification tests are passed.

14. if return code = 0, go to 16

    If the return code is 0, the display operation is successfully completed. Go to step 16 to add the record to the table.

15. if return code = 8, go to 21

    If the return code is 8, the END or RETURN command was entered on the display by the user. Go to step 21 to end processing.

16. TBADD TAB1

    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 3 or 18, and employee name and phone number, stored in function pool variables LNAME, FNAME, I, PHA, and PHNUM by step 13. 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 request are the same as the names used in the definitions for panels on which the DISPLAY service accepts user input.

17. if return code = 0, go to 18

    If the return code is 0, the TBADD operation is successfully completed. Go to step 18 to display the NEW RECORD INSERTED message.

18. DISPLAY PANEL(SER) MSG(EMPX211)

    This DISPLAY operation uses panel definition SER (Figure 27 on page 78) and message EMPX211 (Figure 30 on page 81) to control the format and content of the display. The short form of message EMPX211, NEW RECORD INSERTED, is displayed on line 1. If the user enters the HELP command while this message is being displayed, the long form of the message (Figure 30 on page 81):

    ```
    ENTER SERIAL NUMBER FOR NEXT EMPLOYEE RECORD TO BE INSERTED
    ```

    is displayed on line 3.

    The user enters another serial number. The DISPLAY service verifies it as described in step 3. When the serial number passes the verification tests, the DISPLAY service returns control to the function.

19. if return code = 0, go to 6

    If the return code is 0, the display operation is successfully completed. Go to step 6 to verify that no record already exists for this employee number.

20. if return code = 8, go to 21

    If the return code is 8, the END or RETURN command was entered on the display by the user. Go to step 21 to end processing.

21. TBCLOSE TAB1

    Close the table TAB1. Write it from virtual storage to permanent storage.

22. End the function.

Specifying DBCS Search Argument Format for Table Services

For table services, you can specify either a DBCS or MIX (DBCS and EBCDIC) format string as a search argument. If either is used as a generic search argument, such as xxx* (any argument whose first three characters are ‘xxx’), the argument must be specified as follows:

- DBCS format string

  ```
  DBDBDBDB**
  ```

  where DBDBDBDB represents a 4-character DBCS string and ** is a single DBCS character representing the asterisk (*).

- MIX (DBCS and EBCDIC) format string

  ```
  eeee[DBDBDBDBDB]*
  ```

where eeee represents a 4-character EBCDIC string, DBDBDBDBDB represents a 5-character DBCS string, [and] represent shift-out and shift-in characters, and * is an asterisk in single-byte EBCDIC format.

Using the File-Tailoring Services

The file-tailoring services, listed in the order they are normally invoked, 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 an output file created by file tailoring.

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

The file-tailoring output can be directed to a file specified by the function, or it can be directed to a temporary sequential file provided by ISPF. The filename of the temporary file is available in system variable ZTEMPF. In MVS, ZTEMPF contains a data set name. The ddname of the temporary file is available in system variable ZTEMPN.

Skeleton Files

Each skeleton file is read record-by-record. Each record is scanned to find any dialog variable names, which are names preceded by an ampersand. When a variable name is found, its current value is substituted from a variable pool.

Skeleton file records can 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.

Problems can occur when using file-tailoring services in conjunction with other services (EDIT, COPY, ...) that result in modifying the data set members in the ISPSLIB concatenation. ISPSLIB is the input skeleton library, and it is assumed to be a static library. FTINCL obtains existing DCB/DEB information based on the last OPEN done against ISPSLIB by ISPF. It is recommended that applications that use file tailoring and modify members of ISPSLIB, use the LIBDEF service for ISPSLIB to point to the application's skeleton library.

Additionally, the application should check for any changes to the data set information DCB/DEB prior to invoking file-tailoring services. If there has been a

change, then the application should issue a NULL LIBDEF for ISPSLIB and then reissue the original LIBDEF for ISPSLIB. This will force a close and re-open of the ISPSLIB library.

Example of a Skeleton File

A sample skeleton file is shown in Figure 35. It contains MVS job control language (JCL) for an assembly and optional load-and-go. The tailored output could be submitted to the background for submission.

```
//ASM  EXEC            PGM=IFOXOO,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 35. Sample Skeleton File

The sample skeleton refers to several dialog variables (ASMPARMS, ASMIN, MEMBER, and so on) highlighted in the figure. 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 (that is, 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-and-go step. An imbed statement, “)IM”, is used to bring in a separate skeleton for the load-and-go step.

Example of Using File-Tailoring Services

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

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

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

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

- DALPHA is a member of the table library, containing:

  DALPHA

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

This example creates a name and address list. The file-tailoring service requests are:

- **ISPEXEC FTOPEN**

  ISPEXEC FTINCL LABLSKEL

  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.

  FTOPEN opens both the file-tailoring skeleton and file-tailoring output files. These files must be defined to ISPF before starting the ISPF session.

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

  You can issue multiple FTINCL commands to pull in more than one skeleton.

- **ISPEXEC FTCLOSE NAME (LABLOUT)**

  Write the resulting file-tailoring output to a member named LABLOUT SKELETON.

At the conclusion of processing the previous commands, file-tailoring output file LABOUT SKELETON contains:

```
     NAME:   Pauly John
APARTMENT:   W590
     CITY:   Jones Beach
     YEAR:   84
     NAME:   Clark Joan
APARTMENT:   Y200
     CITY:   Bar Harbour
     YEAR:   84
```

Using the PDF Services

PDF services consist of the BRIF (Browse Interface), BROWSE, EDIF (Edit Interface), EDIREC (edit recovery for EDIF), EDIT, and EDREC (edit recovery for EDIT) services and a set of library access services.

BROWSE, EDIT, and EDREC

The BROWSE and EDIT services allow you to create, read, or change MVS data sets or members of an ISPF library. An *ISPF library* is a cataloged partitioned data set with a three-level name made up of a project, a group, and type. The ISPF library can be private (available only to you) or can be shared by a group of users. The BROWSE and EDIT *services* provide direct access to the Browse and Edit *options* of PDF, bypassing the Browse mode on the View Entry panel and Edit Entry panels.

The EDREC service, which you usually invoke *before* calling EDIT, helps you recover work that would otherwise be lost if ISPF ended abnormally, such as after a power loss.

Refer to the *ISPF Services Guide* for complete descriptions, including examples, of the BROWSE, EDIT, and EDREC services.

BRIF, EDIF, and EDIREC

Two services, the Browse Interface (BRIF) service and the Edit Interface (EDIF) service, allow dialogs to provide their own I/O for PDF Browse and Edit. These services provide edit and browse functions for data accessed through dialog-supplied I/O routines. BRIF and EDIF require that the invoking dialog perform all environmentally-dependent functions (such as allocating, opening, reading, writing, closing, and freeing files).

Use of the BRIF and EDIF services allows the type of data and data access methods being employed by a dialog to be transparent to Browse and Edit. The Edit Interface Recovery (EDIREC) service performs edit recovery for EDIF.

These services make it possible to implement functions such as:
- Edit/browse of data other than partitioned data sets or sequential files
- Edit/browse of in-storage data
- Pre- and post-processing of edited or browsed data.

Refer to *ISPF Services Guide* for descriptions and examples of BRIF, EDIF, and EDIREC.

Library Access Services

The library access services can interact with the BROWSE and EDIT services and can also give you access to ISPF libraries and to certain system data sets. These services carry out functions such as opening a library, copying a library or library member, and displaying a library's members.

The library access services can also interact with certain LMF options that allow a dialog with proper authority to activate library controls, promote members to a controlled library, and query information about library controls.

You can use the library access services with four types of libraries or data sets:
- An ISPF library known by project, group, and type
- A concatenated set of up to four ISPF libraries
- A single existing TSO or MVS partitioned or sequential data set
- A concatenated set of up to four MVS partitioned data sets.

The library access services support only those data sets that reside on a single DASD volume with record format types F, FB, V, VB, and U. The data set organization must be either partitioned or sequential. *ISPF User's Guide* contains an explanation of the ISPF library structure.

Refer to the *ISPF Services Guide* for complete descriptions, including examples, of the library access services.

Another way you can maintain different levels or versions of a library member is to use the software configuration and library manager (SCLM) utilities. SCLM is a software tool that helps you develop complex software applications. Throughout the development cycle, SCLM automatically controls, maintains, and tracks all of the software components of the application. And, you can lock the version being edited in a private library and then promote it to another group within the library for further development or testing. Refer to *ISPF Software Configuration and Library Manager (SCLM) Developer's and Project Manager's Guide* for more information about SCLM.

Note: Some library access services perform LMF functions. LMCOPY (with LOCK option), LMPROM, and LMMFIND (with LOCK option) will fail if one of the specified libraries is SCLM-controlled: LMF uses the following rule to determine whether a library is SCLM-controlled. If there is a data set with a name of the form '<project>.PROJDEFS.LOAD' such that <project> is the high-level qualifier (or project) of the library in question, then the library is SCLM-controlled; otherwise, the library is not SCLM-controlled. Note also that this means that LMF may treat a library as if it were SCLM-controlled even though it may not be SCLM-controlled.

For example, if a data set existed named 'PDFUSER.PROJDEFS.LOAD', then LMF would treat each of the following data sets as if they were SCLM-controlled:

```
'PDFUSER.LIST'
'PDFUSER.DEV.PANELS'
'PDFUSER.MAINT.NEW.SKELS'
```

Where to Find Examples and Listings of PDF Services

ISPF Examples introduces the PDF services through the following type examples:
- Copying a member from one ISPF library to another, using a CLIST.

- Displaying a list of members of an ISPF library for renaming, deleting, and setting statistics, using PL/I.
- Locking and promoting a member of an LMF-controlled hierarchy, using PL/I and TSO/REXX.
- Using a combination of COBOL and Assembler routines to do the following:
  - Browse an ISPF table with the BRIF service. Included are the main dialog, read, command, and Assembler interface routines.
  - Edit an ISPF table with the EDIF service. Included are the main dialog, read, write, command, and Assembler interface routines.
- Using PL/I with the EDIF service with an Assembler program to pass return codes from the PL/I routine to the EDIF service.

Using the Miscellaneous Services

ISPF provides the CONTROL, GDDM, GETMSG, LIBDEF, LIST, LOG, and PQUERY services, briefly described in the following paragraphs. You can find more information about these services in the following chapter.

CONTROL Service

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

LINE
Expect line output to be generated by the dialog or by execution of a TSO command. Optionally, the starting line can be specified.

LOCK
Allow the next display without unlocking the terminal keyboard. LOCK is generally used with the DISPLAY service to overlay a currently displayed panel with an “in-process” message; for example:

```
DISPLAY PANEL(panel-name)
⋮

CONTROL DISPLAY LOCK
DISPLAY MSG (message-id)
⋮
```

NONDISPL
Do not display the next panel. Process the panel without actually displaying it, and simulate the Enter key or END command.

REFRESH
Refresh the entire screen on the next display. Typically used before or after invoking some other full-screen application that is not using DM display services.

SPLIT
Enable or disable split-screen operation by a user as required by the application.

The disposition of errors can be controlled as follows:

CANCEL
Terminate the function on an error with a return code 12 or higher from any service. A message is displayed and logged prior to termination.

RETURN
Return control to the function on all errors, with appropriate return code. A message ID is stored in system variable ZERRMSG, which can be used by the function to display or log a message.

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

GDDM Services

The graphics initialization (GRINIT) service initializes the ISPF/GDDM interface and optionally requests that ISPF define a panel's graphic area as a GDDM graphics field. The graphics termination (GRTERM) service terminates a previously-established GDDM interface. The graphics error block (GRERROR) service provides access to the address of the GDDM error record and the address of the GDDM call format descriptor module.

GETMSG Service

The GETMSG service obtains a message and related information and stores them in variables specified in the service request.

LIBDEF Service

The LIBDEF service provides applications with a method of dynamically defining application data element files while in an active ISPF session.

LIST Service

The LIST service allows a dialog to write data lines directly (without using print commands or utilities) to the ISPF list data set. You specify the name of the dialog variable containing the data to be written on the LIST service request.

LOG Service

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

PQUERY Service

The PQUERY service returns information for a specific area on a specific panel. The type, size, and position characteristics associated with the area are returned in variables.

Chapter 4. Common User Access (CUA) Guidelines

This chapter briefly describes how ISPF supports the Common User Access (CUA) guidelines.

The CUA 89 guidelines define a user interface in terms of common elements, such as the way information appears on a screen, and interaction techniques, such as the way users respond to what appears on a screen. Refer to the *SAA CUA Basic Interface Design Guide* ISPF supports the CUA guidelines in the following ways. You can:

- Define a list of function keys to be associated with each panel.
- Define an action bar and pull-downs on a panel.
- Define and display pop-up windows.
- Define and display help panels for field-level help, extended help, and keys help. See "Chapter 7. ISPF Help and Tutorial Panels" on page 281 for more information on CUA help panels.

With ISPF, the panel ID is displayed according to CUA defaults and the PANELID command acts as a toggle.

ISPF also lets you indicate, for an application session, if you want to use CUA defaults. If selected, the Panel display CUA mode option on the ISPF Settings panel controls:

- The location of the function keys on the panel in relation to the command and message lines.
- The appearance and display format of the keys.

Using the Dialog Tag Language to Define Dialog Elements

The Dialog Tag Language (DTL) is a set of markup language tags that you can use to define dialog elements. You can use DTL tags in addition to or instead of ISPF methods for defining panels, messages, and command tables. In addition, when you define a panel using DTL tags, you can assign a specific keylist to be associated with and displayed on that panel, if requested by the user.

The DTL defines the source information for the dialog elements, and the ISPF dialog tag language conversion utility converts the source file to a format ISPF understands. The *ISPF Dialog Tag Language Guide and Reference* explains in detail how to create the various elements using the DTL and ISPF conversion utility.

Keylists

The key assignments active for an application panel are defined and stored within keylists. These key assignments allow the user to request commands and other actions through the use of function keys. Key assignments for your application are displayed in the function key area of application panels. Keylists can be *shared* across all users by defining them using DTL. This creates an xxxxKEYS table that is placed in the ISPTLIB concatenation. Users can modify keylists using the KEYS and KEYLIST commands. Both commands invoke the Keylist utility. Modifications to keylists are stored in the user's application profile, thus they are called *private*.

© Copyright IBM Corp. 1980, 1999

You can view or modify keylists either through the KEYLIST command or the **Keylist settings...** choice from the Function keys pull-down on the ISPF Settings panel. You can control whether your application uses keylists or not with the KEYLIST command or the **Keylist settings...** choice from the Function keys pull-down on the ISPF Settings panel. You can also control whether you use keylists as shipped with the application or with user modifications. You assign the keylist to a particular panel by using the keylist keyword on the)PANEL statement or by using the keylist attribute on the PANEL tag. For a description of the panel section, see "Defining the Panel Section" on page 222.

Action Bars and Pull-Downs

An action bar is the panel element located at the top of an application panel that contains action bar choices for the panel. Each action bar choice represents a group of related choices that appear in the pull-down associated with the action bar choice. When the user selects an action bar choice, the associated pull-down appears directly below the action bar choice. Pull-downs contain choices that, when selected by the user, perform actions that apply to the contents of the panel.

For complete details on coding action bars and pull-downs, refer to the *ISPF Dialog Tag Language Guide and Reference* or the "Defining the Action Bar Choice Section" on page 161.

Pop-Up Windows

Pop-up windows display information that extends the user's interaction with the underlying panel. When a pop-up is displayed, the user must finish interacting with that pop-up window before continuing with the dialog in the underlying panel.

The ADDPOP service allows your application to use pop-up windows. After you issue the ADDPOP service, subsequent DISPLAY, TBDISPL, or SELECT service calls display panels in that pop-up window until your application issues a corresponding REMPOP service or issues another ADDPOP service.

You specify the location of the pop-up window using the ADDPOP service call.

Note: When you are running in GUI mode, this pop-up window location specification is ignored. Default positioning is used.

You can specify the size of the window (width and depth) on the panel definition BODY statement or use the WIDTH and DEPTH attributes on the DTL PANEL tag. If you do not specify the size, the Dialog Manager displays the pop-up window in a 76 X 22 window with a border.

Each pop-up window created as a result of a successful ADDPOP service call can also have a window title. The title is imbedded in the top of the window frame border and can be only one line in length. If the title is longer than the window frame, the dialog manager truncates it. To define the window title, set system variable ZWINTTL to the desired window title text.

Note: If you are running in GUI mode, the value in ZWINTTL has a maximum length of 255 characters and will be truncated without notice to the user at display time if it does not fit on the panel.

The following example will display three pop-up windows, as shown in Figure 36. The window that panel B is displayed within will have the title *POPUP WINDOW TITLE*.

```
PROC 0
ISPEXEC ADDPOP
ISPEXEC DISPLAY PANEL(A)
ISPEXEC ADDPOP POPLOC(F1)
SET ZWINTTL = POPUP WINDOW TITLE
ISPEXEC DISPLAY PANEL(B)
SET ZWINTTL =
ISPEXEC ADDPOP
ISPEXEC DISPLAY PANEL(C)
```

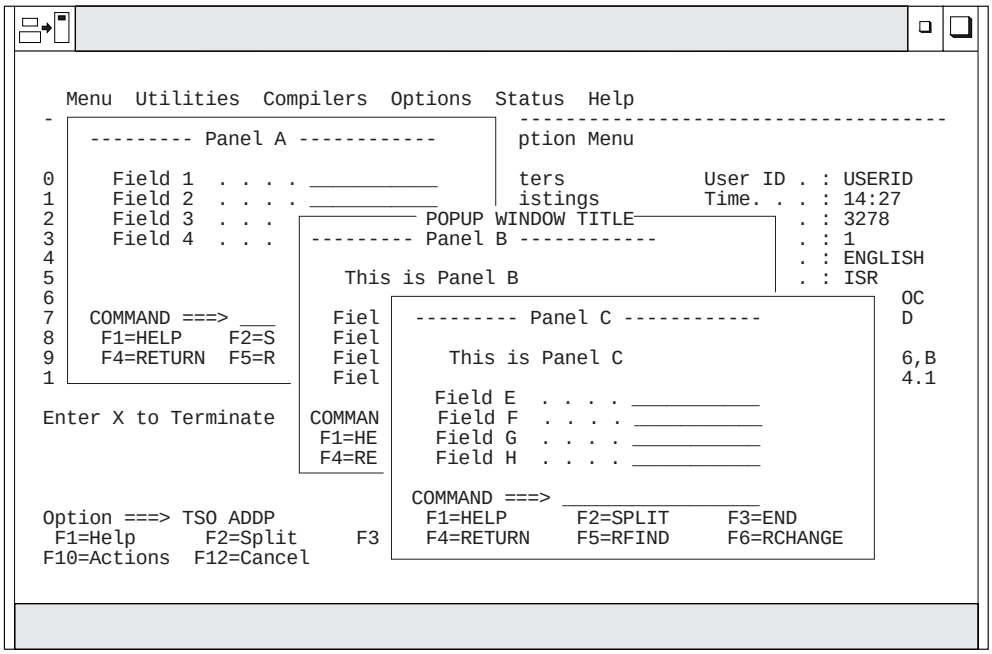


Figure 36. Example Panel Displaying Three Pop-Up Windows

The REMPOP service removes the current pop-up window. After you call the REMPOP service, a subsequent DISPLAY service will either display a panel in the full panel area of the screen or in a lower-level pop-up window, if it is active.

Refer to *ISPF Services Guide* for a complete description of the ADDPOP and REMPOP services.

Moveable Pop-Ups

ISPF provides two distinct mechanisms for you to move the currently active pop-up window: the WINDOW command or manual movement using two terminal interactions and no specific ISPF command. You can also move the window with any other method you normally use to move windows on your workstation.

Note: The WINDOW command is disabled if you are running in GUI mode.

WINDOW Command

The WINDOW command can be associated with a function key or can be typed on the command line. The cursor placement specifies the new location for the upper-left corner of the pop-up window frame. If the pop-up window does not fit on the physical screen at the specified location, it is repositioned to fit following the current pop-up window positioning rules. The cursor is placed in the same relative position it occupied prior to a dialog or help pop-up window being moved.

If the cursor location would be covered as a result of moving a modeless message window, the cursor is repositioned to the first input field on the active panel. If an input field does not exist, the cursor is positioned in the upper-left corner of the active panel. The cursor is returned to its intended location if the modeless message window is moved to a location that no longer conflicts with cursor display. Cursor positioning is not affected by an input field that becomes protected as a result of a modeless message window position unless the cursor itself would be covered. In other words, the cursor can be positioned on a protected input field.

The WINDOW command is an immediate action command. Panel processing is not performed when this command is used.

If the WINDOW command is typed in the command line, the cursor should be moved to the desired window position prior to pressing Enter.

If the WINDOW command is included in the keylist associated with the currently active application panel, the user can move the cursor to any position on the screen, press the function key assigned to the WINDOW command, and the pop-up is repositioned to the user's cursor position. The WINDOW command can be included in the keylist by the application developer, or the user can use the KEYLIST utility to add it to the keylist.

For panels that do not include the KEYLIST keyword in the)PANEL statement, the application can assign the WINDOW command to a ZPFnn system variable. The user can also associate WINDOW with a function key by using the ZKEYS command to access the function key assignment utility.

If the split screen is used, the pop-up cannot be moved to a different logical screen. The new pop-up window location must be in the same logical screen in which the pop-up was originally located. A pop-up is not displayed over the split line. The split line cuts off the pop-up at the split line location; the pop-up is not automatically repositioned to fit above the split line.

Note: Pull Down Choice (PDC), Action Bar is also a pop-up window, so the split screen line cuts off the Action Bar location, too. The pop-up is not automatically repositioned to fit above the split line.

If the WINDOW command is requested when pop-up windows are not active, a message is displayed to the user. A pop-up window containing an Action Bar panel cannot be moved while a pull-down is actively displayed. A message is displayed to the user if the WINDOW command is requested during this condition.

Manual Movement

The second method for moving pop-up windows involves two terminal interactions but does not require a unique ISPF command. A user can request window movement by placing the cursor anywhere on the active window frame and pressing Enter. ISPF acknowledges the window move request by displaying WINDOW MOVE PENDING message. The alarm will sound if the terminal is so equipped. The message text will be yellow/high intensity if the Panel display CUA mode option on the ISPF Settings panel has been selected. Otherwise, the message text will be white/low intensity.

Place the cursor where you want the upper-left corner of the window frame placed and press Enter a second time. The window is moved to the new location just as though the WINDOW command had been issued. The rules for cursor placement inside the window, and window placement on the physical display, are the same as those described for the WINDOW command.

Pop-Up Movement Considerations

Modeless and modal message pop-up windows can be moved in the same manner as dialog pop-up windows.

Only the active pop-up window can be moved. If a modal or modeless message pop-up is displayed over a dialog pop-up window, only the message pop-up window can be moved. The underlying dialog pop-up window cannot be moved while a message pop-up window is displayed over it.

Input fields that are partially or totally covered by a pop-up window become protected fields (data cannot be entered into the field). If a field becomes totally uncovered as a result of moving the pop-up window, the field is restored to an unprotected field (data can be entered into the field).

Field-Level Help

Field-level help provides help panels for fields defined on an application panel. When the cursor is on a field and the user requests HELP, ISPF displays the help panel defined for that field. See “Defining the HELP Section” on page 219.

Extended Help

Extended help provides general information about the contents of a panel. The information in extended help can be an overall explanation of items on the panel, an explanation of the panel’s purpose in the application, or instructions for the user to interact with the panel. The user invokes extended help by issuing the command EXHELP. EXHELP requests ISPF to display help text for the entire panel.

For more information on help, see “.HELP” on page 275 and “Chapter 7. ISPF Help and Tutorial Panels” on page 281.

Keys Help

Keys help provides the user with a brief description of each key defined for a panel. You define the contents of this help panel. The user invokes keys help by issuing the command KEYSHELP.

KEYSHELP requests ISPF to display the help panel for the current keylist. The help panel name can be provided as part of the keylist definition. If the keys help panel is not identified in the keylist definition, it can be supplied in the ZKEYHELP system variable. Use separate ZKEYHELP variable values for each keys help panel to be displayed.

Reference Phrase Help

Reference phrase (RP) help is available on all panels. Place the cursor on a highlighted reference phrase within a panel, request help, and you receive the help panel defined for that reference phrase.

When a panel with reference phrases is displayed for the first time, the cursor is positioned in the upper-left corner. After a reference phrase is selected and control is returned to the original panel, the panel scrolls automatically to put the cursor on the reference phrase from which the reference phrase help was invoked. The exact scroll position might not be the same as when the reference phrase help was invoked. ISPF positions the reference phrase at the top of the display is scrolling is necessary to display the reference phrase help field. The reference phrase is an input-capable field that allows tabbing. Therefore, the reference phrase text is refreshed whenever the panel is redisplayed.

Reference phrase help panels themselves can also contain reference phrases. When a reference phrase help panel is cancelled, the panel from which reference phrase help was requested is redisplayed. All other help facilities are available from a reference phrase help panel.

The TYPE(RP) attribute in the panel attribute section is used to identify a reference phrase in a panel. See “Defining the Attribute Section” on page 176. An entry is then placed in the)HELP section of the panel for each reference phrase attribute coded in the)BODY or optional)AREA panel sections. The following example is a)HELP section reference phrase definition:

```
)HELP
  FIELD(ZRPxxyyy)  PANEL(panel-name)
```

xx 00 for a reference phrase defined in)BODY section and 01 to 99 for the number of the scrollable area in which the reference phrase is defined.

Each scrollable area is assigned a sequential number based on its relative position within the panel body. The scrollable area closest to the upper-left corner of the panel body is assigned number 01 with each additional scrollable area, scanning left to right, top to bottom, assigned the next sequential number. A maximum of 99 scrollable areas in any given panel may contain reference phrases.

yyy 001 to 999 for the relative number of the reference phrase within the panel body or within a particular scrollable area.

panel-name
Name of the help panel to be displayed when HELP for this reference phrase is requested.

A reference phrase can wrap around multiple terminal lines in panels that are not displayed in a window. A reference phrase that logically wraps in a pop-up window requires the beginning of each wrapped line to contain a RP field attribute, and there must be an entry in the)HELP section for each wrapped line. This is also true for panels containing the WINDOW() keyword that are not displayed in a pop-up window. The additional)HELP section entries would normally be pointing to the same panel.

The example in Figure 37 illustrates both single and multiple line reference phrases.

```
)PANEL
)ATTR
  #  TYPE(RP)
  $  AREA(SCRL) EXTEND(OFF)
)BODY
+This is sample text.  This is a #Reference Phrase+.
+This is an example of a #Reference Phrase being
 physically continued to the next line.+
+  *********************
+  *$SAREA1           $*      ****************
+  *$                 $*      *$SAREA2      $*
+  *$                 $*      *$            $*
+  *********************      *$            $*
+  *********************      *$            $*
+  *$SAREA3           $*      *$            $*
+  *$                 $*      ****************
+  *$                 $*
+  *********************
+This is an example of a #Reference Phrase being+
#logically continued to the next line.+
+
)AREA SAREA1
+                  +
#Area 01 Ref Phrase+
+                  +
)AREA SAREA2
+            +
+  #Area 02+ +
+ #Reference++
+  #Phrase+  +
)AREA SAREA3
+                  +
#Area 03 Ref Phrase+
+                  +
)HELP
 FIELD(ZRP00001)  PANEL(BODY0001)
 FIELD(ZRP00002)  PANEL(BODY0002)
 FIELD(ZRP00003)  PANEL(BODY0003)
 FIELD(ZRP00004)  PANEL(BODY0003)
 FIELD(ZRP01001)  PANEL(AREA0101)
 FIELD(ZRP02001)  PANEL(AREA0201)
 FIELD(ZRP02002)  PANEL(AREA0201)
 FIELD(ZRP02003)  PANEL(AREA0201)
 FIELD(ZRP03001)  PANEL(AREA0301)
)END
```

Figure 37. Reference Phrase Help Example

START Service

You can use the START service to start a dialog in a new logical screen. This function is similar to the function nesting made available with action bars except that the "nesting" occurs in a new logical screen.

You can invoke the START service in any of the following ways:

- From any command line, you can enter the command

  ```
  START some_dialog
  ```

  where *some_dialog* can be any of the following:
  - A command from the command table; for example, `MYCMD1`
  - A command with parameters (must be in quotes); for example, `'MYCMD1 PARM1'`
  - A dialog invocation; for example, `PANEL(MYPAN1)`, or `'PGM(MYPGM1) PARM(MYPARM1,MYPARM2)'`
- You can code a pull-down choice,

  ```
  ACTION RUN(START) PARM(some_dialog)
  ```

  where *some_dialog* is the same as outlined above.
- You can code a selection panel option,

  ```
  'PGM(ISPSTRT) PARM(some_dialog)'
  ```

  For example,

  ```
  &ZSEL = TRANS(&XX
          0,'PGM(ISPSTRT) PARM(PGM(MYPGM0))'
          1,'PGM(ISPSTRT) PARM(PGM(MYPGM1) PARM(MYPARM1))'
          2,'PGM(ISPSTRT) PARM(MYCMD1 MYPARM2)'
          3,'PGM(ISPSTRT) PARM(PANEL(MYPANEL1))'
  ```
- From a dialog, you can invoke,

  ```
  ISPEXEC SELECT PGM(ISPSTRT) PARM(some_dialog)
  ```

  where *some_dialog* is the same as described above.

  Note: The *some_dialog* should not exceed 249 characters. It will be truncated at 249 without warning. You should *not* use either WSCMD or WSCMDV in your specification of *some_dialog*.

Note: For ISPF functions that have service interfaces, such as EDIT and BROWSE, you should use the service invocations. Using ISPSTRT passing the selection strings from panel ISR@PRIM does not work in all situations and is not supported.

If the maximum number of logical screens do not exist when the START command is invoked and:

- *some_dialog* is a command from the command table, the new screen is invoked with the default initial command (in non-display mode) and the command is run. When the user ends the dialog this new screen still exists.
- if *some_dialog* is specified as PGM(xxx), CMD(xxx), or PANEL(xxx), the new screen is invoked with PGM(xxx), CMD(xxx), or PANEL(xxx) as the initial command, program, or panel. The result is that when you end the xxx dialog, this new screen is terminated.

If the maximum number of logical screens has already been reached when the START command is invoked, the specified *some_dialog* will be executed on top of the currently displayed screen. The result is that when you end the dialog, ISPF returns to the previously-displayed screen.

On 3270 displays, if ISPF is not in split screen mode the START command and ISPSTRT program split the screen at extreme top or bottom of the display. If ISPF is already in split screen mode, ISPF starts the new screen in the opposite screen, using the existing split line location.

Chapter 5. Panel Definition Statement Guide

This chapter provides guide-type information for sections, panel definition statements, and control variables.

You can create ISPF panels in one of three ways:

- Use the Dialog Tag Language (DTL) and ISPF DTL conversion utility only. With DTL, you create a source file containing DTL tags that define what information you want for each panel. This source file is then processed through the ISPF conversion utility to produce a pre-processed ISPF panel library member ready for display.
- Use DTL and panel definition statements. This option allows you to stop the conversion process at the ISPF panel definition source level. You can then edit the resulting panel definition source file using any of the panel definition statements available in this book.
- Use panel definition statements only. Using panel definition statements, you define panels closely resembling the finished panel. Each character position in the panel definition corresponds to the same relative position on the display screen.

To create panels with DTL or to learn how to capture the panel definition source file, refer to the *ISPF Dialog Tag Language Guide and Reference*

This chapter explains how to create panels using the panel definition statements. (This information applies to the second and third options described above.) Both general overview information on panel definition and specific information on each panel section is included. The topics are arranged as follows:

- An introduction to the panel definition sections
- General tips and guidelines for formatting panels
- Syntax rules and restrictions for panel definition
- A discussion of each panel section
- Using Z variables as field name placeholders
- Panel processing considerations
- Support for panel user exit routines
- Special requirements for defining menus, table display panels, and panels with dynamic or graphic areas.

Figure 39 on page 109 shows an example panel definition which uses CUA panel-element attributes. See Figure 66 on page 217 for an example panel definition that does not use CUA panel-element attributes.

Note: When not in TEST mode, the most recently accessed panel definitions are retained in virtual storage for performance reasons. If you have modified a panel, using TEST mode will ensure that the updated version of the panel will be picked up by ISPF services. See “ISPF Test and Trace Modes” on page 25 for more information.

© Copyright IBM Corp. 1980, 1999

Introduction to Panel Definition Sections

Each panel definition consists of various combinations of the sections described in Table 4. The sections you choose to use must be in the order listed in this table.

Table 4. Panel Definition Sections

Section	Optional	Required	Description
)CCSID	Yes		CCSID section. Specifies the Coded Character Set Identifier (CCSID) used in the panel definition. If used, panel text characters are translated to the terminal code page for display.
)PANEL	Yes		Panel section. Specifies a keylist to be used during the display of the panel. Identifies where the keylist is to be found. Specifies that the panel is to be displayed in CUA mode.
)ATTR	Yes		Attribute section. Defines the special characters in the body of the panel definition that represent attribute (start of field) bytes. You can override the default attribute characters provided with ISPF.
)ABC	Yes		Action bar choice section. Defines a choice in the action bar, its associated pull-down choices, and the actions to be taken for each pull-down choice.
)ABCINIT		Yes, if)ABC is specified	Action bar choice initialization section. Specifies processing that is to occur for an action bar choice prior to the display of the panel.
)ABCPROC			Action bar choice processing section. Specifies processing that is to occur for an action bar when the panel is submitted for processing.
)BODY		Yes	Body section. Defines the format of the panel as seen by the user and defines the name of each variable field on the panel.
)MODEL		Yes, for table display	Model section. Defines the format of each row of scrollable data. This section is required for table display panels. Only one)MODEL section is allowed per panel.
)AREA	Yes		Scrollable area definition section. Defines a scrollable section of the panel.
)INIT	Yes		Initialization section. 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.
)REINIT	Yes		Reinitialization section. Specifies processing that is to occur prior to redisplay of a panel.
)PROC	Yes		Processing section. Specifies processing that is to occur after the panel has been displayed or redisplayed. This section is typically used to define how variables are to be verified and translated.
)HELP	Yes		Help section for field help. Specifies the help panels to display when help is requested for a field, list column, action bar choice, or pull-down choice defined in the panel or reference phrase.
)PNTS	Yes		Point-and-shoot section. Contains an entry for each field on a panel that has been designated as a point-and-shoot field.
)LIST	Yes		List section. Specifies a list to build on the panel.

Table 4. Panel Definition Sections (continued)

)END	Yes	End section. Specifies the end of the panel definition. The)END (end) section consists only of the)END statement. ISPF ignores any data that appears on lines following the)END statement.

Guidelines for Formatting Panels

Once ISPF is installed, consider using its edit model facilities to help you create panel definitions.

When using Edit to create a panel definition, specify NUMBER OFF to prevent numbers from appearing in the file. Numbers cause a panel syntax error when you attempt to process the panel definition.

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 referred to by its name, which is the same as the member name in the library.

You can create or change panel definitions by editing directly into the panel library. No compile or preprocessing step is required. Use the name of this panel library member as the panel-name parameter when requesting dialog services, such as DISPLAY and SELECT.

As shown in Figure 38, the first three displayable lines below the action bar, if present, in a panel definition include:

- Panel ID and title area
- System-defined (default) areas for message display
- A command/option field
- A scroll field, if applicable.

You can override the location of the long message area and command field from the ISPF Settings panel.

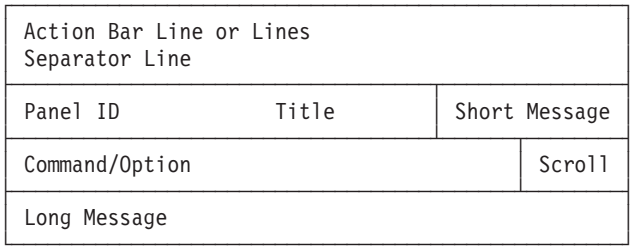


Figure 38. Sample Panel Definition Format

Action Bar Line

The action bar line displays the action bar choice-description-text. You can define multiple action bars for a panel. A separator line should follow the last action bar line. ISPF considers the panel line following the last action bar choice as part of the action bar area. See "Defining the Action Bar Choice Section" on page 161.

Title Line

The title line should contain a centered title indicating the function being performed or, where appropriate, should display information critical to that function. If not running in the GUI mode, the panel ID is added to the beginning of this line if a panel ID is requested by using the PANELID system command. The right-hand 26 characters of this line should not contain critical information if short messages are to be used in the default short message area.

Short Messages

If short messages are used, they should provide a brief indication of either:

- Successful completion of a processing function
- Error conditions, accompanied by audible alarm.

Short messages temporarily overlay information 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.

Use short messages consistently throughout the application, or not at all.

For table display, the short message area contains a top-row-displayed indicator, except when overlaid by a function-requested message. The TBDISPL service automatically generates this indicator, and replaces data that was in the panel definition in that area. Attribute bytes in the short message area can cause the top-row displayed indicator to be unreadable.

Command/Option Line

The command/option line 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 the keyword CMD on the header of the panel body section, can be named using any valid variable name. However, the name ZCMD is generally used.

Cursor placement for viewing a panel differs, depending on the use of the name ZCMD or other names. When you use ZCMD and cursor placement is not explicitly specified, ISPF skips over a blank command field to place the cursor on a following input field. When you use a name other than ZCMD, ISPF does not skip over a blank command field when placing the cursor during display.

Scroll Amount

For table display, Edit, and Browse panels, as well as panels with scrollable dynamic areas, the scroll amount field should be on the right-hand side of the command line. The scroll amount field must be the first input field following the command field and must be exactly 4 characters in length. A scroll amount field is not meaningful for other types of panels and should be omitted from them.

Long Messages

The long message line 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. An input field, such as the command field, should not be located on the same line on which long messages are displayed. The display of long messages will be superimposed on the input field, and results are unpredictable.

Requirements for Specifying Message and Command Line Placement

The placement of the command line and long message field at the bottom of a logical screen is a user-definable option. Placement is controlled by the system variable ZPLACE. You can select or deselect Command line at bottom on the ISPF Settings panel, and the setting changes the value of ZPLACE. ZPLACE can also be changed in a dialog.

The value of ZPLACE is stored in the application profile pool. To change the value, use the VPUT statement in a panel definition, the VPUT service in a dialog function, or the ISPF Settings panel options. None of these settings takes priority over the others. For example, an ISPF Settings panel selection can change what a dialog set, and vice versa.

If the panel specifies ASIS on the)BODY statement for a panel, the command and message lines are not repositioned, even if you specify placement at the BOTTOM. The command line moves only if all of the following are true.

- **For primary windows**:
  1. If the WINDOW(*w*,*d*) keyword is specified on the header statement where *w* is less than the screen width, then:
     a. The keyword ASIS must not be specified on the)BODY header statement.
     b. The first character of the command line must be an attribute character.
  2. If the WINDOW(*w*,*d*) keyword is specified on the header statement where *w* is equal to the screen width or the WINDOW keyword is not specified, then:
     a. The keyword ASIS must not be specified on the)BODY header statement.
     b. The first and last character of the command line must be an attribute character, and one of the following is true:
        1) There is an attribute byte in the first column of the line following the command line.
        2) There is an attribute byte in the last column of the line preceding the command line.
  3. **For pop-up windows**, the keyword ASIS is not specified on the)BODY header statement.

Command lines that move in panels designed for primary windows will continue to move if these panels are displayed in pop-up windows. In addition, command lines in panels created using the DTL and converted using the ISPF conversion utility will move in both primary and pop-up windows.

If requirement 2b1 is false, but 2b2 is true, ISPF changes the attribute byte in the last column of the line preceding the command line to match the attribute byte in the last column of the command line. This gives the same result as 2b1.

For the long message line to be moved, the panel must be designed so that the system default is used to position the long message. That is, an alternate long message field cannot be specified by the panel designer using the keyword 'LMSG' on the)BODY header statement.

The long message line is not moved unless the command line is moved, but the command line is moved regardless of whether the long message field is moved.

Additional Suggestions for Designing Panels

- Avoid overly cluttered panels. Split *busy* panels into two or more simple panels that have less information and are easier to read. Use scrollable areas where appropriate.
- Do not use the last available line in a panel body. For example, if the dialog can be used on 24-line terminals, limit the body to 23 lines, or less. This is because in split-screen mode the maximum length of a logical screen is one less than the length of the physical screen.

  The PFSHOW|FKA command usually requires a minimum of two lines of a panel for displaying function key status. Therefore, you should leave the bottom two panel lines blank.
- Place important input fields near the top of the panel and less important fields, especially optional input fields, further down. In split-screen mode, the bottom of the panel might not be visible unless you reposition the split line.
- Place important input fields near the top of a scrollable area to minimize the need for scrolling.
- Place the command line near the top of the panel. If the command line is near the bottom and you enter split-screen mode, the command line cannot be visible on the screen. Therefore, if you do not have function keys, you might not be able to continue processing the dialog. If, for a particular session, you will not be entering the split-screen mode, you can use the option 0 (Settings) to specify that the command line be placed at the bottom of the screen. However, if you want to place the command line at the bottom, use the ZPLACE system variable.
- Where practical, align fields vertically on a panel, especially input fields. Group related input fields under a common heading. Minimize the use of multiple input fields on the same line, so that the NEW LINE key can be used to skip from one input field to the next.
- Use visual signals to indicate particular types of fields, such as arrows to indicate input fields, and colons to indicate variable information that is protected. Examples:

  ```
  FILE NAME ===> (arrow signals an input field)

  EMPLOYEE SERIAL:  123456 (colon signals a protected field)
  ```

  To conform to the CUA guidelines, use leader dots and an ending colon for all protected fields, use leader dots for all input fields, and use ===> for all command areas. For example:

  ```
  EMPLOYEE NUMBER . :  015723
  ADDRESS . . . . . .  6510 Main Street
  CITY, STATE . . . .  Imperial, PA

  Command  ===>
  ```

  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.
- It is recommended that DTL be used to design CUA-based panels. The conversion process can be stopped at the ISPF panel definition source level in order to add any additional processing.

Example of a CUA Panel Definition

The panel definition shown in Figure 39 illustrates many of the panel sections and panel-element attributes that are available to support CUA panel definitions.

```
)PANEL KEYLIST(ISPSAB,ISP)
)ATTR  FORMAT(MIX)
 ! TYPE(AB)
 @ TYPE(ABSL)
 # TYPE(PT)
 $ TYPE(CH)
 < TYPE(FP)
 ¬ TYPE(NT)
 _ TYPE(NEF) PADC(_)
 ? TYPE(NEF) PADC(_) CAPS(ON)
 | TYPE(LEF) PADC(_)
 % TYPE(LI)
 ¨ TYPE(LI) CAPS(ON)
)ABC
  DESC('Options')
 PDC DESC('Create ')
 PDC DESC('Change ')
 PDC DESC('Delete ')
 PDC DESC('Browse ')
 PDC DESC('Exit Keylist Utility ')
)ABCINIT
   .ZVARS=ZPDC
   &ZPDC=' '
   IF (&COPTIONS=CREATE)
      &ZPDC=1
   IF (&COPTIONS=CHANGE)
      &ZPDC=2
   IF (&COPTIONS=DELETE)
      &ZPDC=3
   IF (&COPTIONS=BROWSE)
      &ZPDC=4
   IF (&COPTIONS=EXIT)
      &ZPDC=5
)ABCPROC
   VER (&ZPDC,LIST,1,2,3,4,5)
   IF (&ZPDC=1)
      &COPTIONS=CREATE
   IF (&ZPDC=2)
      &COPTIONS=CHANGE
   IF (&ZPDC=3)
      &COPTIONS=DELETE
   IF (&ZPDC=4)
      &COPTIONS=BROWSE
   IF (&ZPDC=5)
      &COPTIONS=EXIT
```

Figure 39. CUA Panel Definition (Part 1 of 2)

```
)ABC
  DESC('Change Keylists')
 PDC DESC('Current panel keylist ')
 PDC DESC('Current dialog keylist ')
 PDC DESC('Specify keylist ')
)ABCINIT
   .ZVARS=ZPDC
   &ZPDC=' '
   IF (&CCHANGE=PANEL)
      &ZPDC=1
   IF (&CCHANGE=DIALOG)
      &ZPDC=2
   IF (&CCHANGE=ANY)
      &ZPDC=3
)ABCPROC
   VER (&ZPDC,LIST,1,2,3)
   IF (&ZPDC=1)
      &CCHANGE=PANEL
   IF (&ZPDC=2)
      &CCHANGE=DIALOG
   IF (&ZPDC=3)
      &CCHANGE=ANY
)BODY WINDOW(62,22) CMD(ZCMD)
ˆ! Options! Change Keylistsˆ
@-------------------------------------------------------------
#                Keylist Utility for &kluappl
ˆCommand ===>_Z
ˆ
<Enter keylist name?Z       ˆ<OR   ˆ
ˆ
ˆSelect one keylist name from the list below:                ˆ
$Select  Keylist   T  -
)MODEL
|Zˆ     ˜Z       ˆ%Zˆ%Z
)INIT
.ZVARS = '( ZCMD KEYLISTN S KLUKLNFT SOURCET CURKEYL)'
.HELP = ISP05800
&ZCMD = ' '
.ATTR(S)='JUST(LEFT) '
.ATTR(KLUKLNFT)='JUST(LEFT) '
.ATTR(SOURCET)='JUST(LEFT) '
.ATTR(CURKEYL)='JUST(LEFT) '
.CURSOR = 'KEYLISTN'
)PROC
VER (&KEYLISTN NAME)
)HELP
FIELD(ZABC01) PANEL(ISPKH2)
FIELD(ZPDC0101) PANEL(ISPKH2A)
FIELD(ZPDC0102) PANEL(ISPKH2B)
FIELD(ZPDC0103) PANEL(ISPKH2C)
FIELD(ZPDC0104) PANEL(ISPKH2D)
FIELD(ZABC02) PANEL(ISPKH3)
FIELD(ZPDC0201) PANEL(ISPKH3A)
FIELD(ZPDC0202) PANEL(ISPKH3B)
FIELD(ZPDC0203) PANEL(ISPKH3C)
FIELD(KEYLISTN) PANEL(ISPKH1)
)END                                  ˆ
```

Figure 39. CUA Panel Definition (Part 2 of 2)

This panel definition will display the keylist utility panel, SAMPAN, shown in Figure 40 on page 111.

```
                          ISPF Settings
-    Functions  Change Keylists                                  ----------
     ------------------------------------------------------------
     SAMPAN          Keylist Utility for ISP     Row 1 to 10 of 16  ore:     +
S
     Enter keylist name ________  OR

     Select one keylist name from the list below:                  __
     Select  Keylist   T  -
     _       ISPHELP   P  ________________________________
     _       ISPHLP2   P  ________________________________
     _       ISPKYLST  P  ________________________________
     _       ISPNAB    P  ________________________________
     _       ISPSAB    P  *** Currently active keylist ***
T    _       ISPSNAB   P  ________________________________
     _       ISPTEST   P  ________________________________
     _       ISRHELP   P  ________________________________
     _       ISRNAB    P  ________________________________
     _       ISRNSAB   P  ________________________________

     Command ===> ____________________________________________
C     F1=Help        F2=Split       F3=Exit        F7=Backward   __________
      F8=Forward     F9=Swap       F10=Actions    F12=Cancel     =Swap
F
```

Figure 40. Sample CUA Panel (SAMPAN on ISPKLUP)

Factors That Affect a Panel's Size

The total number of lines allowed in a panel definition depends upon the storage size available. Panel definitions can be 80–160 characters wide. However, the width cannot be greater than that of the physical screen of the terminal used for the display. The WIDTH keyword in the panel definition determines the width of a display. If you are defining a panel to be displayed in a pop-up window, use the WINDOW keyword on the)BODY statement.

Two shared pool system variables, ZSCRMAXD and ZSCRMAXW, contain physical terminal screen depth and width. These variables cannot be modified. When using terminals for which an alternate size is available, these variables reflect the configuration that produces the largest screen buffer.

For example, in the case of a 3278-5 (or 3290 set up as a 3278-5), the available screen sizes are 24 x 80 and 27 x 132. Therefore, the values in ZSCRMAXD and ZSCRMAXW are 27 and 132, respectively. For the 3290, these variables contain the sizes of the hardware partition in which ISPF is operating.

When running in GUI mode, if the panel exceeds the width and/or depth of the physical display, scroll bars are automatically added to allow viewing of the hidden portion of the screen.

Vertically Scrollable Panels

You can also define more information than can fit on the panel display by defining an AREA(SCRL) attribute in the panel attribute section and by defining a panel)AREA section. You can scroll each area to see and interact with the total content defined for the area. See "Defining the Area Section" on page 169 for further discussion of the)AREA section and scrollable panel areas.

Syntax Rules and Restrictions for Panel Definition

The following apply to panel definitions:

- All statements, variable names, keywords, and keyword values can be entered in either uppercase or lowercase. ISPF translates variable names within the panel body or within panel statements to uppercase before processing them. Values assigned to dialog variables in the panel body or in the executable sections are stored as entered, in uppercase or lowercase. When symbolic substitution using a double ampersand is attempted, the variable will not be updated because ISPF makes only one pass when scanning for variable replacement.
- The command field cannot be longer than 255 characters. This is the first input field on the panel, unless otherwise specified by using the CMD keyword on the)BODY statement.

  Fields other than the command field can exceed 255 characters.

  Fields are ended by the attribute character of a following field or by the end of the panel body. A panel with a large number of variables can cause the literal table to exceed 32K bytes. ISPF issues a message when this occurs. To proceed, the panel containing the variables must be divided into two or more panels.
- All header statements, such as)ATTR and)BODY, must be coded starting in column 1. Statements following the header need not begin in column 1.
- At least one attribute must be defined within the panel)BODY section. If the entire)BODY section is defined as an AREA, (DYNAMIC, SCRL, ...), then that AREA variable must contain at least one attribute. For example, if the panel)BODY is defined as a **char AREA(DYNAMIC)**, then there must be at least one attribute variable defined within the Dynamic Area variable **char**.
- If a section is omitted, the corresponding header statement is also omitted. The)BODY header can be omitted if all previous sections are 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. ISPF ignores any data that appears on lines following the)END statement.

Using 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 can be continued on succeeding lines.
- In the action bar choice, initialization, reinitialization, processing, and help sections, several statements can occur on the same line, separated by one or more blanks. Statements cannot be split between lines, except that listed items within parentheses and long strings within quotes can be continued on succeeding lines (see “Formatting Items in Lists” on page 113).
- One or more blanks can occur on either side of operators such as an equal sign (=), a not-equal operator (¬=), greater-than symbol (>), and not-greater-than operator (¬>). Embedded blanks cannot occur in double-character operators such as the not-equal operator.

  For example: ¬ = is invalid.
- One or more blanks can occur on either side of parentheses, except that a blank cannot 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 can be coded in the action bar choice, attribute, initialization, reinitialization, processing, ccsid, panel, point-and-shoot, list, and help sections. Comments must be enclosed with the comment delimiters, /* and */. The comment must be the last item on the line. Additional keywords or statements that follow the comment on the same line are ignored. A comment cannot be continued on the next line. For multi-line comments, the comment delimiters must be used on each line.
- Blank lines can occur anywhere within the action bar choice, attribute, initialization, reinitialization, processing, and help sections.

Formatting Items in Lists

The following rules apply to items in lists:

- Listed items within parentheses can be separated by commas or one or more blanks. This rule also applies to 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 can 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 can be split between lines by coding a plus sign (+) as the last character on each line that is to be continued. That is, the plus sign is used as a continuation character. For example:

  ```
  TRANS (&CASE 1,' THIS IS THE VALUE   +
        FOR CASE 1'  2,'THIS IS THE   +
        VALUE FOR CASE 2')
  ```

Using Variables and Literal Expressions in Text Fields

The following rules apply to literals and variables in text fields:

- A literal is a character string not beginning with an ampersand or period. A literal value can be enclosed in single quotes (''). It must be enclosed in single quotes if it begins with a single ampersand or a period, or if it contains any of the following special characters:

  ```
  Blank < ( + | ) ; ¬ – , > : =
  ```

  A literal can 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, with trailing blanks stripped, before the statement is processed. Trailing blanks are stripped from the variable before the replacement

is done. A double ampersand can be used to specify a literal character string starting with, or containing, an ampersand.

In the DBCS environment, a mixed EBCDIC/DBCS literal can be specified as follows:

```
eeee[DBDBDBDB]eeeeee[DBDBDBDBDBDB]
```

In this example, *e* represents an EBCDIC character and *DB* represents a double-byte character. The brackets [and] represent shift-out and shift-in characters, in which DBCS subfields must be enclosed. These appear as blanks when displayed.

If a mixed literal contains two DBCS subfields, and

– the last character of the first subfield is a shift-in that terminates a DBCS subfield, and
– the first character of the second subfield is a shift-out that begins a DBCS subfield,

the shift-in and shift-out character pair is eliminated.

- In the panel)BODY or)AREA section, a variable can appear within a text field. In the action bar choice, initialization, reinitialization, processing, and help sections, a variable can 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. Trailing blanks are stripped from the variable before the replacement is done. 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 or by a line-end is interpreted as a literal ampersand character, not the beginning of a substitutable variable. An ampersand followed by a nonblank is interpreted as the beginning of a substitutable variable.
- A double ampersand can be used to produce a character string starting with, or containing, an ampersand. The double-character rule also applies to single quotes within literal values, if the literal is enclosed within delimiting single quotes, and to a period if it immediately follows a variable name. That is:

  ```
   &&  yields  &
   ''  yields  ' within delimiting single quotes
   ..  yields  . immediately following a variable name
  ```

  Note: To add another layer of quotes, you must double the number of quotes used in the previous layer. For example:

  ```
   'one o''ne' yields one o'ne
   'two t''''wo' yields two t''wo
  ```

- 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 right-most character in the field is replicated (shifted in), provided it is a special (non-alphanumeric) character. For example:

  ```
  %DATA SET NAME: &DSNAME ----------------------%
  ```

Assuming that the value of variable DSNAME is greater than 7 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 fewer than 7 characters, additional dashes are *pulled* in from the right. Fields defined in a scrollable area end at the end of the line where their definition starts. They will not wrap to the next line.

Validating DBCS Strings

ISPF validates DBCS data as follows:

- All DBCS output values are checked to determine whether they contain valid 16-bit DBCS codes. If an invalid code is found, it is replaced with the hexadecimal value 4195.
- The lengths of DBCS subfields in FORMAT(MIX) fields, and all FORMAT(DBCS) fields, are checked for an even number of bytes. If an exception occurs, the data is displayed in EBCDIC format.
- Split-screen or a floating command line can result in a DBCS field or subfield being divided. If this occurs in the middle of a DBCS character, the remainder of the byte is displayed as a blank and is protected.
- If the division of a DBCS subfield results in no divided DBCS characters, but the shift-in character is separated, the subfield is displayed as a DBCS field and is protected. However, if a divided DBCS character results, the remainder of the byte is displayed as a blank and is protected, and the remainder of the subfield is displayed as a DBCS field and is protected.
- If a DBCS field split results in the division of a DBCS character, the remainder of the byte is displayed as a blank and is protected.

In all of the previous cases, no message is issued to the user.

Special Requirements for Defining Certain Panels

Special requirements exist for defining the following types of panels:

- Menus
- Help tutorials. Refer to “Chapter 7. ISPF Help and Tutorial Panels” on page 281
- Table displays
- Panels containing dynamic areas
- Panels containing a graphic area.

Defining Menus

A menu, also called a selection panel (Figure 41 on page 116), is a special type of panel.

```
                    ISPF Master Application Menu

    1 Sample 1      Sample application 1                      Userid . : USERID
    2 .             (Description for option 2                 Time . . : 13:41
    3 .             (Description for option 3)                Terminal : 3278
    4               (Description for option 4)                PF keys  : 12
    5 .             (Description for option 5)                Screen . : 1
    X Exit          Terminate ISPF using list/log defaults    Language : ENGLISH
                                                              Appl ID  : ISR
                                                              Release  : ISPF

  Enter END command to terminate application

  5655-042 (C) Copyright IBM CORP 1982, 1996


   Licensed Materials - Property of IBM 5645-001, 5655-042 (C) Copyright IBM
   Corp. 1980, 1996. All rights reserved. US Government Users Restricted Rights
   - Use, duplication or disclosure restricted by GSA ADP Schedule Contract
   with IBM Corp.

  Option  ===> _____________________________________________________________
   F1=Help      F2=Split     F3=Exit      F9=Swap     F10=Actions  F12=Cancel
```

Figure 41. Example of a Menu (ISP@MSTR)

The sections that can be used in a menu definition are the same as those that can be used in other panel definitions. However, a menu requires a processing section in addition to the body section. The processing section must be in a special format.

Menu definitions are processed by the SELECT service. A menu must have an input field to allow users to enter selection options. Generally, this is the command field, and is the first input field on the panel. This field should be named ZCMD to be consistent with the field name used in this manual.

Besides ZCMD, a menu can have input fields to set up dialog variables needed by that 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, can also be displayed on a menu to provide information to users.

The required processing section must provide for the variable ZCMD to be 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 the System Productivity Facility (SPF). Use of ZSEL is recommended.

The processing section of a menu is in the following general format:

```
)PROC
  &ZSEL = TRANS( TRUNC(&ZCMD,'.')
           value, 'string'
           value, 'string'
                    .
                    .
           value, 'string'
                  ' ', ' '
                    *, '?'      )
```

The maximum length for ZSEL is 80 characters. If ZSEL is assigned a string longer than 80 characters, the string is truncated.

The ZCMD variable is truncated prior to translation to allow users to bypass one or more intermediate menus. For example, `1.2` means primary option 1, suboption 2. This is generally called a *nested option.* ZCMD is automatically stored, untranslated, as entered. When the SELECT service discovers that variable ZCMD 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 nonblank, the lower-level menu is processed in the normal fashion but is not displayed to the user.

Each **value** is one of the options that can be entered on the menu. Each *string* contains selection keywords indicating the action to occur. The selection keywords are:

```
'PANEL(pnl-name) [NEWAPPL [(appl-id)]
                 [PASSLIB]]|[NEWPOOL] [ADDPOP] [SUSPEND] [SCRNAME]'
'CMD(command)    [NEWAPPL [(appl-id)] [PASSLIB]]|[NEWPOOL] [SUSPEND]
                 [NOCHECK] [LANG(APL|CREX)]
                 [MODE(LINE|FSCR)]
                 [BARRIER]
                 [NEST]
                 [SCRNAME]'

'PGM(prog-name)  [PARM(parameters)]
                 [NEWAPPL [(appl-id)] [PASSLIB]]|[NEWPOOL] [SUSPEND]
                 [NOCHECK]
                 [MODE(LINE|FSCR)]
                 [SCRNAME]'
'WSCMD(workstation-command)
                [MODAL|MODELESS]
                [WSDIR(DIR)]
                [MAX|MIN]
                [VIS|INVIS]'

'WSCMDV(var_name)
                [MODAL|MODELESS]
                [WSDIR(DIR)]
                [MAX|MIN]
                [VIS|INVIS]'
 EXIT
```

Except for EXIT, each string of keywords must be enclosed in single quotes because it contains parentheses, and sometimes blanks.

The following selection keywords are the same as those that can be specified for the SELECT service.

`PANEL(panel-name)`

CMD(command) [LANG(APL|CREX)] [MODE(LINE|FSCR)] [BARRIER] [NEST]

PGM(program-name) [MODE(LINE|FSCR)] PARM(parameters)

[NEWAPPL[(application-id)] [PASSLIB]]|[NEWPOOL] [SUSPEND] [SCRNAME(screen_name)]

WSCMD(workstation-command) [MODAL|MODELESS] [WSDIR(DIR)] [MAX|MIN] [VIS|INVIS]

WSCMDV(var_name) [MODAL|MODELESS] [WSDIR(DIR)] [MAX|MIN] [VIS|INVIS]

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. NOCHECK, MODE, and EXIT are described following.

NOCHECK Keyword

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 can enter `1.x` as a 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 usually 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 ISPF.

MODE Keyword

You can use the MODE keyword, with either the LINE or the FSCR option, on a SELECT service request to control whether ISPF enters line mode or full-screen mode when a TSO command or dialog program is invoked. This eliminates the need to control line mode by prefixing TSO commands with a percent sign.

EXIT Keyword

The EXIT keyword, if used, applies only to a primary option menu. It terminates ISPF, using defaults for list/log data set processing. EXIT need not be enclosed in single quotes.

Blank or Invalid Options (‘’ or *,‘?’)

If you use a blank ‘ ’ for the value (ZCMD variable is blank), use a blank as the 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 you use an asterisk (*) for the value, indicating an invalid option was entered, use a question mark (?) as the string. This causes the SELECT service to redisplay the menu with an *invalid option* message.

Defining 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.

The first menu displayed when ISPF is invoked is usually treated as a primary option menu. For example, if ISPF is invoked with:

```
ISPSTART PANEL(XYZTOP)
```

panel XYZTOP is treated as a primary option menu.

Similarly, if ISPF is invoked with:

```
ISPSTART CMD(XYZ) or
ISPSTART PGM(XYZ)
```

and dialog XYZ subsequently issues:

```
SELECT PANEL(XYZTOP)
```

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 approach is not recommended because the RETURN command then causes an immediate exit from the dialog, which can be confusing to the user.

A dialog can have lower-level (nested) primary option menus. This technique is implemented 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 user. It is recommended that there be only one primary option menu per **major** application.

Specifying the Next Menu to Display

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 can be overridden. A developer can 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 can 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 can 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, the one originally invoked from a menu, completes execution.

Notes:

1. A value can be stored in ZPARENT in a function, or it can be stored in the)INIT,)REINIT,)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 ISPF each time it is used. If the dialog does not continue to set ZPARENT, ISPF 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. For example, 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 and is ignored when operating in explicit chain mode.

Example of a Master Application Menu

A master application menu, named ISP@MSTR (See Figure 41 on page 116), is distributed with ISPF as part of the panel library. This menu can be used, if desired, to allow selection of the various applications available at an installation.

If used, the master menu should be the first menu displayed when the user logs on. It can be displayed automatically by including the following command in the user's TSO LOGON procedure:

```
ISPSTART [PANEL(ISP@MSTR)]
```

When no keywords are specified on the ISPSTART command, PANEL (ISP@MSTR) is assumed.

The master application menu is generated from a DTL source file (Figure 43 on page 123). The menu selections are enabled for point-and-shoot selection.

The master application menu)INIT,)PROC, and)PNTS sections are included in Figure 42 on page 122 to illustrate some of the special menu statement formats discussed above.

```
)INIT
·ZVARS = '(ZCMD ZUSER ZTIME ZTERM ZKEYS ZSCREEN ZLANG ZAPPLID ZENVIR)'
·HELP = ISP00005
&ZPRIM = YES
&ZPRIM   = YES          /* This is a primary option menu         */
IF (&ZLOGO = 'YES')        /*                                @L5A*/
  IF (&ZSPLIT = 'NO')      /* Not in split screen            @L5A*/
    IF (&ZCMD = &Z)        /* No command pending             @L5A*/
      IF (&ZLOGOPAN ^= 'DONE') /* No logo displayed yet      @L5A*/
        .MSG = ISPLO999    /* Set logo information           @L5A*/
        .RESP = ENTER      /* Simulate enter                 @L5A*/
        &ZLOGOPAN = 'DONE' /*                                @L5A*/
        &ZCLEAN = 'NO'     /*                                @L5A*/
    IF (&ZCMD ^= &Z)       /* Command pending                @L5A*/
      &ZLOGOPAN = 'DONE'   /*                                @L5A*/
    VPUT (ZLOGOPAN) SHARED /*                                @L5A*/
  IF (&ZSPLIT = 'YES')     /* In split screen                @V5A*/
    &ZLOGOPAN = 'DONE'
)PROC
/* This in a GML based panel generated by ISPDTLC.                   */
/*                                                                   */
/* Make changes by updating the GML source file                      */
/* and reconverting ISP@MSTR.                                        */
&ZSEL = TRANS (TRUNC (&ZCMD,'.')
  1,'PANEL(ISP@PRIM) SCRNAME(PRIM)'
  X,EXIT
 ' ',' '
   *,'?' )
&ZTRAIL=;TRAIL
)PNTS
FIELD(ZPS01001) VAR(ZCMD) VAL(1)
FIELD(ZPS01002) VAR(ZCMD) VAL(2)
FIELD(ZPS01003) VAR(ZCMD) VAL(3)
FIELD(ZPS01004) VAR(ZCMD) VAL(4)
FIELD(ZPS01005) VAR(ZCMD) VAL(5)
FIELD(ZPS01006) VAR(ZCMD) VAL(X)
FIELD(ZPS00001) VAR(ZCMD) VAL(END)
)END
/* 5655-042 (C) COPYRIGHT IBM CORP 1982, 1996 */
```

Figure 42. Master Application Menu Definition

The following figure shows the DTL source for panel ISP@MSTR. All of the translatable text is defined with ENTITY tags and is placed at the beginning of the file. Special comments bordered by a DTL comment line:

```
<!-- ######################################### -->
```

identify the places where the source file can be modified and provide an explanation for including additional options.

```
<!-- ISR@MSTR selection menu -->
<!doctype dm system(
  <!ENTITY ispzmstr system -- common logic file imbed -->

<!-- Start of translatable panel text section                        -->
<!--   text delimited by " is to be translated                       -->
<!--   text should end with '">' as shown.                           -->
<!--     the '">' can be moved to the right for text expansion       -->

 <!-- panel title text follows - maximum length = 74 bytes           -->
  <!ENTITY panel_title
           "ISPF Master Application Menu">

 <!-- choice selection text entries follow                           -->
 <!-- choice text for this panel consists of 2 parts:                -->
 <!--   part 1 - point and shoot - primary description               -->
 <!--   part 2 - additional descriptive text                         -->
 <!--  if combined length of text for part 1 plus part 2 exceeds     -->
 <!--  54 bytes, the part 2 text will be folded into multiple lines  -->

 <!-- part 1 - point and shoot - primary description follows         -->
 <!-- pad short text with blanks, aligning the ending quote mark     -->
 <!--     all text strings must be the same length, including blanks -->
 <!-- ############################################################ -->
 <!-- To add options 2, 3, 4, or 5 to this panel:                    -->
 <!--   - Replace the text below for "choice_n_pnts"                 -->
 <!--        (where "n" is the option number)                        -->
 <!--     with the point-and-shoot key identifying option text.      -->
 <!--                                                                -->
 <!-- To add new options to this panel:                              -->
 <!--   - repeat the text below for "choice_n_pnts"                  -->
 <!--        (where "n" is the option number)                        -->
 <!--     for the new option number and add it to the list           -->
 <!--     with the point-and-shoot key identifying option text.      -->
 <!--      for example:                                              -->
 <!--          <!ENTITY choice_6_pnts  "New option 6">               -->
 <!-- ############################################################ -->
   <!ENTITY choice_1_pnts "Sample 1 ">
   <!ENTITY choice_2_pnts ". ">
   <!ENTITY choice_3_pnts ". ">
   <!ENTITY choice_4_pnts ". ">
   <!ENTITY choice_5_pnts ". ">
   <!ENTITY choice_X_pnts "Exit ">
```

Figure 43. Master Application Menu DTL Source (Part 1 of 4)

```
<!-- part 2 - additional descriptive text                          -->
<!-- ############################################################ -->
<!-- To add options 2, 3, 4, or 5 to this panel:                   -->
<!--   - Replace the text below for "choice_n_text"                -->
<!--        (where "n" is the option number)                       -->
<!--     with the additional option description text.              -->
<!--                                                               -->
<!-- To add new options to this panel:                             -->
<!--   - repeat the text below for "choice_n_text"                 -->
<!--        (where "n" is the option number)                       -->
<!--     for the new option number and add it to the list          -->
<!--     with the additional option description text.              -->
<!--      for example:                                             -->
<!--          <!ENTITY choice_6_text "(Description for option 6)  ">-->
<!-- ############################################################ -->
  <!ENTITY choice_1_text
      "Sample application 1        ">
  <!ENTITY choice_2_text
      "(Description for option 2)  ">
  <!ENTITY choice_3_text
      "(Description for option 3)  ">
  <!ENTITY choice_4_text
      "(Description for option 4)  ">
  <!ENTITY choice_5_text
      "(Description for option 5)  ">
  <!ENTITY choice_X_text
      "Terminate ISPF using list/log defaults">
 <!-- Status area labels          - maximum text length = 10 bytes   -->
 <!ENTITY status_userid "Userid . :">
 <!ENTITY status_time "Time . . :">
 <!ENTITY status_term "Terminal :">
 <!ENTITY status_pfkeys "Pf keys :">
 <!ENTITY status_scrnum "Screen . :">
 <!ENTITY status_lang "Language :">
 <!ENTITY status_appl "Appl ID :">
 <!ENTITY status_rel "Release :">

 <!-- Generated panel comments    - maximum text length = 66 bytes   -->
  <!ENTITY panel_cmnt1
          "This in a GML based panel generated by ISPDTLC.">
  <!ENTITY panel_cmnt2
          "                                              ">
  <!ENTITY panel_cmnt3
          "Make changes by updating the GML source file  ">
  <!ENTITY panel_cmnt4
          "and reconverting ISP@MSTR.                    ">

 <!-- panel instruction text line - maximum text length = 78 bytes   -->
 <!-- panel instruction entities will be concatenated                -->
  <!ENTITY panel_instruct_1
    "Enter <ps var=zcmd value=END csrgrp=99>END</ps> ">
  <!ENTITY panel_instruct_2
    "command to terminate application">

<!-- End of translatable panel text section                          -->
)>         <!-- DO NOT DELETE THIS LINE -->
```

Figure 43. Master Application Menu DTL Source (Part 2 of 4)

```
<varclass name=vcc type='char 80'>
<xlatl format=upper>
</xlatl>

<varclass name=vco type='char 7'>

<varlist>
  <vardcl name=zcmd varclass=vcc>
  <vardcl name=zuser varclass=vco>
  <vardcl name=ztime varclass=vco>
</varlist>
<copyr>5655-042 (C) COPYRIGHT IBM CORP 1982, 1996
<panel name=isp@mstr help=isp00005 padc=user keylist=isrnsab applid=isr
       width=80 depth=24 menu prime window=no>&panel_title;

<cmdarea noinit>
<area depth=8 extend=force width=59 dir=horiz>

  <!-- selection options follow - left side of panel                -->
  <selfld type=menu selwidth=* trail=ztrail fchoice=1 entwidth=1
          tsize=12>
    <choice> <ps var=zcmd value=1 csrgrp=99>
        &choice_1_pnts</ps>;
        &choice_1_text;
      <action run=isp@prim type=panel scrname=prim>
 <!-- ############################################################ -->
 <!-- To add options 2, 3, 4, or 5 to this panel:                  -->
 <!--     add a <ACTION> tag provide the selection                 -->
 <!--     information for the generated ZSEL statement.            -->
 <!--                                                              -->
 <!--     <action run=newoptn2 type=panel scrname=opt2>            -->
 <!--       where:                                                 -->
 <!--             run=newoptn2   - provides the name of the panel,  -->
 <!--                              pgm, cmd, wscmd, wscmdv         -->
 <!--             type=panel     - provides the selection choice:   -->
 <!--                              panel, pgm, cmd, wscmd, wscmdv   -->
 <!--             scrname=opt2   - provides an optional screen name -->
 <!-- ############################################################ -->
    <choice> <ps var=zcmd value=2 csrgrp=99>
        &choice_2_pnts;</ps>
        &choice_2_text;
    <choice> <ps var=zcmd value=3 csrgrp=99>
        &choice_3_pnts;</ps>
        &choice_3_text;
    <choice> <ps var=zcmd value=4 csrgrp=99>
        &choice_4_pnts;</ps>
        &choice_4_text;
    <choice> <ps var=zcmd value=5 csrgrp=99>
        &choice_5_pnts;</ps>
        &choice_5_text;
```

Figure 43. Master Application Menu DTL Source (Part 3 of 4)

```
 <!-- ############################################################ -->
 <!-- To add new options to this panel:                            -->
 <!--   - add a new <choice> tag to this list following the        -->
 <!--     pattern of the <choice> tags above.                      -->
 <!--     a new <ACTION> tag is required to provide the selection  -->
 <!--     information for the generated ZSEL statement.            -->
 <!--                                                              -->
 <!--     <choice> <ps var=zcmd value=6 csrgrp=99>                 -->
 <!--         &choice_6_pnts;</ps>                                 -->
 <!--         &choice_6_text;                                      -->
 <!--     <action run=newoptn6 type=panel scrname=opt6>            -->
 <!--       where:                                                 -->
 <!--             run=newoptn6   - provides the name of the panel, -->
 <!--                              pgm, cmd, wscmd, wscmdv         -->
 <!--             type=panel     - provides the selection choice:  -->
 <!--                              panel, pgm, cmd, wscmd, wscmdv  -->
 <!--             scrname=opt6   - provides an optional screen name -->
 <!-- ############################################################ -->
    <choice selchar=X> <ps var=zcmd value=X csrgrp=99>
        &choice_X_pnts;</ps>
        &choice_X_text;
      <action run=exit     type=exit>
    <comment type=proc>&panel_cmnt1;
    <comment type=proc>&panel_cmnt2;
    <comment type=proc>&panel_cmnt3;
    <comment type=proc>&panel_cmnt4;
  </selfld>
</area>
  <!-- right side of option menu panel follows, status area  -->
<area dir=horiz>
  <region dir = vert>
    <divider>
    <dtacol pmtwidth=10 entwidth=8>
      <dtafld datavar=ZUSER usage=out>  &status_userid;
      <dtafld datavar=ZTIME usage=out>  &status_time;
      <dtafld datavar=ZTERM usage=out>  &status_term;
      <dtafld datavar=ZKEYS usage=out>  &status_pfkeys;
      <dtafld datavar=ZSCREEN usage=out>&status_scrnum;
      <dtafld datavar=ZLANG  usage=out> &status_lang;
      <dtafld datavar=ZAPPLID usage=out>&status_appl;
      <dtafld datavar=ZENVIR usage=out> &status_rel;
    </dtacol>
  </region

<!-- panel logic file imbed -->
  &ispzmstr;
</area>
<region>
  <info width=78>
    <lines>
&panel_instruct_1;&panel_instruct_2;
    </lines>
<p>5655-042 (C) COPYRIGHT IBM CORP 1982, 1996
  </info>
</region>
</panel>
```

Figure 43. Master Application Menu DTL Source (Part 4 of 4)

To add a new application to the master menu, copy the ISP@MSTR DTL source file from the GML library to a private data set. Locate the sections of code within the DTL comment lines:

```
 <!-- ############################################################ -->
```

and modify the DTL source code to:

1. Define the point-and-shoot option text
2. Define the option description text
3. Add an <ACTION> tag for each additional option.

Refer to the *ISPF Dialog Tag Language Guide and Reference* for a description of the Dialog Tag Language syntax and information about compiling DTL panels.

Compile the modified DTL source file using the ISPDTLC command, and review the generated panel to confirm that your changes have been processed.

Example of a Primary Option Menu

Figure 45 on page 129 shows a primary option menu panel DTL source file definition. This is the sample primary option menu ISP@PRIM, distributed with ISPF. &ZPRIM=YES specifies that this panel is a primary option menu.

The primary option menu)INIT,)PROC, and)PNTS sections are included in Figure 44 on page 128 to illustrate some of the special menu statement formats discussed above.

The initialization section sets the control variable .HELP to the name of a tutorial page to be displayed if a 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.

The processing section specifies the action to be taken for each option entered by the user. If option 0 is selected, program ISPISM is invoked. If option 1 is selected, panel ISPUCMA is displayed; and so on.

For the tutorial, program ISPTUTOR is invoked and passed a parameter, ISP00000, which ISPTUTOR interprets as the name of the first panel to be displayed. Panel ISP00000 is the first panel in the tutorial for ISPF. Other applications should pass the name of the first tutorial page for that application.

```
)INIT
·ZVARS = '(ZCMD ZUSER ZTIME ZTERM ZKEYS ZSCREEN ZLANG ZAPPLID ZENVIR)'
·HELP = ISP00003
&ZPRIM = YES
&ZHTOP = ISP00003     /* Tutorial table of contents for this appl*/
&ZHINDEX = ISP91000   /* Tutorial index - 1st page for this appl */
VPUT (ZHTOP,ZHINDEX) PROFILE
)PROC
/* This in a GML based panel generated by ISPDTLC.                    */
/*                                                                    */
/* Make changes by updating the GML source file                       */
/* and reconverting ISP@PRIM.                                         */
&ZSEL = TRANS (TRUNC (&ZCMD,'.')
  0,'PGM(ISPISM) SCRNAME(SETTINGS)'
  1,'PANEL(ISPUCMA) SCRNAME(CMDS)'
  2,'PGM(ISPPREP) NEWAPPL SCRNAME(PREP)'
  3,'CMD(ISPDTLC) SCRNAME(DTLC)'
  7,'PGM(ISPYXDR) PARM(&ZTAPPLID) SCRNAME(DTEST) NOCHECK'
  T,'PGM(ISPTUTOR) PARM(ISP00000) SCRNAME(TUTOR)'
  X,EXIT
 ' ',' '
   *,'?' )
&ZTRAIL=TRAIL
)PNTS
FIELD(ZPS01001) VAR(ZCMD) VAL(0)
FIELD(ZPS01002) VAR(ZCMD) VAL(1)
FIELD(ZPS01003) VAR(ZCMD) VAL(2)
FIELD(ZPS01004) VAR(ZCMD) VAL(3)
FIELD(ZPS01005) VAR(ZCMD) VAL(4)
FIELD(ZPS01006) VAR(ZCMD) VAL(5)
FIELD(ZPS01007) VAR(ZCMD) VAL(7)
FIELD(ZPS01008) VAR(ZCMD) VAL(T)
FIELD(ZPS01009) VAR(ZCMD) VAL(X)
FIELD(ZPS00001) VAR(ZCMD) VAL(END)
)END
```

Figure 44. ISPF Primary Option Menu Definition

The following figure shows the DTL source for panel ISP@PRIM. All of the translatable text is defined with ENTITY tags and is placed at the beginning of the file. Special comments bordered by a DTL comment line:

```
<!-- ########################################################### -->
```

identify the places where the source file can be modified and provide an explanation for including additional options.

```
<!'-- ISR@PRIM selection menu -->

<!'doctype dm system(
  <!'ENTITY ispzprim system -- common logic file imbed -->

<!'-- Start of translatable panel text section                      -->
<!'--  text delimited by " is to be translated                       -->
<!'--  text should end with '">' as shown.                           -->
<!'--    the '">' can be moved to the right for text expansion       -->

 <!'-- panel title text follows - maximum length = 74 bytes          -->
  <!'ENTITY panel_title
           "Sample Primary Option Menu">

 <!'-- choice selection text entries follow                          -->
 <!'-- choice text for this panel consists of 2 parts:               -->
 <!'--  part 1 - point and shoot - primary description               -->
 <!'--  part 2 - additional descriptive text                         -->
 <!'-- if combined length of text for part 1 plus part 2 exceeds     -->
 <!'-- 54 bytes, the part 2 text will be folded into multiple lines  -->

 <!'-- part 1 - point and shoot - primary description follows        -->
 <!'-- pad short text with blanks, aligning the ending quote mark    -->
 <!'--    all text strings must be the same length, including blanks -->
 <!'-- ############################################################ -->
 <!'-- To add options 4, or 5 to this panel:                         -->
 <!'--  - Replace the text below for "choice_n_pnts"                 -->
 <!'--       (where "n" is the option number)                        -->
 <!'--    with the point-and-shoot key identifying option text.      -->
 <!'--                                                               -->
 <!'-- To add new options to this panel:                             -->
 <!'--  - repeat the text below for "choice_n_pnts"                  -->
 <!'--       (where "n" is the option number)                        -->
 <!'--    for the new option number and add it to the list           -->
 <!'--    with the point-and-shoot key identifying option text.      -->
 <!'--     for example:                                              -->
 <!'--         <!'ENTITY choice_8_pnts "New option 8">               -->
 <!'-- ############################################################ -->
   <!'ENTITY choice_0_pnts "Settings   ">
   <!'ENTITY choice_1_pnts "Commands   ">
   <!'ENTITY choice_2_pnts "ISPPREP    ">
   <!'ENTITY choice_3_pnts "ISPDTLC    ">
   <!'ENTITY choice_4_pnts ".          ">
   <!'ENTITY choice_5_pnts ".          ">
   <!'ENTITY choice_6_pnts ".          ">
   <!'ENTITY choice_7_pnts "Dialog Test">
   <!'ENTITY choice_T_pnts "Tutorial   ">
   <!'ENTITY choice_X_pnts "Exit       ">
```

Figure 45. Master Application Menu DTL Source (Part 1 of 6)

```
<!'-- part 2 - additional descriptive text                        -->
<!'-- ############################################################ -->
<!'-- To add options 4, or 5 to this panel:                        -->
<!'--  - Replace the text below for "choice_n_text"                -->
<!'--       (where "n" is the option number)                       -->
<!'--    with the additional option description text.              -->
<!'--                                                              -->
<!'-- To add new options to this panel:                            -->
<!'--  - repeat the text below for "choice_n_text"                 -->
<!'--       (where "n" is the option number)                       -->
<!'--    for the new option number and add it to the list          -->
<!'--    with the additional option description text.              -->
<!'--     for example:                                             -->
<!'--         <!'ENTITY choice_8_text "(Description for option 8) ">-->
<!'-- ############################################################ -->
  <!'ENTITY choice_0_text
      "Terminal and user parameters">
  <!'ENTITY choice_1_text
      "Create/change command table ">
  <!'ENTITY choice_2_text
      "Preprocessed panel utility  ">
  <!'ENTITY choice_3_text
      "ISPF DTL Conversion Utility ">
  <!'ENTITY choice_4_text
      "(Description for option 4)  ">
  <!'ENTITY choice_5_text
      "(Description for option 5)  ">
  <!'ENTITY choice_6_text
      "(Description for option 6)  ">
  <!'ENTITY choice_7_text
      "Perform dialog testing">
  <!'ENTITY choice_T_text
      "Display information about this application">
  <!'ENTITY choice_X_text
      "Terminate ISPF using list/log defaults">
```

Figure 45. Master Application Menu DTL Source (Part 2 of 6)

```
 <!'-- Status area labels         - maximum text length = 10 bytes   -->
  <!'ENTITY status_userid "Userid . :">
  <!'ENTITY status_time   "Time . . :">
  <!'ENTITY status_term   "Terminal :">
  <!'ENTITY status_pfkeys "Pf keys  :">
  <!'ENTITY status_scrnum "Screen . :">
  <!'ENTITY status_lang   "Language :">
  <!'ENTITY status_appl   "Appl ID  :">
  <!'ENTITY status_rel    "Release  :">

 <!'-- Generated panel comments   - maximum text length = 66 bytes   -->
  <!'ENTITY panel_cmnt1
          "This in a GML based panel generated by ISPDTLC.">
  <!'ENTITY panel_cmnt2
          "                                              ">
  <!'ENTITY panel_cmnt3
          "Make changes by updating the GML source file  ">
  <!'ENTITY panel_cmnt4
          "and reconverting ISP@PRIM.                    ">

 <!'-- panel instruction text line - maximum text length = 78 bytes  -->
 <!'-- panel instruction entities will be concatenated               -->
  <!'ENTITY panel_instruct_1
    "Enter <ps var=zcmd value=END csrgrp=99>END</ps> ">
  <!'ENTITY panel_instruct_2
    "command to terminate application">

<!'-- End of translatable panel text section                         -->
)>         <!'-- DO NOT DELETE THIS LINE -->
```

Figure 45. Master Application Menu DTL Source (Part 3 of 6)

```
<varclass name=vcc type='char 80'>
<xlatl format=upper>
</xlatl>

<varclass name=vco type='char 7'>

<varlist>
  <vardcl name=zcmd varclass=vcc>
  <vardcl name=zuser varclass=vco>
  <vardcl name=ztime varclass=vco>
</varlist>

<copyr>5655-042 (C) COPYRIGHT IBM CORP 1982, 1996
<panel name=isp@prim help=isp00003 padc=user keylist=isrnsab applid=isr
       width=80 depth=24 menu prime window=no>&panel_title;

<cmdarea noinit>
<area depth=11 extend=force width=59 dir=horiz>

  <!'-- selection options follow - left side of panel               -->
  <selfld type=menu selwidth=* trail=ztrail fchoice=0 entwidth=1
          tsize=12>
    <choice> <ps var=zcmd value=0 csrgrp=99>
        &choice_0_pnts;</ps>
        &choice_0_text;
      <action run=ispism type=pgm scrname=settings>
    <choice> <ps var=zcmd value=1 csrgrp=99>
        &choice_1_pnts;</ps>
        &choice_1_text;
      <action run=ispucma type=panel scrname=cmds>
    <choice> <ps var=zcmd value=2 csrgrp=99>
        &choice_2_pnts;</ps>
        &choice_2_text;
      <action run=ispprep type=pgm newappl scrname=prep>
    <choice> <ps var=zcmd value=3 csrgrp=99>
        &choice_3_pnts;</ps>
        &choice_3_text;
      <action run=ispdtlc type=cmd scrname=dtlc>
```

Figure 45. Master Application Menu DTL Source (Part 4 of 6)

```
 <!'-- ############################################################ -->
 <!'-- To add options 4, or 5 to this panel:                       -->
 <!'--    add a <ACTION> tag provide the selection                  -->
 <!'--    information for the generated ZSEL statement.             -->
 <!'--                                                              -->
 <!'--    <action run=newoptn4 type=panel scrname=opt4>             -->
 <!'--      where:run=                                              -->
 <!'--            run=newoptn4   - provides the name of the panel,  -->
 <!'--                             pgm, cmd, wscmd, wscmdv          -->
 <!'--            type=panel     - provides the selection choice:   -->
 <!'--                             panel, pgm, cmd, wscmd, wscmdv   -->
 <!'--            scrname=opt4   - provides an optional screen name -->
 <!'-- ############################################################ -->
    <choice> <ps var=zcmd value=4 csrgrp=99>
        &choice_4_pnts;</ps>
        &choice_4_text;
    <choice> <ps var=zcmd value=5 csrgrp=99>
        &choice_5_pnts;</ps>
        &choice_5_text;
    <choice hide> <ps var=zcmd value=6 csrgrp=99>
        &choice_6_pnts;</ps>
        &choice_6_text;
    <choice> <ps var=zcmd value=7 csrgrp=99>
        &choice_7_pnts;</ps>
        &choice_7_text;
      <action run=ispyxdr type=pgm parm=&amp;ZTAPPLID nocheck scrname=dtest>
 <!'-- ############################################################ -->
 <!'-- To add new options to this panel:                            -->
 <!'--  - add a new <choice> tag to this list following the         -->
 <!'--    pattern of the <choice> tags above.                       -->
 <!'--    a new <ACTION> tag is required to provide the selection   -->
 <!'--    information for the generated ZSEL statement.             -->
 <!'--                                                              -->
 <!'--    <choice> <ps var=zcmd value=8 csrgrp=99>                  -->
 <!'--        &choice_8_pnts;</ps>                                  -->
 <!'--        &choice_8_text;                                       -->
 <!'--    <action run=newoptn8 type=panel scrname=opt8>             -->
 <!'--      where:run=                                              -->
 <!'--            run=newoptn8   - provides the name of the panel,  -->
 <!'--                             pgm, cmd, wscmd, wscmdv          -->
 <!'--            type=panel     - provides the selection choice:   -->
 <!'--                             panel, pgm, cmd, wscmd, wscmdv   -->
 <!'--            scrname=opt8   - provides an optional screen name -->
 <!'-- ############################################################ -->
    <choice selchar=T> <ps var=zcmd value=T csrgrp=99>
        &choice_T_pnts;</ps>
        &choice_T_text;
      <action run=isptutor type=pgm parm=ISP00000 scrname=tutor>
    <choice selchar=X> <ps var=zcmd value=X csrgrp=99>
        &choice_X_pnts;</ps>
        &choice_X_text;
      <action run=exit     type=exit>
    <comment type=proc>&panel_cmnt1;
    <comment type=proc>&panel_cmnt2;
    <comment type=proc>&panel_cmnt3;
    <comment type=proc>&panel_cmnt4;
  </selfld>
</area>
```

Figure 45. Master Application Menu DTL Source (Part 5 of 6)

```
  <!'-- right side of option menu panel follows, status area -->
<area dir=horiz>
  <region dir = vert>
    <divider>
    <dtacol pmtwidth=10 entwidth=8>
      <dtafld datavar=ZUSER usage=out>  &status_userid;
      <dtafld datavar=ZTIME usage=out>  &status_time;
      <dtafld datavar=ZTERM usage=out>  &status_term;
      <dtafld datavar=ZKEYS usage=out>  &status_pfkeys;
      <dtafld datavar=ZSCREEN usage=out>&status_scrnum;
      <dtafld datavar=ZLANG  usage=out> &status_lang;
      <dtafld datavar=ZAPPLID usage=out>&status_appl;
      <dtafld datavar=ZENVIR usage=out> &status_rel;
    </dtacol>
  </region

<!'-- panel logic file imbed -->
  &ispzprim;
</area>
<region>
  <info width=78>
    <lines>
&panel_instruct_1;&panel_instruct_2;
    </lines>
<p>5655-042 (C) COPYRIGHT IBM CORP 1982, 1996
  </info>
</region>
</panel>
```

Figure 45. Master Application Menu DTL Source (Part 6 of 6)

To add a new application to the primary option menu, copy the ISP@PRIM DTL source file from the GML library to a private data set. Locate the sections of code within the DTL comment lines:

```
<!-- ############################################################ -->
```

and modify the DTL source code to:

1. Define the point-and-shoot option text
2. Define the option description text
3. Add an <ACTION> tag for each additional option.

Refer to the *ISPF Dialog Tag Language Guide and Reference* for a description of the Dialog Tag Language syntax and information about compiling DTL panels.

Compile the modified DTL source file using the ISPDTLC command, and review the generated panel to confirm that your changes have been processed.

The required input field, ZCMD, appears in the second line of the panel body. It is followed by a description of the various options.

This menu also has eight variables within text fields at the upper-right corner of the screen. These reference system variables from the shared variable pool that display user ID, time, terminal type, number of function keys, screen number, language, application ID, and ISPF release number.

Defining Table Display Panels

A table display panel is a special panel that is processed by the TBDISPL service. When it is displayed, it has a fixed (nonscrollable) portion followed by a scrollable

table portion. The fixed portion is defined by the)BODY section in the panel definition. The scrollable portion is defined by the)MODEL section.

The fixed portion contains the command field and usually the scroll amount field. It can also include other input fields as well as output fields, action bars, text, dynamic areas, scrollable areas, and a graphic area.

The scrollable portion is defined by up to eight *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 corresponds 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. The application user can 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 corresponds to multiple lines on the screen. If desired, a separator line, consisting of blanks or dashes, for example, can 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.

Each definition using the model lines on the display is known as a *model set*.

Table Display Vocabulary

This topic defines some terms related to table display. Figure 46 illustrates those terms that refer to parts of a TBDISPL display. The two main parts of a TBDISPL display are the fixed portion and the scrollable portion. The fixed portion contains the command field and commonly a scroll amount field and a top-row-displayed indicator. The scrollable portion contains the table information and usually, if the screen is not filled, a bottom-of-data marker.

```
          Command Field                Top-Row-Displayed Indicator
                |                                |
                |                                |
+--------------V--------------------------------V-------------+
|  ------------------- Population Change ------ ROW 4 OF 10 | ----*  Scroll
|  Command ==>                                Scroll ==>PAGE | <------ Amount
|                                                            |      |  Field
|  This table shows selected metropolitan areas which had a  |      |
|  large relative increase in population from 1970 to 1980.  |      |  Fixed
|                                                            |      |  Portion
|  Metro area        State     Change                        |      |
|                              (Percent)                     | ----*
|  Fort Collins       co        +66.0                        | ----*
|  West Palm Beach    fl        +64.3                        |      |  Scrollable
|  Fort Lauderdale    fl        +63.6                        |      |  Portion
|  Bryan              tx        +61.5                        |      |
|  Reno               nv        +60.0                        |      |
|  Provo              ut        +58.4                        |      |
|  McAllen            tx        +56.1                        |      |  Bottom-of-
|  ******************* BOTTOM OF DATA *******************---------- Data
|                                                            | ----*  Marker
+------------------------------------------------------------+
```

Figure 46. Parts of a TBDISPL display

auto-selection

The process by which the row specified in the CSRROW parameter or .CSRROW control variable is selected, even if the user did not explicitly select that row by modifying the corresponding model set displayed on the screen.

Relevant concepts include: selected row, user-selection, CSRROW parameter, .CSRROW control variable, AUTOSEL parameter, and .AUTOSEL control variable.

bottom-of-data marker

The low-intensity text that appears after the last displayed row in the last page of data in a TBDISPL display. If there are no displayed rows, this marker will be the only information displayed in the scrollable portion. The text `BOTTOM OF DATA`, with asterisks on each side, appears after the last row on a table display. The dialog can define an alternate marker by assigning text to ZTDMARK.

ISPF uses the + default attribute character for the bottom-of-data marker. The default attribute characters are %, +, and _. For a description of the default attribute characters see "Using Default Attribute Characters" on page 177. You can change the default attribute characters by using the DEFAULT keyword on either the)ATTR or)BODY head statement. For example: `DEFAULT(abc)` where a, b, and c are the 3 characters that take the place of %, +, and _, respectively. The default attribute characters are position-sensitive. Thus, if you change the default character ″b″ in the second position of the DEFAULT keyword parameter (ISPF's default character is +), it must maintain the characteristics of TYPE(TEXT), INTENS(LOW), COLOR(BLUE) for the bottom-of-data marker to display correctly.

Relevant concepts include: system variable ZTDMARK.

command field

A required field in the fixed portion of a TBDISPL display where commands are entered. The command field can be identified in the panel definition through use of the CMD parameter on the)BODY statement. If the CMD parameter is not specified, the first input field is assumed to be the command field.

Relevant concepts include: system commands, application commands, and function commands.

dynamic expansion

The process by which a table being displayed is expanded as needed if a user scrolls beyond the top or bottom of data contained in the table at the time of the scroll request.

Relevant concepts include: scrolling and TBDISPL.

fixed portion

The nonscrollable portion of a TBDISPL display. That is, the part of the display that is not affected by the UP or DOWN commands. Note that both the fixed and scrollable portions are unaffected by the LEFT and RIGHT commands. The fixed portion is defined by the)BODY section of the panel definition.

Relevant concepts include: scrollable portion,)BODY section.

model lines
The lines in the)MODEL section of a TBDISPL panel definition, which form a template, or *model*, for the scrollable portion of a TBDISPL display.

Relevant concepts include:)MODEL section, model set, scrollable portion.

model set
The lines in the scrollable portion of a TBDISPL display that correspond to a particular table row. Model sets are created by ISPF by replicating the model lines in the panel definition and then filling in the fields with variable and table row information. Each model set on the display corresponds to a table row. If there are *n* model lines, where *n* can be from 1 to 8, then each model set is made up of *n* lines on the display.

Relevant concepts include: model lines, and scrollable portion.

pending END request
The situation that exists when a user has selected more than one row and has entered the END or RETURN command. The dialog can choose to ignore the selected rows, or it can process the selected rows in a TBDISPL series. In the latter case, each call of TBDISPL results in a return code of 8. When all the selected rows have been processed, the dialog commonly honors the pending END request by not invoking the TBDISPL service again.

Relevant concepts include: TBDISPL series, pending scroll request, and pending selected row.

pending scroll request
The situation that exists when a user has selected one or more rows, and has entered the UP or DOWN command. After the dialog has processed all the selected rows, it can invoke TBDISPL without the PANEL and MSG parameters to display the table and panel and have the pending scroll request honored. A pending scroll request can also exist when a user has issued the UP or DOWN command and the dialog is dynamically building the table. After adding the rows needed to satisfy the scroll request, the dialog can invoke TBDISPL without the PANEL or MSG parameters and ISPF will honor the pending scroll request.

Relevant concepts include: TBDISPL series, pending END request, pending selected row, and controlling the top-row-displayed.

pending selected rows
Occurs when a user has selected more than one row in a single interaction. Upon return from the TBDISPL display, the CRP is positioned to the first of the selected rows. The other rows, which remain to be processed, are the *pending selected rows*.

Relevant concepts include: selected row, TBDISPL series, pending END request, pending scroll request, system variable ZTDSELS.

scroll amount field
An optional field in the fixed portion of a TBDISPL display where scroll amounts, for example, PAGE, HALF, or 10, are entered. If the input field immediately following the command field is exactly 4 characters long, it is assumed to be the scroll amount field.

Relevant concepts include: scrolling, and system variables ZSCROLLA and ZSCROLLN.

scrollable portion
The part of a TBDISPL display defined by the)MODEL section of the

panel definition and made up of model sets. It contains the ISPF table information. It is affected by the UP and DOWN commands.

Relevant concepts include: fixed portion,)MODEL section, model lines, and model sets.

select field

A field in the scrollable portion where line commands are entered. For example, a `d` entered into the select field of a model set can indicate that the corresponding table row is to be deleted. TBDISPL does not *officially* identify any field as a select field. It is up to the dialog to determine the characteristics or meaning of a select field.

Relevant concepts include: line commands, scrollable portion, model set, selected row, and user-selection.

selected row

A row in an ISPF table that has been auto-selected or user-selected.

Relevant concepts include: auto-selection, user-selection, model set, pending selected row, system variable ZTDSELS, POSITION parameter, and ROWID parameter.

TBDISPL series

A call of the TBDISPL service that results in a display where the user selects more than one row, followed by calls of the TBDISPL service without the PANEL and MSG parameters to process the pending selected rows.

Relevant concepts include: pending selected rows, pending END request, pending scroll request, and system variable ZTDSELS.

top-row-displayed indicator

There are three possible texts for the top-row-displayed indicator:

- ROW x OF y

  x is the current row pointer of the top row displayed. *y* is the total number of rows in the table.

- ROW x TO z OF y

  x is the current row pointer of the top row displayed. *z* is the row pointer of the last visible table row. *z* is calculated as the current row pointer of the top row displayed plus the number of lines displayed minus one. *y* is the total number of rows in the table.

- ROW x FROM y

  x is the row pointer of the table row that has met the criteria of the SCAN. *y* is the total number of rows in the table.

The text used for the top-row-displayed indicator is determined by the CUA mode selected and by whether ROWS is set to ALL or SCAN in the panel model section. Table 5 on page 139 is a summary of the CUA mode and ROWS(ALL) or ROWS(SCAN) combinations and the resulting top-row-displayed messages. CUA mode of YES is determined by the presence of a panel statement or by specifying CUA MODE=YES on Option 0.

Table 5. Text for Top-Row-Displayed Indicator

CUA Mode	ROWS	Top-Row-Displayed Message	Message ID
YES	ALL	ROW x TO z OF y	ISPZZ102
YES	SCAN	ROW x FROM y	ISPZZ103
NO	ALL	ROW x OF y	ISPZZ100
NO	SCAN	ROW x OF y	ISPZZ100

The message text appears right-justified on the top line of the display, or just below the action bar separator line if an action bar is defined. Your dialog can define an alternate indicator if you assign a message ID to ZTDMSG. TBDISPL invokes the GETMSG to get the short and long message text. If a short message is found, it is used as the top-row-displayed indicator; if not, the long message text is used. In either case, any variables in the messages are substituted with their current values. If ZTDMSG does not exist, the long form of message ISPZZ100, ISPZZ102, or ISPZZ103 is used.

If the model section for a table contains more than one line, it is possible that the entire model section will not fit on the screen. In this case, the last rows of the table area are left blank. A partial model section is not displayed. The only way to display a partial model section is if you request your function keys to appear over your table display, or if you split your screen over your table display.

When you specify ROWS(SCAN) in your panel model section, ISPF finds only enough rows to fill the display, thus providing a performance boost. Therefore, you cannot know the entire number of table rows that meet your search criteria without scrolling through the complete table.

When a table is being built dynamically to satisfy scroll requests, you can make the top-row-displayed indicator reflect the positioning in the logical table instead of the physical table. See the description of ZTDLTOP and ZTDLROWS in *ISPF Services Guide*

Relevant concepts include: system variables ZTDMSG, ZTDTOP, ZTDLTOP, ZTDROWS, and ZTDLROWS; messages ISPZZ100, ISPZZ101, ISPZZ102, and ISPZZ103; and controlling the top-row-displayed.

user-selection

The process by which ISPF table rows are chosen or *selected* for processing by the user modifying the corresponding model sets on the display. A user *modifies* a model set by entering data into that model set. Overtyping a model set with the same data does not cause the row to be selected.

Relevant concepts include: auto-selection, selected row, model set, and system variable ZTDSELS.

Requirements for Attribute Section

Attribute characters can be defined for use in the panel body and the model lines. In the)BODY section, any attribute except EXTEND(ON) and SCROLL(ON) can be associated with any field or area. In the)MODEL section, any attribute except

those associated with dynamic and graphic areas can be used with any field. That is, the attributes AREA, EXTEND, SCROLL, USERMOD, and DATAMOD are not allowed in model lines.

Input and output fields default to CAPS(ON) and JUST(LEFT), in the)BODY section, but they default to CAPS(OFF) and JUST(ASIS) in the)MODEL section.

An attribute section is required if the model line contains output fields. There is no default attribute character for output fields.

Requirements for Body Section

The panel body section is required. It contains the nonscrollable data, which is the command field and, commonly, the scroll amount field. The rules for their definition are:

Command field (required)

This field must not be longer than 255 characters.

The command field can have any desired name. The position of the command field can be specified through use of the CMD parameter on the)BODY statement. If the CMD parameter is not specified, the first input field is assumed to be the command field.

The command field is used, 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 in the corresponding dialog variable. Upon return from TBDISPL, the dialog function can interpret this field and take appropriate action. The ZCMD field is cleared each time a TBDISPL request is received with the MSG or PANEL parameter. If the TBDISPL request contains a table name and no other parameters, the ZCMD field contains what was entered on the previous TBDISPL.

The ISPF commands are system commands, while the application-defined commands are application commands. The commands processed by the dialog function are function commands.

Scroll amount field (optional)

If the input field immediately following the command field is exactly 4 characters long, it is assumed to be the scroll amount field.

The field can have any desired name. Its initial value can be set in the)INIT section of the panel definition to any valid scroll amount.

If no scroll amount field is specified, the system variable ZSCROLLD, which can be set by a dialog, is used to determine the default scroll amount. If there is no scroll amount field and ZSCROLLD has not been set, PAGE is assumed.

When a user enters a scroll request, variables ZSCROLLA and ZSCROLLN are set. ZSCROLLA contains the value of the scroll amount field (MAX, CSR, for example). ZSCROLLN contains the number of lines or columns to scroll, computed from the value in the scroll amount field. For example, if a dialog is in split-screen mode and if 12 lines are currently visible and a user requests DOWN HALF, ZSCROLLN contains a ’6’. The system variable ZVERB contains the scroll direction, DOWN in this case. If ZSCROLLA has a value of MAX, the value of ZSCROLLN is not meaningful.

The following can appear in the)BODY section:
- Action bars
- Text
- Variables within text; for example, `&XYZ`
- Input fields
- Output fields
- Dynamic areas
- Scrollable areas
- Graphic areas.

Notes:

1. Only one extendable area is allowed on a panel. This includes dynamic, scrollable, and graphic areas.
2. Graphic areas are not supported when you are running in GUI mode. When a GRINIT statement is encountered, you will receive a message that panels with graphics will not be displayed. You may choose to continue. When a panel with graphics is encountered, you will receive an error message that the panel cannot be displayed.

   If you are running in split-screen mode, the graphic area panel cannot be displayed on the host session.

   If you specified GUISCRD or GUISCRW values on the ISPSTRT invocation which are different from the actual host screen size, GDDM cannot be initialized and the GRINIT service will end with a return code of 20.

Requirements for Model Section

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 can 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 to blank before each row in the table is read. Thus, if the variable is an *extension* variable in the table, which cannot exist in all rows, previous values are erased and, thereby, are 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 certain 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 maximum of eight model lines is allowed. Any attribute except those associated with dynamic, graphic, or scrollable areas (AREA, EXTEND, SCROLL, USERMOD, and DATAMOD) can be used with any fields in the model lines. The following can appear in the)MODEL section:
- Text
- Variable model lines (see below)
- Input fields

- Output fields.

The following cannot appear in the)MODEL section:
- Action bars
- Variables within text
- Dynamic areas
- Graphic areas
- Scrollable areas.

Typically, the first field within the model lines specifies the dialog variable into which a selection code, entered by a user, will be stored. 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 can be specified in the model line. A text attribute character can 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.

Variable model lines can be specified in the panel definition. If a variable, a name preceded by an ampersand, begins in column 1 of any model line, the value of that variable defines the model line.

The following rules apply to variable model lines:
- The variable must be the only information on the model line. If any other data is present, an error results.
- If the value of the variable is greater than the screen width, an error results.
- The variable can contain any character string that is a valid panel definition model line, except that the variable cannot define a variable model line. A variable whose value is all blanks is acceptable.
- If the variable contains the character string `OMIT` starting in column 1, that variable model line will not be used in the model definition.
- All model line variables must be initialized before the table display service is called with a nonblank panel name. Changes to the variables that occur within the panel or the dialog function are not honored until table display is called again with a nonblank panel name.
- If variable model lines are being used, the panel is retrieved from disk every time that table display is called with a nonblank panel name and the value of the variable model line has changed.

Requirements for Initialization Section

An initialization section, if present, is processed when the TBDISPL service is invoked with the panel name specified.

If Z variables occur as name placeholders within the model lines or the fixed portion, 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. 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 body or model sections. For example, if one Z variable occurs as a placeholder within the panel body and two Z variables occur as placeholders within the model lines, then NAME1 corresponds to the field in the body and NAME2 and NAME3 correspond to the two fields in the model lines. For compatibility with SPF, Z variables in the model lines of a table display panel can be assigned to the VARS variable, rather than to the control variable, .ZVARS. For example:

```
)INIT
  &VARS  = '(NAME2,NAME3)'
```

It is recommended, however, that table display panels use the .ZVARS control variable. It *must* be used if the CLEAR keyword is added to the)MODEL heder statement, explicit cursor placement is used for table display, or Z variable placeholders are in the)BODY section.

The)INIT section of a TBDISPL panel definition can contain any statement that is valid in an)INIT section of a DISPLAY panel definition.

Requirements for Reinitialization Section

If a)REINIT section is included, it is executed when TBDISPL is reinvoked without a panel name or when a redisplay occurs automatically because of the .MSG control variable being nonblank.

The)REINIT section of a TBDISPL panel definition can contain any statement that is valid in a)REINIT section of a DISPLAY panel definition.

Any control variable except .ZVARS can be set within the)REINIT section. If table variables that are in the model lines are referenced within the)REINIT section, then the values for the current row, as specified by the CRP, are used. For example, if the .ATTR control variable is set for fields that are in the)MODEL section, then only fields in the model set on the display that corresponds to the current selected row will have their attributes changed.

Requirements for Processing Section

If a)PROC section is included, it is executed before control returns to the dialog function. It is not executed while the user is scrolling.

The)PROC section of a TBDISPL panel definition can contain any statement that is valid in a)PROC section of a DISPLAY panel definition.

Any control variable except .AUTOSEL and .ZVARS can be used in the)PROC section. If table variables that are in the model lines are referenced within the)PROC section, then the values for the current row, as specified by the CRP, are used. For example, if the .ATTR control variable is set for fields that are in the)MODEL section, only fields in the model set on the display that corresponds to the current selected row will have their attributes changed.

The)PROC section can check the value of ZTDSELS to determine if any rows were selected. This value and its interpretation are:

0000 No selected rows

0001 One selected row (now the current row)

0002 Two selected rows, consisting of the current row and a pending selected row

0003 Three selected rows, consisting of the current row and two pending selected rows

... And so forth.

Using Control Variables

Two control variables, .AUTOSEL and .CSRROW, can be used in the executable—)INIT,)REINIT, and)PROC—sections of a TBDISPL panel definition. They are ignored in a DISPLAY panel definition.

The .AUTOSEL and .CSRROW control variables can be used to control the selection (and preselection) of a row in a table display. For more information on these variables, see “.AUTOSEL” on page 272 and “.CSRROW” on page 273.

Processing Panels by Using the TBDISPL Service

When a panel is displayed by the TBDISPL service, the model lines in the)MODEL section are duplicated at 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 usually displayed. Two exceptions are:

- When the command line is repositioned to the bottom of the screen, the line above it, which can be a model line, may be overlaid with a blank line and used as the long message line. This prevents table display data from being overlaid with long message data.
- When the PFSHOW command is in effect, up to four additional lines can be overlaid.

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, in all rows, with the current contents of the corresponding dialog variables. If these fields are to be blank, the corresponding variables must be set to blanks or null prior to each call of TBDISPL. The CLEAR keyword can be used to specify that they are to be blanked.

A user can scroll the data up and down. Scroll commands, such as `DOWN 5`, apply to the number of table entries to scroll up or down. For example, if three model lines are specified, `DOWN 5` would scroll by 5 table entries, which corresponds to 15 lines on the display.

A user can enter information in any of the input fields within the fixed or scrollable portion of the panel.

Figure 47 on page 145 shows a sample panel definition for table display.

```
)ATTR
  @ TYPE(OUTPUT) INTENS(LOW)
)BODY
%----------------------------  EMPLOYEE LIST  ---------------------------------
%COMMAND INPUT ===>_ZCMD                                     %SCROLL ===>_AMT +
+
%EMPLOYEES IN DEPARTMENT@Z +
+
+SELECT     ------ EMPLOYEE NAME -------          -- PHONE ---         EMPLOYEE
+ CODE      LAST         FIRST        MI          AREA NUMBER           SERIAL
)MODEL
  _Z+      @LNAME       @FNAME       @I          @PHA @PHNUM            @EMPSER
)INIT
  .ZVARS = '(DEPT SELECT)'
  &AMT  = PAGE
  .HELP = PERS123
)REINIT
  IF (.MSG = ' ')
     &SELECT = ' '
     REFRESH (SELECT)
)PROC
  IF (&ZTDSELS ¬= 0000)
     VER (&SELECT, LIST, A, D, U)
)END
```

Figure 47. Table Display Panel Definition

Assuming that the current contents of the table are as shown in Table 6 and that dialog variable DEPT contains '27', the resulting display is shown in Figure 48.

Table 6. Table Display Data

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	Kelvey	Ann	A	914	555-4156

```
----------------------------  EMPLOYEE LIST  --------------------  ROW 1 OF 5
COMMAND INPUT ===> _                                   SCROLL ===> PAGE

EMPLOYEES IN DEPARTMENT 27

SELECT     ------ EMPLOYEE NAME -------       --- PHONE ---        EMPLOYEE
 CODE      LAST         FIRST        MI       AREA  NUMBER          SERIAL
           Robertson    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      A        914   294-1168         502774
****************************** BOTTOM OF DATA ******************************
```

Figure 48. 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 and placed in a variable named SELECT. The other variables in the model line correspond to variables in the table. The example also illustrates the use of two Z variables as placeholders in the body of the panel and in the model line, the initialization of the scroll amount field to PAGE, and the specification of a corresponding help panel.

The same table might be displayed by using several model lines with the panel definition shown in Figure 49.

```
)ATTR
 @ TYPE(OUTPUT)  INTENS(LOW)
 # TYPE(INPUT)  PAD('_')
)BODY
%----------------------------  EMPLOYEE LIST  ---------------------------------
%COMMAND INPUT ===>_ZCMD                                   %SCROLL ===>_AMT +
+
%EMPLOYEES IN DEPARTMENT@Z +
+
+ENTER CHANGES ON THE LINES BELOW.
+
)MODEL
  #Z   +  SERIAL: @EMPSER +              LAST NAME:  @LNAME                +
          PHONE:  @PHA@PHNUM   +         FIRST NAME: @FNAME                +
                                         INITIAL:    @I                       +
          ----------------------------------------------------------------------
)INIT
  .ZVARS = '(DEPT SELECT)'
  &AMT = PAGE
  .HELP = PERS123
)END
```

Figure 49. Table Display Panel Definition with Several Model Lines

The resulting display is shown in Figure 50. 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.

```
----------------------------  EMPLOYEE LIST  ----------------------  ROW 1 OF 5
COMMAND INPUT ===> &us.                                          SCROLL ===> PAGE

EMPLOYEES IN DEPARTMENT 27

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
      ---------------------------------------------------------------------
****************************** BOTTOM OF DATA ******************************
```

Figure 50. Table as Displayed with Several Model Lines

Formatting Panels That Contain Dynamic Areas

ISPF facilities permit the format and content of a display to be determined in the same dialog in which it is displayed. This is called dynamic formatting. See "Specifying Dynamic Areas" on page 205 for information on how to specify a dynamic area in the)ATTR section header.

Areas are reserved for this purpose in a panel definition and are called dynamic areas. A dynamic area can encompass all or part of a panel display.

The format of a dynamic area is specified by a string of control and data characters, stored in a dialog variable. This variable may have been produced either in the current dialog or, earlier, in another dialog or program. The string usually contains a mixture of nondisplayable attribute characters and data to be displayed. The name of the dialog variable is chosen by the panel designer. This name is placed in the panel definition within the dynamic area.

A dialog uses the DISPLAY, TBDISPL, or SELECT service to display a panel containing a dynamic area. After the display and after entry of any input by the user, data from within the dynamic area is stored in the variable, associated with the area, and is available for processing by the dialog function.

When a panel is displayed, the number of lines in a dynamic area can be increased automatically to accommodate the number of lines available on the terminal being used for the display.

Panel Processing Considerations

When you are defining a dynamic area and generating a dynamic character string that defines the format of the data to be placed within that area on the panel, a number of rules apply:

- The area cannot be specified by using a Z-variable *place-holder* within the panel body.
- Within the dynamic area, all nonattribute characters are treated as data to be displayed. Unlike other parts of the panel body, a variable name does not follow an attribute character.
- The dialog is responsible for insuring data integrity, validity of attribute codes, and so on, for the dynamic character string.
- If the dynamic area has a width that is less than the screen size, the panel designer must place the appropriate attribute characters around this *box* so that the data within the area is not inadvertently affected. For example, the panel designer can place fields with SKIP attributes following the right-most boundaries so that the cursor is properly placed to the next or continued input field within the area.
- If the dialog must know the dimensions of the dynamic area before the data is formatted, this information is available by invoking the PQUERY dialog service. All dialog services are described in *ISPF Services Guide*
- The scroll amount field is optional. On a panel with a scrollable area, if the input field following the command field in the panel body is exactly 4 characters long, it is assumed to be the scroll amount field. Otherwise, the system variable ZSCROLLD, which can be set by the dialog, is used to determine the default scroll amount. If there is no scroll amount field and ZSCROLLD has not been set, PAGE is assumed. ZSCROLLA contains the value of the scroll amount field, such as MAX or CSR. ZSCROLLN contains the scroll number computed from the value in the scroll amount field (number of lines or columns to scroll). For example, if a user is in split-screen mode, 12 lines are currently visible, and the user requests DOWN HALF, ZSCROLLN contains a ’6’. The system variable ZVERB contains the scroll direction, DOWN in this case. If ZSCROLLA has a value of MAX, the value of ZSCROLLN is not meaningful.
- A nonblank input or output field preceding a dynamic area must be terminated by an attribute character.

- When variable substitution occurs within a text field in the panel body, the field must be terminated by an attribute character, prior to a special character defining a dynamic area. See "Using Variables and Literal Expressions in Text Fields" on page 113 for additional information about text field variable substitution.

Although panel display processing cannot provide point-and-shoot support for dynamic areas, it does provide the PAS(ON) keyword for TYPE(DATAOUT). The PAS(ON) keyword reflects the CUA point-and-shoot color. It is up to application developers to provide the point-and- shoot function in programs they develop.

Similarly, while the panel display service does not perform the scrolling for dynamic or graphic areas, it does provide an interpretation of the user's scroll request.

The *value* for the SCROLL keyword cannot be specified as a dialog variable.

A panel cannot have more than one scrollable area or more than one extended area. The scrollable area can be a panel with a scrollable area or a table display.

These rules are applied in Figure 51.

```
)ATTR
 #   AREA(DYNAMIC)  SCROLL(ON)  EXTEND(ON)
)BODY
%-------------------- TITLE -----------------------
%COMMAND ===>_ZCMD                  +SCROLL ===>_AMT +
+
+  (Instructions for this panel ...)
+
#SAREA -------------------------------------------#
+
+  (More instructions for this panel ...)
+
```

Figure 51. Panel Definition Illustrating SCROLL and EXTEND

In this example, there are:

- 5 lines in the panel body before the extended area
- 3 more lines after the extended area.

This makes a total of 8 lines that are outside the dynamic area. Therefore, if the panel were displayed on a 3278 Model 4, which has 43 lines, the depth or extent of the dynamic area would be 43 minus 8, or 35 lines. In split-screen mode, the panel is still considered to have a 35-line scrollable area, even though part of it is not visible.

In this example, the dynamically-generated data string to be placed in the area is taken from the dialog variable SAREA. If, for example, the dynamic area is 60 characters wide and 10 lines deep, the first 60 characters of the string are placed in the first line of the area, the next 60 characters are placed in the second line of the area, and so on, until the last 60 characters are placed in the tenth line of the area. Following a user interaction, the contents of the area are stored in the same variable.

The width of the dynamic area includes the special characters that designate the vertical sides. These delimiter characters do not represent attribute characters.

A number of the capabilities described in the previous sections have implications for panel areas as well as panel fields. These include:

- A REFRESH statement can be used to reset an *area* when reinitializing or redisplaying a panel. The variable value is again read and placed in the area. Since the value also contains attribute information that may have changed, the characteristics for each field are again analyzed.
- The cursor placement capability applies to dynamic areas. That is, .CURSOR can be assigned to a dynamic area name and .CSRPOS can be assigned to a position within the dynamic area. The position within an area applies within the rectangular bounds of that area.
- The .ATTRCHAR control variable can be used to override attribute characters that are used within dynamic areas. In addition, .ATTRCHAR can be used to define a new attribute character that has not been previously listed within the panel)ATTR section. Using .ATTRCHAR as a vehicle for defining new attribute characters can be done only within the)INIT section and only for fields within dynamic areas (TYPE(DATAIN) or TYPE(DATAOUT)).
- The PQUERY service can be invoked by the dialog function to determine the characteristics of the dynamic area before the dialog function constructs the dynamic character string.

Character-Level Attribute Support for Dynamic Areas

ISPF allows you to associate character-level attributes with individual characters within a dynamic area. Each character in the dynamic area can be assigned characteristics of color and extended highlighting, which override these attribute values identified in the field attribute. You can also specify that a graphic escape (GE) order be used to display a graphic character from an alternate character set. See “Defining the Attribute Section” on page 176 for more information.

Note: Character-level color and extended highlighting will be ignored when running in GUI mode.

These attributes are treated as character attributes only if they are used in the *shadow* variable for the dynamic area (see the description of shadow variable below); otherwise, they are treated as text.

Variables may be substituted for the values of the COLOR, HILITE, and GE keywords in the same way they are substituted for field attributes.

The .ATTRCHAR control variable may be used to override the COLOR, HILITE, and GE keywords for character attributes in the same way it is used to override field attributes. The TYPE keyword cannot be overridden from TYPE(CHAR) to any other type, nor can a different type value be overridden as TYPE(CHAR). See “Relationship to Control Variables .ATTR and .ATTRCHAR” on page 210.

Refer to the *ISPF Dialog Tag Language Guide and Reference* for details in defining character attributes within dynamic areas in panels created using DTL.

Specifying Character Attributes in a Dynamic Area

If a dynamic area is to contain character attributes, a shadow variable must be defined. The TYPE(CHAR) attributes must be placed in this variable such that they

map to the characters in the dynamic area affected by the attribute. ISPF ignores any other characters or field attributes that are placed in this shadow variable, but it is recommended that blanks be used as filler characters.

Note: If consecutive characters have the same character attributes (an entire word, for example), the attribute character must be repeated in the shadow variable for EACH character affected. For panels to be displayed on DBCS terminals, a TYPE(CHAR) attribute should only map to the first byte of a double-byte character.

The shadow variable is associated with the dynamic area by placing the shadow variable name after the dynamic area name in the panel definition. The two variable names must be separated by a comma only, and the shadow variable name must be followed by a blank.

Note: The dynamic area and shadow variables cannot be Z variables in the panel source.

Refer to the *ISPF Dialog Tag Language Guide and Reference* for details on specifying a shadow variable using Dialog Tag Language.

Conflict Resolution Between Attributes

If the terminal does not support the specified TYPE(CHAR) attribute of color or extended highlighting, this attribute is ignored and defaults to the field attribute.

If the terminal does not support the graphic escape order, or if the character defined by TYPE(CHAR) GE(ON) is not in the range '40'X through 'FE'X, ISPF does not place a GE order in the order stream prior to this character and displays this character as a blank.

- System variable ZGE can be checked by the dialog to determine if the terminal supports the graphic escape order, and if not, can substitute different characters in the dynamic area. The characteristics for this variable are as shown in Figure 52.

Name	Pool	Type	Len	Description
ZGE	shr	non	3	YES - Terminal supports the graphic escape order NO - Terminal does not support the graphic escape order

Figure 52. ZGE Characteristics

Note: When running in GUI mode, ZGE=OFF and any character defined with GE(ON) will display as a blank.

If a TYPE(CHAR) attribute is defined with other keywords, such as INTENS, CAPS, JUST, PAD, and so forth, in addition to COLOR, HILITE, or GE, only the COLOR, HILITE, and GE keywords are recognized. If the GE keyword is specified for any type other than TYPE(CHAR), TYPE(ABSL), TYPE(WASL), or TYPE(CH), it

is ignored. If a TYPE(CHAR) attribute is specified in the shadow variable that contains neither the COLOR nor the HILITE keywords, the character defaults to the field attribute.

Any character attribute specified in the shadow variable that maps to the location of a field attribute character in the dynamic area variable is ignored. (For instance, see Figure 53. A $ in the first character position of the variable SHADOW is ignored because the first character position in the variable CATTAREA is a ¬ indicating a field attribute.)

On DBCS terminals, ISPF ignores any TYPE(CHAR) attribute that maps to a character that precedes the first field attribute. Following the first field attribute, any TYPE(CHAR) attribute that maps to the second byte of a double-byte character is ignored. In addition, the GE(ON) keyword specified for a TYPE(CHAR) attribute that maps to a double-byte character is ignored.

A character attribute specifying the GE(ON) keyword can be defined within a TYPE(DATAIN) field. However, any data typed into this character position might be returned to the dialog as an unpredictable character.

Character attributes are associated with a character and not with the character's position in the buffer. If a character is moved, for example, because of an insert or delete operation, the attribute moves with the character.

The screen image recorded in the list data set as a result of the PRINT, PRINT-HI, PRINTL, or PRINTLHI contains a blank character for all character attributes defined with the GE(ON) keyword.

Figure 53 shows an example of the panel source for a panel with a dynamic area containing character attributes.

```
)ATTR
 * AREA(DYNAMIC)
 $ TYPE(CHAR) HILITE(REVERSE) COLOR(YELLOW)
 ø TYPE(CHAR) COLOR(RED)
 # TYPE(CHAR) COLOR(BLUE) HILITE(USCORE)
 ^ TYPE(DATAOUT) INTENS(LOW) COLOR(WHITE)
)BODY
 %-------------------CHARACTER ATTRIBUTE PANEL------------------------
 %COMMAND ===>_ZCMD


 +The following will contain character attributes:
 *CATTAREA,SHADOW   ----------------------------------------------*

 )END
```

Figure 53. Dynamic Area with Character Attributes

The next example shows how the dynamic area and shadow variables are defined and initialized in a PL/1 program to display the above panel.

```
DECLARE CATTAREA CHAR(50) INIT    /* Dynamic Area Variable */
  ('^These words contain character attributes: Fox Cat');
DECLARE SHADOW CHAR(50) INIT  /* Shadow of Dynamic Area Variable */
  ('                                          $## ø  ');
```

In the panel displayed from the examples above, the `F` in the word `Fox` is yellow and displayed in reverse video, the `ox` in the word `Fox` is blue and underscored, the `C` in the word `Cat` is red with no highlighting, and the `at` in the word `Cat` as well as the rest of the sentence, defaults to the field attribute and is displayed low intensity and white with no highlight.

Formatting Panels That Contain a Graphic Area

ISPF panel definition syntax allows specification of a graphic area within a panel. An ISPF display can contain a picture or graph generated through use of the Graphical Data Display Manager (GDDM) licensed program. A graphic area defined within a panel definition provides part of the interface between ISPF and GDDM. A graphic area can contain either a picture, constructed by use of GDDM services or a graph, constructed by use of the GDDM Presentation Graphics Feature (PGF). Graphic areas can contain alphanumeric fields within them, represented in the usual panel field syntax. These fields can partially overlap the graphic area.

Formatting of a graphic area display is controlled by GDDM.

When specifying a graphic area display, the dialog developer issues a request for the ISPF GRINIT service specifying the name of the panel definition in which the graphic area is defined. This request establishes the interface to GDDM. Next, calls to GDDM that request GDDM services specify the picture to appear in that graphic area. Then the ISPF DISPLAY service is used to display the panel.

The dialog must provide an 8-byte area, called an application anchor block (AAB), which is on a full-word boundary, to the GRINIT call. This AAB identifies the ISPF/GDDM instance and must be used in all GDDM calls made by the dialog. Within the ISPF/GDDM instance, the dialog cannot perform any of the following GDDM calls:

```
ASREAD  FSSHOR  ISFLD   MSPCRT  MSQMOD  PTNSEL  WSCRT
FSSHOW  ISQFLD  MSPQRY  MSQPOS  PTSCRT  WSDEL   WSIO
FSENAB  FSTERM  ISXCTL  MSPUT   MSREAD  PTSDEL  WSMOD
FSEXIT  GSREAD  MSCPOS  MSQADS  PTNCRT  PTSSEL  WSSEL
FSINIT  ISCTL   MSDFLD  MSQGRP  PTNDEL  PTSSPP  WSSWP
FSRNIT  ISESCA  MSGET   MSQMAP  PTNMOD  SPINIT
```

ISPF GDDM services do not run in the background, and thus, cannot be requested in a batch environment. See “Defining the Attribute Section” on page 176 for information using the AREA keyword in the)ATTR section to define a graphic area.

Graphics Panel Processing Considerations

ISPF automatically switches into graphics interface mode when the GRINIT service is requested. This mode continues for the life of the ISPF session. GDDM is called to perform all full-screen displays from this point on, or until a request for the dialog service GRTERM is issued. The following notes apply to graphics interface mode:

Stacked TSO commands
: The field mark key is not available to enter commands at one time.

5550 terminals
: GDDM graphics are supported through the Japanese 3270PC/G Version 3 emulator program. The ISPF-GDDM interface allows DBCS and

mixed-character fields in the panel body, outside the graphics area, to be displayed through GDDM. Full color and highlighting are supported through use of the Japanese 3270PC/G Version 3 and 3270PC Version 5 emulator programs.

3290 terminals
The vertical split function is disabled. Panels are displayed with a larger-size character set. The partition jump key is *not* functional.

Alternate screen widths
You cannot use GDDM with terminal devices whose primary width is different from their alternate width. For example, 3278 model 5.

Autoskip facility
When entering data in a field, GDDM automatically moves the cursor to the next input field when the preceding field is full.

First field attribute
GDDM requires that the first field on a panel begin with an attribute character. Therefore, the ISPF/GDDM interface copies the attribute character for the last field on a panel to the first panel position. This can result in the first byte of the panel data being overlaid.

Data transfer
The entire screen buffer is sent to the terminal even if no fields have been modified.

NUMERIC (ON)
The numeric lock feature is not active when using GDDM.

Graphic output
GDDM calls issued from an application are used to define graphic primitives for the next full-screen output and are unknown to ISPF. Any full-screen output, following the ISPF full-screen output containing the graphic area, can cause the loss of the graphic primitives on the ISPF panel. Hence, the application can be required to reissue the GDDM calls.

Pop-up windows
Pop-up windows cannot be displayed over graphic areas nor can graphic areas be displayed over pop-up windows.

GUI mode
Graphic areas are not supported if you are running in GUI mode. When a GRINIT statement is encountered, you will receive a message that panels with graphics will not be displayed. You may choose to continue. When a panel with graphics is encountered, a pop-up window is displayed asking if you want the panel displayed on your host emulator session or on your workstation without the graph.

Notes:

1. If you are in split-screen mode, the graphic area panel cannot be displayed on the host session.
2. If you specified GUISCRD and GUISCRW values on the ISPSTRT invocation which are different from the actual host screen size, GDDM cannot be initialized and the GRINIT service will end with a return code of 20.

Using DBCS-Related Variables in Panels

The following rules apply to substituting DBCS-related variables in panel text fields.

- If the variable contains MIX format data, each DBCS subfield must be enclosed with shift-out and shift-in characters.

  Example:

  ```
  eeee[DBDBDBDBDB]eee[DBDBDB]
  ```

  ee... represents a field of EBCDIC characters; DBDB... represents a field of DBCS characters; [and] represent shift-out and shift-in characters.
- If the variable contains DBCS format data only, the variable must be preceded by the ZE system variable, without an intervening blank.

  Example:

  ```
  ...text...&ZE&DBCSVAR..text...
  ```
- If the variable contains EBCDIC format data, and it is to be converted to the corresponding DBCS format data before substitution, the variable must be preceded by the ZC system variable, without an intervening blank.

  Example:

  ```
  ...text...&ZC&DBCSVAR..text...
  ```

The ZC and ZE system variables can be used only for the two purposes described above. When variable substitution causes a subfield length of zero, the adjacent shift-out and shift-in characters are removed.

Using Preprocessed Panels

You can store preprocessed panel definitions to reduce transition time. These preprocessed panel definitions are in an encoded format, and cannot be edited directly. Panels preprocessed with ISPF Version 3 Release 3 or Version 3 Release 5 cannot be used with any prior releases.

Preprocessed panel data sets must be defined to ISPF as you would define other data sets. This can be either by normal allocation prior to invoking ISPF, or dynamically during an ISPF session by using the LIBDEF service. ISPF provides a dialog, ISPPREP, for creating preprocessed panels. This dialog can be run either in batch mode or interactively.

You invoke the ISPPREP dialog by:
- Issuing the ISPPREP command from the command line
- Selecting it from the Functions pull-down on the ISR@PRIM panel.
- Specifying ISPPREP with the PGM keyword on the SELECT service request

To run ISPPREP by using the SELECT service, issue ISPPREP with no parameters. For example:

```
ISPEXEC SELECT PGM(ISPPREP)
```

causes the selection panel shown in Figure 54 on page 156 to be displayed. Issuing the ISPPREP command from a command line or invoking ISPPREP from the Functions choice on the action bar of the ISPF Primary Option Menu also causes this selection panel to be displayed.

```
                        Preprocessed Panel Utility

 Specify input and output data set names below:

  Panel input data set:
    Data set name  . . _____________________________________________
    Member . . . . . . ________         ( * for all members )
    Volume serial  . . ______           ( If not cataloged  )

  Panel output data set:
    Data set name  . . _____________________________________________
    Member . . . . . . _________        ( blank or member name )
    Volume serial  . . ______           ( If not cataloged     )

  Enter "/" to select option
  _ Replace like-named members
  / Save statistics for members




 Command ===> ___________________________________________________________
  F1=Help      F2=Split     F3=Exit      F9=Swap     F10=Actions  F12=Cancel
```

Figure 54. Panel for Specifying Preprocessed Panel Input and Output Files (ISPPREPA)

To run ISPPREP in batch mode, you include the PARM keyword, together with the panel-input and panel-output identifiers, on the SELECT service request. For example:

```
ISPEXEC SELECT PGM(ISPPREP) PARM(INPAN('ISPFPROJ.GRE.PANELS(PANA)'),
OUTPAN('ISPFPROJ.PXY.PANELS(PANB)')
```

requests the SELECT service to convert member PANA in ISPFPROJ.GRE.PANELS to the internal format and to write it to member PANB in ISPFPROJ.PXY.PANELS.

Note: The previous example must be run from a REXX or CLIST command procedure.

You can control whether existing members in the output data set having the same identification as that specified will be replaced. In batch mode, use the NOREPL|REPLACE parameter with the PARM keyword for specifying whether members are to be replaced. In interactive mode, use the line provided on the panel shown in Figure 54 for specifying whether members are to be replaced.

ISPPREP converts panel input data set members to an internal format and writes them to the specified output panel data set members. A given panel file can contain a mixture of preprocessed panels and regular panel definitions.

ISPPREP does not destroy the source panels from which it creates preprocessed panels. However, you should save those panels in case they must be updated in the future. When the preprocessed panels are ready for use, you can use them to replace the corresponding source files for the ISPPLIB defaults.

ISPPREP provides an option for generating statistics for preprocessed panels. ISPF provides the version (always 1), modification counter, creation date last-modified date, current number of lines, initial number of lines, number of modified lines (always 0), and user ID for the message or panel. These statistics are visible on memberlist displays such as ISPF BROWSE and EDIT. The statistics are placed in the SPF directory.

Restrictions for Using ISPPREP

When using ISPPREP, you should note that certain restrictions apply to those panel definitions that can be converted to their internal format. These restrictions apply only when creating preprocessed panels and are based on the fact that preprocessed panels cannot have a dynamically defined width and depth.

The following restrictions apply to panel definitions to be converted:

1. The use of a dialog variable with the WIDTH keyword on the)BODY header statement of a panel definition is not allowed.
2. The specification of EXTEND(ON) for the attribute character of a dynamic, graphic, or scrollable area is not allowed.
3. The use of a dialog variable to define a model line in a table display panel definition is not allowed.
4. For DBCS panels, the correct character set must be loaded prior to invoking ISPPREP. Panels to be displayed on a 5550 3270 Kanji Emulation terminal must be converted using the 3278KN character set (set in option 0.1).

Preprocessed panel objects should not be copied from a fixed to a variable record format data set. Blank data could be lost. This can cause the product to abend or can create a display error when the copied panel object is used by display processing. Use ISPPREP to transfer preprocessed panel objects to a variable record format data set or when the receiving data set logical record length or logical record format is not the same as the source data set.

ISPPREP output data sets must conform to the same LRECL limits as ISPPLIB.

Using ISPPREP with the SELECT Service

You can use the PGM keyword of the SELECT service to invoke ISPPREP. The syntax for invoking ISPPREP is as follows:

```
ISPEXEC SELECT PGM(ISPPREP) [PARM(INPAN(PDSin),
                                  OUTPAN(PDSout)
                                 [,INVOL(volser#)]
                                 [,OUTVOL(volser#)]
                                 [,NOREPL|REPLACE]
                                 [,STATS|NOSTATS])]
```

The PARM keyword on the SELECT indicates that ISPPREP is to be run in batch mode. The absence of the PARM keyword indicates that ISPPREP is to be run as an interactive dialog and that **PDSin**, the panel input library, and **PDSout**, the panel output library, are to be specified on a data-entry panel. Both the ISPPREP command and option 2 on the ISP@PRIM primary option panel select ISPPREP in interactive mode.

The panel input and panel output library identifiers, whether specified on the SELECT statement when in batch mode or on the data entry panel when in interactive mode, follow the same guidelines.

PDSin (panel input library)
The name of the library of panel definitions to be converted to their internal format. PDSin must be in the form:

```
('partitioned data set name[(member)]')
```

The member name can be specified either by indicating the specific name or by coding an asterisk. Coding an asterisk for the member name indicates that all members in the specified data set are to be converted to preprocessed panels. This allows conversion of all panel definitions within a data set in one call of ISPPREP.

You cannot specify the same name for the input partitioned data set and for the output data set, even if you specify REPLACE unless the data sets exist on different volumes and you specify the appropriate volume serial numbers by using the INVOL and/or OUTVOL parameters.

When running in batch mode, you are not required to enter a member name. The absence of the member name is equivalent to coding an asterisk for the member name. In interactive mode, failure to explicitly state a member name or an asterisk causes the data-entry panel to be redisplayed with a message prompting the user for the member name.

PDSout (panel output library)
The name of the library to which the preprocessed panels will be written.

The form of PDSout is the same as that of PDSin. You can specify a blank or name for the member name. A blank indicates that the member name specified for PDSin is to be used as the member name for PDSout.

Coding an asterisk for a member name in PDSout is invalid.

INVOL (input PDS volume serial number)
Specifies the serial number of the volume on which PDSin resides. If this parameter is omitted, the system catalog is searched.

It must be used when the data set exists but is not cataloged. INVOL is optionally specified in batch mode as well as in interactive mode. In batch mode the keyword (INVOL) is specified along with the volume serial number as part of the SELECT statement.

OUTVOL (output PDS volume serial number)
Specifies the serial number of the volume on which PDSout resides. If this parameter is omitted, the system catalog is searched.

It must be used when the data set exists but is not cataloged. OUTVOL is optionally specified in batch mode as well as in interactive mode. In batch mode the keyword (OUTVOL) is specified along with the volume serial number as part of the SELECT statement.

NOREPL|REPLACE
A keyword that specifies whether existing partitioned data set members are to be replaced in PDSout. The default is NOREPL in batch mode. In interactive mode, an option must be specified.

STATS/NOSTATS
User controls whether member statistics are to be saved in the SPF directory. The default option is STATS.

Any panel specified in the panel input library that is already a preprocessed panel is copied directly to the panel output library (contingent on the NOREPL|REPLACE specification).

ISPPREP should be invoked with the NEWAPPL keyword specified on the SELECT statement. (This is necessary because ISPPREP issues LIBDEF service calls.) If NEWAPPL is not specified, any LIBDEF issued prior to the execution of ISPPREP can no longer be in effect.

Examples of Using ISPPREP

- Convert PDS member PANA, in ISPFPROJ.GRE.PANELS, and write the preprocessed panel to member PANB, in ISPFPROJ.PXY.PANELS, if it does not already exist. Both PDSs are cataloged.

```
SELECT PGM(ISPPREP) PARM(INPAN('ISPFPROJ.GRE.PANELS(PANA)'),
                         OUTPAN('ISPFPROJ.PXY.PANELS(PANB)'),
                         NOREPL) NEWAPPL
```

- Convert PDS member PANA, in ISPFPROJ.GRE.PANELS, and unconditionally write the preprocessed panel to member PANB, in ISPFPROJ.PXY.PANELS. Both PDSs are cataloged.

```
SELECT PGM(ISPPREP) PARM(INPAN('ISPFPROJ.GRE.PANELS(PANA)'),
                         OUTPAN('ISPFPROJ.PXY.PANELS(PANB)'),
                         REPLACE) NEWAPPL
```

- Convert the entire PDS ISPFPROJ.GRE.PANELS, which contains three members (PANA, PANB, and PANC), and unconditionally write the preprocessed panels to PDS ISPFPROJ.PXY.PANELS, which contains three members also (PANA, PANB, and PANC). Both PDSs are cataloged.

```
SELECT PGM(ISPPREP) PARM(INPAN('ISPFPROJ.GRE.PANELS(*)'),
                         OUTPAN('ISPFPROJ.PXY.PANELS( )'),
                         REPLACE) NEWAPPL
```

- Convert the entire PDS ISPFPROJ.GRE.PANELS, which contains four members (PAN1, PAN2, PAN3, and PAN4) and is cataloged. If the members do not already exist, write the preprocessed panels to PDS ISPFPROJ.PXY.PANELS, which is not cataloged

```
SELECT PGM(ISPPREP) PARM(INPAN('ISPFPROJ.GRE.PANELS(*)'),
                         OUTPAN('ISPFPROJ.PXY.PANELS( )'),
                         OUTVOL(TSOPK7),NOREPL) NEWAPPL
```

- Convert the entire PDS ISPFPROJ.GRE.PANELS and unconditionally write the preprocessed panels to PDS ISPFPROJ.PXY.PANELS. Both PDSs are not cataloged.

```
SELECT PGM(ISPPREP) PARM(INPAN('ISPFPROJ.GRE.PANELS(*)'),
                         INVOL(TSOPK7),
                         OUTPAN('ISPFPROJ.PXY.PANELS( )'),
                         OUTVOL(TSOPK7),REPLACE) NEWAPPL
```

Handling Error Conditions and Return Codes

There are two general classes of error conditions involved with ISPPREP: those associated with the dialog itself, and those associated with the conversion of individual panel definitions.

The dialog error conditions encountered cause immediate termination of ISPPREP conversion processing. If you are operating in interactive mode and recovery is possible, the data-entry panel will be redisplayed with an appropriate message. Otherwise, ISPPREP will terminate. Dialog errors include conditions such as: invalid input or output PDS names; a reference to a non-existent PDS; or a reference to an uncataloged PDS without providing the correct volume serial number.

Panel conversion error conditions apply only to the current panel being converted and are usually due to an error in the panel definition. If such an error is

encountered, processing of the current panel definition halts, and processing of the next panel definition (if it exists) begins. A panel conversion error associated with one panel definition does not effect the conversion of subsequent panel definitions.

ISPPREP logs error and informational messages in ISPLOG. Any error conditions encountered cause an appropriate message and return code to be written to the log. This is also true for any conditions that warrant an informational message.

The following return codes are possible from ISPPREP:

0 Normal completion.

4 Panel definition cannot be processed (see restrictions); NOREPL is specified and the panel (member) already exists in the output library.

8 Panel definition contains syntax errors; panel already in use (enqueue failed) or panel (member) not found.

12 Invalid syntax or keyword in parameter string; data set is not found.

16 Data set allocation or open failure.

20 Severe error.

Since ISPPREP can convert a number of panel definitions to their internal format in one call, a number of conditions may arise that generate a return code other than '0'. ISPPREP returns the highest return code generated. However, if invoked in interactive mode, ISPPREP will return '0' unless an unrecoverable dialog error is encountered, in which case the code returned is '20'. Refer to the log for a more comprehensive look at ISPPREP's results.

Chapter 6. Panel Definition Statement Reference

This chapter is divided into three sections that provide reference information for:

- Panel sections—page 161
- Panel definition statements—page 233
- Control variables—page 266.

The information in each section is arranged in alphabetical order.

Defining Panel Sections

Table 4 on page 104 describes the panel sections *in the order in which they must be defined*.

This section provides reference information for each of the following panel sections in *alphabetical order* :

)ABC—page 161

)ABCINIT—page 167

)ABCPROC—page 168

)AREA—page 169

)ATTR—page 176

)BODY—page 212

)CCSID—page 218

)END—page 219

)HELP—page 219

)INIT—page 220

)LIST—page 221

)MODEL—page 222

)PANEL—page 222

)PNTS—page 226

)PROC—page 230

)REINIT—page 230.

Defining the Action Bar Choice Section

The)ABC (action bar choice) section defines an action bar choice for a panel and its associated pull-down choices. An)ABC section must exist for each action bar choice displayed in the Action Bar area on a panel. The maximum number of)ABC sections on a panel is 40.

```
)ABC DESC(choice-description-text)[MNEM(number)]
```

© Copyright IBM Corp. 1980, 1999

where:

DESC(*choice-description-text***)**
Text displayed in the panel's action bar area for the action bar choice. The maximum length of the text is 64 characters.

The action bar choice-description-text must match the choice-description-text specified in the)BODY section of the panel. ISPF does not translate the value to uppercase. If choice-description-text contains any special characters or blanks, you must enclose it in quotes in the)ABC DESC parameter. However, when it is specified in the)BODY section of the panel, you should not enclose it in quotes. Each action bar choice should be unique.

MNEM(*number***)**
Specifies the position of the character that will be the mnemonic for the action bar text. The letter is designated by an underscore on the display. This keyword, if it exists, must follow the DESC keyword. *Number* is the position of the character (not byte position).

```
)ATTR
 # TYPE(AB)
 @ TYPE(NT)
 ? TYPE(PT)
 $ TYPE ABSL
⋮

)ABC DESC('Menu') MNEM(1)
⋮

)BODY CMD(ZCMD)
@# Menu# Utilities# Compilers# Options# Status# Help@
$-----------------------------------------------------------------------
@                      ?ISPF Primary Option Menu+
⋮
```

For SBCS/DBCS mixed choice-description-text, *number* cannot be the position of a double-byte character position. Shift-in/shift-out bytes are not considered characters. For action bar text containing double-byte characters, we recommend adding a single-byte character, enclosed in parentheses, to the end of the double-byte text. The MNEM (*number*) is the position of this single-byte character. For example:

```
)ATTR
 # TYPE(AB)
 @ TYPE(NT)
 ? TYPE(PT)
 $ TYPE ABSL
⋮

)ABC DESC('OEDDOOUUBBLLEE0F(M)') MNEM(8)
⋮

)BODY CMD(ZCMD)
@# 0EDDOOUUBBLLEE0F(M)# Utilities# Compilers# Options# Status# Help@
$-----------------------------------------------------------------------
@                      ?ISPF Primary Option Menu+
⋮
```

where DD,OO,UU,BB,LL, and EE represent double-byte characters, 0E and 0F are shift-out and shift-in characters. The single-byte character, M, enclosed in parenthesis is the mnemonic letter. The MNEM (*number*), 8, indicates the

underscored mnemonic letter is in the eighth character position (not byte position). Shift-out and shift-in characters are not considered as character positions.

In 3270 mode you access the action bar choice in one of the following ways, where ″x″ is the mnemonic letter that is underscored:

1. Enter ″ACTIONS x″ in the command field
2. Enter ″x″ in the command field and press the function key assigned to the ACTIONS command.

The pull-down menu for that action bar choice displays. If you enter a mnemonic letter, ″x″, that is not found to be an underscored mnemonic letter on the panel, then the cursor is placed on the first action bar choice.

In 3720 mode, panels without a command line will not display mnemonic characters, because there is no command line on which to enter the ACTIONS command and parameter. Terminals or emulators that do not support extended highlighting will not display host mnemonics.

In GUI mode you use a *hot key* to access an action bar choice; that is, you can press the ALT key in combination with the letter that is underscored in the choice. A hot key is also referred to as an *accelerator key* or *shortcut key*. If the character in the ALT character combination is not found to be an underscored mnemonic letter in the panel, then no action is taken.

Note: If you specify duplicate characters (case insensitive) for the mnemonics within the action bar, the result of invoking the mnemonics is operating system dependent.

Note: For each separate action bar choice section, you must define a corresponding)ABCINIT (action bar choice initialization) section. An)APCPROC (action bar choice processing) section is optional. You must include these sections in the panel source definition in the proper order as shown in the following example:

```
)ABC
)ABCINIT
)ABCPROC
```

Specifying Action Bar Choices in Panel)BODY Section

The specification of an action bar choice is included in the panel source immediately following the)BODY panel definition statement header. The order in which the action bar choices are specified indicates to ISPF how the choices will appear in the action bar area on the displayed panel. Internally, action bar choices are numbered sequentially starting from left to right and from top to bottom. The first action bar choice will be numbered one.

```
)ATTR
 @ TYPE(AB)
 # TYPE(NT)
   ⋮

)BODY
 @ choice1@ choice2@ choice3#
```

Notes:

1. A blank must separate the choice-description-text and the AB attribute character. The attribute byte for the first choice can be in any column *except* column 1. A text attribute character to delimit an action bar line should be coded immediately following the last character of the last choice-description-text on each action bar line.
2. A separator line should follow the last action bar line.

   When the panel is displayed in GUI mode, the separator line (the line following the last action bar choice) is not displayed.
3. ISPF considers the panel line following the last action bar choice as part of the action bar area.

The action bar can consist of multiple lines by specifying action bar choices on more than one line in the panel)BODY section.

```
)ATTR
 @ TYPE(AB)
 # TYPE(NT)
   ⋮

)BODY
 @ choice1@ choice2@ choice3#
 @ choice4@ choice5@ choice6#
```

Defining Pull-Down Choices within the)ABC Section

Within each action bar section, pull-down choices are defined with the PDC statement.

```
PDC DESC(choice-description-text)
[UNAVAIL(unavail_variable_name)]
[MNEM(number)]
[ACC(key1[+key2][+key3])]
[PDSEP(OFF|ON)]
```

where:

DESC(*choice-description-text*)
Actual text that is displayed for the pull-down choice it defines. Special characters or blanks must be enclosed within quotes. The maximum length of the text is limited to 64 characters. ISPF numbers each choice. Do not include choice numbers in your text. The pull-down choices defined in each)ABC section are internally numbered sequentially starting with the number one (1,2,...,n) and the number is prefixed to the pull-down choice-description-text.

Note: Numbers do not appear with pull-down choices when you are running in GUI mode.

UNAVAIL(*unavail_variable_name*)
Name of a variable that contains a value to indicate whether or not the pull-down choice is available for selection when the panel is displayed. When the variable contains a value other than 0 (false, therefore available) or 1 (true, therefore unavailable), the variable is ignored and the choice is available. The choice is available even if the specified variable cannot be found.

Note: The current setting is shown as an unavailable choice; that is, it displays in blue (the default) with an asterisk as the first digit of the selection number. If you are running in GUI mode, the choice is *grayed*. ISPF

issues an error message if you try to select it. You can change the color, highlight, and intensity of an unavailable choice by using the CUA Attribute Utility.

MNEM(*number***)**
Specifies the position of the character that will be the mnemonic for the pull-down choice text. The letter is designated by an underscore on the GUI display. *Number* is the position of the character (not byte position). For SBCS/DBCS mixed choice-description-text, *number* cannot be the position of a double-byte character. Shift-in/shift-out bytes are not considered characters.

Note: If you specify duplicate characters (case insensitive) for the mnemonics within the action bar, the result of invoking the mnemonics is operating system dependent.

This keyword is ignored on a 3270 display.

ACC(*key1***[+***key2***] [+***key3***])**
Specifies an *accelerator*, or shortcut, key. This is a key or combination of keys assigned to a menu choice that initiates that choice, even if the associated menu is not currently displayed. The accelerator key text is displayed next to the choice it pertains to on the menu.

The variables *key1, key2,* and *key3* can be any of the following keys: Ctrl, Shift, Alt, Insert, Delete, Backspace, Fn, the single keys a through z, and 0 through 9. The keys Ctrl, Shift, Alt, a through z, and 0 through 9 cannot be used as single accelerator keys. They must be used in combination with other keys.

Rules for accelerator keys
- It is suggested that you avoid using the Alt key combined with a single character key as an accelerator. Alt + char should be used for mnemonic access only. Also, avoid using a function key, or Shift + function key, as an accelerator.
- The following single keys must be used in combination with some other key: Ctrl, Shift, Alt, A—Z, a—z, and 0—9.
- Only one key can be a function key.
- If you use a two key combination, one key must be Ctrl, Shift, or Alt, and the other must be Insert, Delete, Backspace, F1-F12, A-Z, a-z, or 0-9.
- If you use a three key combination, two key must be Ctrl, Shift, or Alt, and the other must be Insert, Delete, Backspace, F1-F12, A-Z, a-z, or 0-9.
- The combined text string cannot exceed 30 characters.

After you define your accelerators, remember to keep the following accelerator search order in mind when you hit a key or combination of keys:
1. Operating system specific definitions. For example, Ctrl+Alt+Delete reboots your OS/2 machine rather than invoke a pulldown choice that might have this key combination specified as an accelerator.
2. Pulldown choice accelerator definitions.
3. Accelerator assigned with the panel. For example, a function key.
4. System menu-type definitions. For example, Alt+F4 is defined in the Communications Manager System Menu as an accelerator for closing the emulator session.

For example, if F2 is defined as an accelerator key on the ISPF Primary Option Panel's Menu pulldown for the *EDIT* pulldown option, and the F2 function key is set to the ISPF SPLIT command, when you hit the F2 key, EDIT is started instead of the screen being split.

Accelerators are a GUI-specific function. An option appears on the ISPF Settings Panel (under GUI settings) that specifies whether or not accelerators are supported. The default is to have the support. If you turn this setting off, accelerators are not functional, and do not appear in the pull-down menus.

PDSEP(OFF|*ON*)
Specifies a *pull-down choice separator bar*. These are separators within a pull-down that group logically related choices.

The separator is a solid line between the previous choice and the first choice in the logical group. You code the PDSEP keyword on the pull-down choice AFTER the separator bar. That is, the separator bar is displayed prior to the choice it is coded on.

Any separator coded on the first pull-down choice is ignored, and because the function is GUI-specific, separator bars are ignored in the host environment.

You must associate the pull-down choice entry field with a variable name. To do this, code a .ZVARS statement in the)ABCINIT section. This variable is used as the pull-down entry field name of each pull-down.

The PDC statement is paired with an optional ACTION statement. When some action is to be performed for a pull-down choice, an ACTION statement must immediately follow the PDC statement defining the pull-down choice.

```
ACTION RUN(command-name) [PARM(command-parms)]
```

where:

RUN(*command-name*)
Required keyword. Specifies the name of a command to be executed. The command name must be 2–8 characters. Coding the keyword ACTION RUN(x), where x is a 1-character command name, results in an error condition. ISPF searches for the command in the application, user, site, and system command tables, if they are defined.

You can use the ISRROUTE command, which is an ISPF command in ISPCMDS, to invoke the SELECT service. The ACTION RUN statement is as follows:

```
ACTION RUN (ISRROUTE) PARM (SELECT 'your-select-command-parms')
```

where *your-select-command-parms* contains the SELECT service invocation and all required parameters. This allows your dialog to not have to create a separate command in the application command table for every RUN statement coded within your dialog panels.

PARM(*command-parms*)
Optional keyword. Specifies the parameters to use when processing the command in the application, user, site, or system command table. Enclose the *command-parms* value in quotes if it contains special characters or blanks.

You can define only one ACTION statement per PDC statement in the)ABC panel section. You can specify the RUN() or PARM() keywords in any order on an ACTION statement. Also, if the RUN() or PARM() keywords are duplicated within an ACTION statement, ISPF will use the last occurrence of the keyword. Figure 55 shows an example of an action bar section definition.

```
)PANEL
)ATTR
 @ TYPE(AB)
 # TYPE(NT)
   ⋮

)ABC DESC(FILE) MNEM(1)
  PDC DESC(file-choice1) ACC(Alt+F1)
  ACTION RUN(command-name) PARM(command-parms)
  PDC DESC(file-choice2) UNAVAIL(&unvar2)
  ACTION RUN(command-name) PARM(command-parms)
  PDC DESC(file-choice3) PDSEP(ON)
  ACTION RUN(command-name) PARM(command-parms)
)ABCINIT
  .ZVARS = PDCHOICE
  &PDCHOICE = '
  &unvar2 = 1
   ⋮

)ABCPROC
  VER (&PDCHOICE,LIST,1,2,3)
   ⋮

)ABC DESC(HELP)
  PDC DESC(help-choice1) MNEM(6)
  ACTION RUN(command-name) PARM(command-parms)
  PDC DESC(help-choice2)
  ACTION RUN(command-name)
  PDC DESC(help-choice3)
  ACTION RUN(command-name) PARM(command-parms)
   ⋮

)ABCINIT
  .ZVARS = PDCHOICE
  &PDCHOICE = '
   ⋮

)ABCPROC
  VER (&PDCHOICE,LIST,1,2,3)
   ⋮

)BODY
 @ FILE@ HELP#
   ⋮

)END
```

Figure 55. Action Bar Section Example

Defining the Action Bar Choice Initialization Section

The)ABCINIT section header statement has no parameters. ISPF associates the first)ABCINIT section it encounters before another panel definition statement header with the previous)ABC section.

)ABCINIT Section

```
)ABCINIT
```

The rules that apply to the)ABCINIT section and its contents are the same as those that apply to the ISPF)INIT panel definition statements. However, the processing is limited to the action bar choice and its pull-down.

The)ABCINIT section runs when the user selects that action bar choice.

Note: If you are running in GUI mode, the)ABCINIT section runs prior to sending the panel to the workstation.

At least one statement must be specified in the)ABCINIT section. The)ABCINIT section must contain a .ZVARS control variable assignment statement to associate a field name with the pull-down entry field.

See “Formatting Panel Definition Statements” on page 233 for additional information.

Defining the Action Bar Choice Processing Section

The)ABCPROC section header statement has no parameters. ISPF associates the first)ABCPROC section it encounters before another panel definition statement header with the previous)ABC section.

```
)ABCPROC
```

The rules that apply to the)ABCPROC section and its contents are the same as those that apply to the ISPF)PROC panel definition statement. However, the processing is limited to the action bar choice and its pull-down.

The)ABCPROC section runs when the user completes interaction with the pull-down choice.

Note: If you are running in GUI mode, the)ABCPROC section runs after the pull-down has been selected at the workstation.

The)ABCPROC section is not required. ISPF verifies all valid pull-down choices for you.

When you manually position the cursor in the action bar area with the CANCEL, END, or RETURN command on the command line, and you press ENTER, or if you manually position the cursor in the action bar area and you press a function key to execute the CANCEL, END, or RETURN commands, the cursor is repositioned to the first input field in the body of the panel. If there is not an input field, the cursor is repositioned under the action bar area. If the request is to execute the EXIT command, the action taken is controlled by the application.

When you use the ACTIONS command to position the cursor in the action bar area and you execute the CANCEL command, the cursor is returned to where it was before the ACTIONS command was executed. A CANCEL command executed from a pull-down removes the pull-down.

See “Formatting Panel Definition Statements” on page 233 for additional information.

Defining the Area Section

The)AREA (scrollable area definition) section allows you to define scrollable areas on a panel. See "Defining the Attribute Section" on page 176 for information on using the AREA(SCRL) keyword to specify that you want a scrollable area. You can see and interact with the total content defined for the panel area by scrolling the area.

Use the)AREA section header to describe the scrollable area.

```
)AREA name [DEPTH(depth)]
```

name
Specifies the name of the scrollable area that is to be matched with the name specified in the)BODY section. This name cannot be specified as a dialog variable.

DEPTH(*depth*)
Optional. Specifies the minimum number of lines in the scrollable area (not including the scroll indicator line) when EXTEND(ON) has been specified. DEPTH has no effect when EXTEND(OFF) is used. The top line is always reserved for the scroll information and is not considered part of the depth value. DEPTH can be used to insure that a required number of lines are displayed. The depth value cannot be specified as a dialog variable. It must be greater than or equal to the number of lines defined for the area in the)BODY section and less than or equal to the number of lines in the)AREA definition.

A panel)AREA section defines the size and the contents of the information to be scrolled. The contents of the)AREA section generally follow the same rules as the)BODY section. See "Panel Definition Considerations" on page 170 for rules concerning the definition of a scrollable area. Multiple scrollable areas can be defined. The name specified immediately following an AREA(SCRL) character in the)BODY section is used to match each scrollable area to its corresponding)AREA section. If the default EXTEND(OFF) is used, you designate the desired depth of the scrollable area by repeating the AREA(SCRL) attribute. If EXTEND(ON) is specified, the minimum depth is the DEPTH specified in the)AREA section.

The width of the scrollable area includes the special characters that designate the vertical sides. These delimiter characters do not represent attribute characters.

The scrollable area is identified in the panel source with a new attribute defined in the)ATTR section. This new attribute designates the borders of the scrollable area. For example:

```
)ATTR
  # AREA(SCRL) EXTEND(ON)
)BODY
#myarea---------#
#               #
#               #
#               #
```

A single character, Z, can be used in the)AREA section, just as it can be used in the)BODY section, as a place-holder for an input or output field. The actual name of the field is defined in the INIT section with the control variable .ZVARS. The actual field names are in a name list, with all the actual field names for the)BODY and)MODEL sections. The actual field names must appear in the name list in the order

they appear in the panel definition, not in the order they will appear when the panel is displayed. The names must appear in the)BODY section, then)MODEL section, and then)AREA section order.

If you have defined several)AREA sections, the .ZVARS must be listed in order from top-to-bottom left-to-right as they appear in the panel definition.

Cursor position determines how an area scrolls. This is called cursor-dependent scrolling. If scroll down is requested, the line on which the cursor is placed is moved to the top line. If the cursor is currently on the top line of the scrollable area, the section is scrolled as total visible lines minus one. On a panel with only one scrollable area, if the cursor is not within the area and scrolling is requested, the area is scrolled by the total visible lines minus one. If scrolling an area causes the last line of an area to not be the last visible line in the area, the cursor is moved so that the last line of the area appears at the last visible line of the scrollable area.

The top line of the scrollable area is reserved for the scroll indicators. Actual information from the)AREA section is displayed beginning on the second line of the scrollable area. The scroll indicators are displayed only if more data was defined in the)AREA section than fits in the panel area.

The scroll indicators are displayed as follows:

More:	**+**	You can only scroll forward.
More:	**–**	You can only scroll backward.
More:	**– +**	You can scroll forward or backward.

Forward and backward function keys should be defined in the keylist for any application panel that has scrollable areas.

The)AREA section can contain any of the items that can be included in the)BODY section except for the following:

- Action Bar lines
- Graphics Area
- Model Section
- Command Line
- Alternate Message Locations
- Another scrollable area using AREA(SCRL)
- Dynamic Area using EXTEND(ON) or SCROLL(ON).

The)AREA section must fit within the general panel limit of 32K.

Panel Definition Considerations

When you are defining a scrollable area, a number of rules apply:

- The area cannot be specified by using a Z-variable place-holder within the panel body.
- To allow for the scroll information, the minimum width for a scrollable area is 20. The minimum depth of the scrollable area is 2.
- If the width of the scrollable area is less than the screen size, you must place appropriate attribute characters around this area so that the data within the area is not inadvertently affected. For example, by using place fields with SKIP attributes following the right-most boundaries of the area, you can insure that the cursor will tab correctly to the next or continued input field within the area.

- You must terminate an input or output field preceding a scrollable area with an attribute character.
- Fields in the scrollable area or text fields cannot be defined to wrap. A field cannot extend beyond one line of the area.
- The initialization of variables in the scrollable area has nothing to do with Z variables. The setting of .ZVARS simply associates the name of a variable with a Z place holder. It does not initialize the variable value.

  An explicit setting of the variable in the)INIT section will initialize the variable whether it is in a scrollable are or not. Normally, variables that are not explicitly defined are set to null by ISPF. This occurs because ISPF tries to retrieve an existing value from the variable pool and finds that it is not defined. ISPF then defines the variable and sets it to null.

  For scrollable areas, ISPF does not retrieve the variable unless it is to be displayed. Therefore, a variable in a scrollable area that is not visible on the screen does not get implicitly initialized. This is true for all the variables. If the user wishes to initialize a variable it can be done by setting the variable to null in the)INIT section.

If an EXTEND(ON) scrollable area is defined on a panel that does not have a)BODY definition that covers the entire depth of the screen on which it is displayed, the)BODY line over which the last line of the scrollable area is defined is repeated for the remaining depth of the glass, or for the remaining number of lines of data in the scrollable area, whichever is larger.

It is good practice to frame a scrollable area or to allow enough blank space so that the definition of the scrollable area is clear. You should consult you own usability standards to determine the best implementation.

Help Panels

When a help panel is defined with a scrollable area, the Left, Right, and Enter keys that currently scroll through the tutorial panels also scroll the scrollable area. When running under tutorial and trying to scroll past the end of the scrollable area, a message will be displayed indicating that no more information is available in the scrollable area. If RIGHT or ENTER is pressed again, ISPF will follow the normal tutorial flow and display the next help panel if one has been defined. The same is true when scrolling to the TOP of the scrollable AREA; a message indicating that no more information is available will be displayed, and if LEFT is pressed, the previous tutorial panel will be displayed if one has been defined.

Cursor positioning usually defines which scrollable area will be scrolled. However, when in tutorial, if the cursor is not within a scrollable area, the first area defined in the)BODY section will be scrolled. The LEFT and RIGHT commands should be included in any keylist specified for a scrollable help panel.

Panel Processing

When a DISPLAY service is issued, the)INIT section is processed before the panel is displayed on the glass. Each time you scroll and the panel is redisplayed, the)PROC and)REINIT sections are not processed. The)PROC section is only processed when the panel is submitted for processing as when the Enter or End key is pressed.

When panel processing is complete and ISPF returns control to the dialog, it is possible that required fields were not displayed. Therefore, unless a VER NB was

coded in the panel for a required field, it is possible that the application user never scrolled the panel to see the field. It is your responsibility to insure that all required information is obtained.

When fields are displayed on a panel, their characteristics can change without the user interacting with the fields. For example, when CAPS(ON) is set for a field, this only affects fields that actually are displayed. If a field is initialized with lowercase letters and it appears on a portion of the panel that is never displayed, the data remains in lowercase even if CAPS(ON) was set for the field.

Scrollable Area Examples

Figure 56 shows an invalid scrollable area definition. The last line of the extendable scrollable area also contains a line of nonextendable text to its right.

```
)ATTR
   # AREA(SCRL)  EXTEND(ON)
   $ AREA(SCRL)
)BODY
%                       New Patient Information
%Command  ===>_ZCMD
%
+Name . . . . . . . . . ._pname                        %
+
#area1 ---------#           $area2 --------------$
#               #           $                     $
#               #           $                     $
#               #           $                     $
#               #           $                     $
#               #           $                     $
#               #           $                     $
                            $                     $
                            $                     $
+
+Please fill in all information.
+
)AREA AREA1 DEPTH(5)
   ⋮

)AREA AREA2 DEPTH(5)
   ⋮
```

Figure 56. Invalid Scrollable Area Definition

Figure 57 on page 173 shows a valid scrollable area definition. It is followed by the actual scrollable panel displays.

```
)ATTR
  # AREA(SCRL) EXTEND(ON)
)BODY
%
%Command ===>_ZCMD
%
+Patient name . . . . ._pname                       %
+
#myarea ------------------------------------------------------------#
+
+Please fill in all information.
+
```

Figure 57. Valid Scrollable Area Definition (Part 1 of 2)

```
)AREA MYAREA DEPTH(5)
+Personal information
+   Address . . . . . . ._address                    %
+   City, State . . . . ._ctyst                      %
+   Zip Code  . . . . . ._zip  %
+   Birth date  . . . . ._birth   %
+   Sex . . . . . . . . ._SX%  (M=Male or F=Female)
+   Marital Status  . . ._MS+1. Married
+                           2. Single
+                           3. Divorced
+                           4. Widowed
+
+   Home phone  . . . . ._hphone       %
+   Work phone  . . . . ._wphone       %
+
+Emergency Contact
+   Name  . . . . . . . ._ename                       %
+   Home phone  . . . . ._ehphone      %
+   Work phone  . . . . ._ewphone      %
++Emergency Contact
+   Name  . . . . . . . ._ename                       %
+   Home phone  . . . . ._ehphone      %
+   Work phone  . . . . ._ewphone      %
+
+Insurance Coverage
+   Insurance Company . ._insure                       %
+   Group number  . . . ._gn%
+   ID number . . . . . ._ID   %
+   Cardholder's name . ._cname                        %
+   Relationship  . . . ._RL+1. Self
+                           +2. Spouse
+                           +3. Parent
+                           +4. Relative
+                           +5. Other
+  Signature on file  . ._SG+ (Y=Yes N=No)
)INIT
   ⋮

)PROC
   ⋮

)HELP
   ⋮

)END
```

Figure 57. Valid Scrollable Area Definition (Part 2 of 2)

Figure 58 on page 175 shows the initial panel display, which contains a scrollable area. `More:  +` indicates that you can now scroll forward in the scrollable area.

```
Command ===>

Patient name . . . . . CECILIA COFRANCESCO

                                                        More:     +
Personal information
   Address . . . . . . . 2825 N. OCEAN BOULEVARD
   City, State . . . . . BOCA RATON, FL
   Zip Code  . . . . . . 33432
   Birth date  . . . . . 00/00/00
   Sex . . . . . . . . . F    (M=Male or F=Female)
   Marital Status  . . . 1  1. Married
                            2. Single
                            3. Divorced
                            4. Widowed


   Home phone  . . . . . (407)395-9446
   Work phone  . . . . . (407)982-6449


Please fill in all information.
```

Figure 58. Scrollable Area Screen Display (Part 1 of 3)

Figure 59 shows the panel display after one scroll request has been processed. `More:  - +` indicates that you can now scroll forward or backward in the scrollable area.

```
Command ===>

Patient name . . . . . CECILIA COFRANCESCO

                                                        More:   - +
   Home phone  . . . . . (407)395-9446
   Work phone  . . . . . (407)982-6449

Emergency Contact
   Name  . . . . . . . . PAULO COFRANCESCO
   Home phone  . . . . . (407)395-9446
   Work phone  . . . . . (407)982-6449

Insurance Coverage
   Insurance Company . . BLUE CROSS BLUE SHIELD
   Group number  . . . . 22
   ID number . . . . . . 45463
   Cardholder's name . . CECILIA COFRANCESCO
   Relationship  . . . . 1  1. Self
                            2. Spouse

Please fill in all information.
```

Figure 59. Scrollable Area Screen Display (Part 2 of 3)

Figure 60 on page 176 shows the panel display after you have completely scrolled through the scrollable area. `More:  -` indicates that you can now only scroll backward in the scrollable area.

```
Command ===>

Patient name . . . . . CECILIA COFRANCESCO

                                                        More:   -
   Home phone  . . . . . (407)395-9446
   Work phone  . . . . . (407)982-6449

Emergency Contact
   Name  . . . . . . . . PAULO COFRANCESCO
   Home phone  . . . . . (407)395-9446
   Work phone  . . . . . (407)982-6449

Insurance Coverage
   Insurance Company . . BLUE CROSS BLUE SHIELD
   Group number  . . . . 22
   ID number . . . . . . 45463
   Cardholder's name . . CECILIA COFRANCESCO
   Relationship  . . . . 1  1. Self
                            2. Spouse

Please fill in all information.
```

Figure 60. Scrollable Area Screen Display (Part 3 of 3)

Defining the Attribute Section

The)ATTR (attribute) section of a panel contains the definitions for the special characters or two-digit hexadecimal codes that are to be used in the definition of the body of the panel to represent attribute (start-of-field/end-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 you do not define attribute characters, ISPF uses defaults.

If specified, the attribute section precedes the panel body. It begins with the)ATTR header statement.

```
)ATTR [DEFAULT(def1def2def3)]
[FORMAT(EBCDIC|DBCS|MIX)]
[OUTLINE([L][R][O][U]|BOX|NONE)]
```

where:

DEFAULT(*def1def2def3*)
You can use the DEFAULT keyword to specify the characters that define a high-intensity text field, a low-intensity text field, and a high-intensity input field, respectively. The value inside the parentheses must consist of exactly three characters, not enclosed in single quotes and not separated by commas or blanks.

The DEFAULT keyword can also be specified on the)BODY header statement.

FORMAT(EBCDIC|DBCS|MIX)
The default value for a TYPE(INPUT) and a TYPE(DATAIN) field is FORMAT(EBCDIC). These two default values can be changed by using the)ATTR statement or the)BODY statement. These values, in turn, can be overridden if explicitly specified on a subsequent statement. For example, the net result of the following two statements is FORMAT(DBCS):

```
)ATTR FORMAT(MIX)
 $ TYPE(INPUT) FORMAT(DBCS)
```

OUTLINE([L][R][O][U] | BOX | NONE)
The default value for OUTLINE is NONE. The default value for TYPE(INPUT) and TYPE(DATAIN) fields can be specified on the)ATTR or)BODY statement and can be overridden by the OUTLINE keyword. For example:

```
)ATTR OUTLINE(U)
  @ TYPE(INPUT) OUTLINE(BOX)
```

The attribute section ends with the)BODY header statement. The number of lines allowed in an)ATTR section depends upon the storage size available.

Using Default Attribute Characters

If not specified explicitly with the DEFAULT keyword, 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
```

These three defaults are the equivalent to specifying:

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

The default values for the JUST (justification) and CAPS (uppercase and lowercase) keywords vary according to how the field is used. JUST and CAPS are attribute statement keywords that are described in “Formatting Attribute Section Statements” on page 178.

You can change the default characters by using 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.

Typically, you use the DEFAULT keyword on the)ATTR header statement if the 3 default characters are to be changed, and additional attribute characters are also to be defined. For example:

```
)ATTR  DEFAULT($ø_)
   ¬ TYPE(INPUT)  INTENS(NON)
   # TYPE(OUTPUT) INTENS(LOW) JUST(RIGHT) PAD(0)
```

In this example, the default characters for text fields are changed to $ for high intensity, and ø for low intensity. The default character for high-intensity input fields is _, the same as the ISPF-supplied default. The example defines two additional attribute characters: ¬ for nondisplay input fields and # for low-intensity output fields. The output fields are to be right-justified and padded with zeros.

You could use DEFAULT on the)BODY header statement, with the entire attribute section omitted, if the only change is to redefine the default characters. For example:

```
)BODY  DEFAULT($ø_)
```

If you use DEFAULT on both the)ATTR and the)BODY header statements, the)BODY specification takes precedence.

Formatting Attribute Section Statements

Each attribute statement defines the attribute character for a particular kind of field. You can define a given attribute character only once. The remainder of the statement contains keyword parameters that define the nature of the field.

Generally, you should choose special (non-alphanumeric) characters for attribute characters so that they will not conflict with the panel text. An ampersand (&), blank (hexadecimal 40), shift-out (hexadecimal 0E), shift-in (hexadecimal 0F), or null (hexadecimal 00) cannot be used as an attribute character.

You can specify a maximum of 127 attribute characters. This limit includes the 3 default characters, attribute overrides, and TBDISPL dual defaults. For action bar panels or panels with scrollable areas, you can specify a maximum of 110 attribute characters. This is because ISPF uses some attribute characters internally.

Note: In attribute keywords, the *value* can 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)INIT section. The current value of the dialog variable must be valid for the particular keyword. In the previous example, the value of dialog variable A must be HIGH, LOW, or NON.

```
attrchar
[ AREA(DYNAMIC) [EXTEND(ON|OFF)][SCROLL(ON|OFF)]
[USERMOD(usermod-code)]
[DATAMOD(datamod-code)]
[AREA(GRAPHIC) [EXTEND(ON|OFF)]]
[AREA(SCRL) [EXTEND(ON|OFF)]]
[ATTN(ON|OFF)]
[CAPS(ON|OFF|IN|OUT]
[CKBOX(ON|OFF)]
[COLOR(value)]
[CSRGRP(x)]
[COMBO(ON|OFF|name)]
[CUADYN(value)]
[DDLIST(ON|OFF|name)]
[DEPTH(d)]
[FORMAT(EBCDIC|DBCS|MIX)]
[HILITE(value)]
[GE(ON|OFF)]
[INTENS(HIGH|LOW|NON)]
[JUST(LEFT|RIGHT|ASIS)]
[LISTBOX(ON|OFF|name)]
[NUMERIC(ON|OFF)]
[OUTLINE([L][R][O] [U]|BOX|NONE)]
[PAD(char|NULLS|USER)]
[PADC(char|NULLS|USER)]
[PAS(ON|OFF)]
[RADIO(ON|OFF)]
[REP(char)]
[SKIP(ON|OFF)]
[TYPE(value)]
[UNAVAIL(ON|OFF)]
[WIDTH(w)]
```

where:

attrchar
The single-character or two-digit hexadecimal code that is assigned to the attributes that follow.

AREA(DYNAMIC) EXTEND(ON | OFF) SCROLL(ON | OFF) USERMOD(*usermod-code***) DATAMOD(***datamod-code***)**
The value in *attrchar* specifies the special character or two-position hexadecimal value that is used to define the dynamic area within the panel body section. In the panel body section, the name immediately following this character identifies the dialog variable that contains the dynamically-formatted string to be displayed in the area. Subsequent lines of the dynamic area are defined in the panel body by placing this character in the starting and ending columns of the dynamic area. Except on the first line of the dynamic area, where the area name immediately follows the left delimiter character, at least one blank must follow the delimiter characters on the left side of the dynamic area. This is a special character, not an actual attribute character. Other fields must not be defined within or overlapping a DYNAMIC area.

EXTEND(ON | OFF)
Specifies whether or not the depth of an area can be automatically increased.

ON Specifies that the depth (number of lines) of an area can be automatically increased, if required, so that the depth of the entire body of the panel matches the depth of the physical screen on which it is being displayed. Accordingly, an *extendable* area can be designated in the panel definition by a single line unless text or other fields are to appear along the graphic area. Only one extendable area can be specified in a panel definition.

Note: Using EXTEND(ON) is not recommended if your dynamic area is displayed in a pop-up. When EXTEND(ON) is used, the panel is extended to the size of the logical screen. If the panel is then displayed in a pop-up, the panel may be truncated at the pop-up border.

The *value* for the EXTEND keyword cannot be specified as a dialog variable.

OFF The default. Specifies that the depth (number of lines) of an area cannot be automatically increased.

SCROLL(ON | OFF)
Specifies whether or not the area can be treated as a scrollable area.

ON Specifies that the area can be treated as a scrollable area. When a panel containing a scrollable area is displayed, the scrolling commands are automatically enabled. Only one scrollable area can be specified in a panel definition.

The *value* for the SCROLL keyword cannot be specified as a dialog variable.

A panel cannot have more than one scrollable area or more than one extended area.

A panel displayed using TBDISPL cannot have a dynamic area defined by SCROLL ON.

Although the panel display service does not perform the scrolling, it does provide an interpretation of the user's scroll request.

OFF The default. Specifies that the area cannot be treated as a scrollable area.

USERMOD(*usermod-code*) and DATAMOD(*datamod-code*)
Specifies a character or two-position hexadecimal value to be substituted for attribute characters in a dynamic area variable following a user interaction. The attribute characters used within the dynamic area are intermixed with the data. These attribute characters designate the beginning of a new data field within the area. When the dynamic area variable is returned to the dialog, *usermod-code* and *datamod-code* are used to replace the attribute character of each field that has been modified, according to the following rules:

- USERMOD specified but DATAMOD not specified

  If there has been any user entry into the field, even if the field was overtyped with identical characters, the attribute byte for that field is replaced with usermod-code.
- DATAMOD specified but USERMOD not specified

  If there has been any user entry into the field, and if the value in the field has changed, either by the user entry or by ISPF capitalization or justification, the attribute byte for that field is replaced with datamod-code.
- Both USERMOD and DATAMOD specified If there has been any user entry into the field but the value in the field has not changed, the attribute byte for that field is replaced with usermod-code.

  If there has been any user entry into the field and the value in the field has changed, either by the user entry or by ISPF capitalization or justification, then the attribute byte for that field is replaced with datamod-code.
- Neither DATAMOD nor USERMOD specified

  The attribute byte for the field is unchanged.

You can specify more than one dynamic area on a panel. The number of dynamic areas in a panel definition is limited only by physical space limitations of the particular terminal being used for the display.

Examples:

```
)ATTR
  # AREA(DYNAMIC) EXTEND(ON) USERMOD(!)
```

The character '!' replaces the attribute byte for each field in the dynamic area that has been touched, not necessarily changed in value, by the user. All other attribute bytes remain as they are.

```
)ATTR
  # AREA(DYNAMIC) EXTEND(ON) DATAMOD(01)
```

The hexadecimal code '01' replaces the attribute byte for each field in the dynamic area that has been touched by the user and has changed in value. All other attribute bytes remain as they are.

```
)ATTR
  # AREA(DYNAMIC) EXTEND(ON) USERMOD(0C) DATAMOD(03)
```

The hexadecimal code '0C' replaces the attribute byte for each field in the dynamic area that has been touched by the user, but has not changed in value. The hexadecimal code '03' replaces the attribute byte for each field in the dynamic area that has been touched by the user and has changed in value. All other attribute bytes remain as they are.

If the datamod or usermod code is one of the following special characters, it must be enclosed in single quotes in the)ATTR section:

```
blank < ( + | ) ; ¬ - , > : =
```

If the desired character is a single quote, use four single quotes: DATAMOD('''').

AREA(GRAPHIC) EXTEND(ON | OFF)

The value in *attrchar* specifies a character or two-digit hexadecimal value, called the graphic attribute character, to be used to define the graphic area (4 corners) within the panel body. If you use a graphics area, this character must be defined; there is no default value. A panel definition can contain only one graphic area.

EXTEND(ON | OFF)

Specifies whether or not the depth of an area can be automatically increased.

ON Specifies that the depth (number of lines) of an area can be automatically increased, if required, so that the depth of the entire body of the panel matches the depth of the physical screen on which it is being displayed. Accordingly, an *extendable* area can be designated in the panel definition by a single line unless text or other fields are to appear along the graphic area. Only one extendable area can be specified in a panel definition.

Note: Using EXTEND(ON) is not recommended if your graphic area is displayed in a pop-up. When EXTEND(ON) is used, the panel is extended to the size of the logical screen. If the panel is then displayed in a pop-up, the panel may be truncated at the pop-up border.

The *value* for the EXTEND keyword cannot be specified as a dialog variable.

OFF The default. Specifies that the depth (number of lines) of an area cannot be automatically increased.

A graphic attribute character cannot have any other attribute properties. For example, it cannot be mixed with attributes such as INTENS, CAPS, JUST, or PAD.

The graphic attribute character is used to define the boundaries of the graphic area in the panel body, as follows:

- The graphic area is defined on the panel as a rectangle. The graphic attribute character is used to define the 4 corners plus the remaining characters of the vertical sides of this rectangle. You delineate the top and bottom of the rectangle with the characters you use to complete the area outline on the screen. For example, in Figure 61 on page 182, the 4 corners and vertical sides are defined by the asterisk character in the)ATTR section. The top and bottom of the area have been completed with dashes.

- A graphic area must be identified with a name that appears in the left top corner, immediately following the first graphic attribute character of that area. The name of the graphic area must be followed by a blank. This name is used when retrieving information about the area through the PQUERY dialog service or the LVLINE panel built-in function. The PQUERY service is described in *ISPF Services Guide*
- A graphic area can contain ISPF-defined alphanumeric fields.
- ISPF-defined alphanumeric fields can partially overlap graphic areas.
- The first line of the graphic area in the panel definition must have the graphic attribute character in the starting and ending columns of the area. If an alphanumeric field overlaps one of the subsequent lines of the graphic area, it must be delimited by a graphic attribute character. See Figure 63 on page 183 for an example.
- Any field preceding a graphic attribute character should be terminated by an ISPF attribute character to prevent GDDM from overlaying the left-most boundary characters of the area. When variable substitution occurs within a text field in the panel body, the field must be terminated by an attribute character prior to a special character defining a graphic area. "Using Variables and Literal Expressions in Text Fields" on page 113 provides additional information about variable substitution in text fields.
- The width of the graphic area includes the graphic attribute character positions.
- The PQUERY service and the LVLINE panel built-in function can be used to obtain information about the size of the graphic area.

These rules are applied in Figure 61.

```
)ATTR
 *  AREA(GRAPHIC)
)BODY
%-------------------- TITLE -------------------
%COMMAND ===>_ZCMD                            %
%
+  (Text or other fields that are part of the
+   normal panel body ... )
+
+   +*PICT1 ----------------------------*
     *                                 *
     *                                 *
     *                                 *
     *                                 *
     *                                 *
     *                                 *
     * ---------------------------------*
)END
```

Figure 61. Panel Definition Illustrating a Graphic Area

In this example, a graphic area is defined. PICT1 is specified as the name of the area. An asterisk (*) is the delimiter character for the vertical sides of the area, and hyphens (-) are the delimiter character for the top and bottom. Note that a blank follows the area name and follows all asterisks (*) other than the asterisk adjacent to PICT1.

Figure 62 and Figure 63 are examples of panel definitions with a graphic area. In Figure 63, note that the alphanumeric field INPUT1 starts at '_' and ends at '|'.

```
)ATTR
 * AREA(GRAPHIC)

)BODY
%             MY COMPANY OPTION PANEL
% Your selection ==>_ZCMD                        +
+
+  1 Our application 1      +*LOGO ----------------*
+  2 Our application 2      +*                     *
+  3 Our application 3      +*                     *
+  4 Our application 4      +*                     *
+  5 Our application 5      +*                     *
+                           +*                     *
+  X Exit                   +* --------------------*
+  T Tutorial                   <--- Graphic  Area --->
)END
```

Figure 62. Panel Definition with Graphic Area

```
)ATTR
 | AREA(GRAPHIC)

)BODY
%       Panel with Overlapping text field
%
% Here is the data as a graph and with editorial text:
+
     +|PIC1 ------------|
      |                 |
      |                 |
      |                 |
      |                 |
  _INPUT1    |          |
      |                 |
      | ----------------|

%        <- graphic area  ->
)END
```

Figure 63. Definition of Panel Graphic Area with Overlapping Text Field

AREA(SCRL) [EXTEND(ON | OFF)

The value in *attrchar* specifies the special character or two-position hexadecimal value that is used to define the borders of the scrollable area in the)BODY section.

EXTEND(ON | OFF)

Specifies whether or not the depth of an area can be automatically increased.

ON Specifies that the depth (number of lines) of an area can be automatically increased, if required, so that the depth of the entire body of the panel matches the depth of the physical screen on which it is being displayed. Accordingly, an *extendable* area can be designated in the panel definition by a single line unless text or other fields are to appear along the graphic area. Only one extendable area can be specified in a panel definition.

Note: Using EXTEND(ON) is not recommended if your scrollable area is displayed in a pop-up. When EXTEND(ON) is used, the panel is extended to the size of the logical screen. If the panel is then displayed in a pop-up, the panel may be truncated at the pop-up border.

The *value* for the EXTEND keyword cannot be specified as a dialog variable.

OFF The default. Specifies that the depth (number of lines) of an area cannot be automatically increased.

ATTN(ON | OFF)
Defines the attention-select attribute of the field; it is valid only for text fields.

ON Specifies that the field can be selected by using the light pen or cursor select key.

OFF The default. Specifies that the field cannot be selected 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.

CAPS(ON | OFF | IN | OUT)
Specifies the uppercase or lowercase attribute of a field. CAPS is not valid for text fields. The CAPS keyword can have any one of the following values:

ON Data is translated to uppercase before being displayed and all input fields are translated to uppercase before being stored.

OFF Data is displayed as it appears in the variable pool and all input fields are stored as they appear on the screen.

IN Data is displayed as it appears in the variable pool, but all input fields on the screen are translated to uppercase before being stored.

OUT Data is translated to uppercase before being displayed. All input fields are stored as they appear on the screen.

Unless you specify a CONTROL ASIS command procedure (CLIST) statement, the use of CAPS(OFF), CAPS(IN), and CAPS(OUT) is negated if the dialog variable is referred to in the command procedure.

If you omit the CAPS parameter, the default is:

- CAPS(OFF) for input or output fields in the)MODEL section of a table display panel
- CAPS(OFF) for DATAIN and DATAOUT fields in dynamic areas
- CAPS(ON) for all other input or output fields.

CKBOX(ON | OFF)
Allows a one-character input field followed by a protected (text or output) field to be processed as a check box in GUI mode. The input field is displayed as a check box and the protected field is the check box description.

The CKBOX keyword can have one of the following values:

ON Process the input field as a check box.

OFF Process the input field as non-check box field. This is the default setting.

If the check box input field is not blank, the check box is initialized as selected (checked). If the check box is selected, a slash character (/) is placed in the check box input field when the panel is processed.

The CKBOX keyword is ignored if the input field is greater than one character, or if the next field following the check box field is not a protected field. An error message is issued if the CKBOX keyword is used on any fields other than input fields, or the selected choice (SC) output field.

Example:

```
)ATTR
@ TYPE(CEF) CKBOX(ON)
$ TYPE(SAC)
)BODY
% -------- CHECK BOX PANEL ---------- +

+  Select options:
   &INSTR+
   @Z$Check box #1 description+
   @Z$Check box #2 description+
   @Z$Check box #3 description+
   @Z$Check box #4 description+

)INIT
 .ZVARS = '(BOX1 BOX2 BOX3 BOX4)'
 IF (&ZGUI = ' ')
   &INSTR = 'Enter '/'' to select option.'
 ELSE
   &INSTR = 'Check box to select option.'
)END
```

COLOR(*value*)

For 3279-B terminals (or other ISPF-supported seven-color terminals), the COLOR keyword defines the color of a field. The value can be: WHITE, RED, BLUE, GREEN, PINK, YELLOW, or TURQ (turquoise). If a color has not been specified and the panel is displayed on a terminal, a default color is generated based on the protection (TYPE) and intensity attributes of the field. Table 7 shows which defaults are the same as the hardware-generated colors for 3279-A (or other ISPF-supported four-color terminals).

Table 7. Color Defaults

Field Type	Intensity	Default Color
Text/Output	HIGH	WHITE
Text/Output	LOW	BLUE
Input	HIGH	RED
Input	LOW	GREEN

If a color has been specified and the panel is displayed on a terminal other than one with features such as those on the 3279-B, the following occurs:

- If an explicit intensity has also been specified for the field, the color specification is ignored. For example:

  ```
  )ATTR
    @ TYPE(INPUT) INTENS(HIGH) COLOR(YELLOW)
  ```

  In this example, COLOR(YELLOW) is ignored except on terminals like the 3279-B. On a 3279-A terminal, for example, the resulting color is red.

- If an explicit intensity has not been specified for the field, the color is used to generate a default intensity. Specification of blue, green, or turquoise defaults to low intensity. Specification of red, yellow, pink, or white defaults to high intensity. For example:

  ```
  )ATTR
    $ TYPE(OUTPUT) COLOR(GREEN)
  ```

  In this example, a low-intensity output field results.
- If neither color nor intensity has been specified for a field, the default intensity is HIGH.

Note: You can make global changes to one or more of the ISPF-supported colors by using the COLOR command or by selecting the Global Color Change choice from the Colors pull-down on the ISPF Settings panel. You can control the colors when you are in GUI mode. Refer to the *ISPF User's Guide* for more information.

COMBO(ON | OFF | *name*)

Enables you to define choices for a combination box in GUI mode. This keyword is used in conjunction with the)LIST section. See "Defining the LIST Section" on page 221 for more information about the)LIST section. The COMBO attribute keyword is valid on input type fields only. The combination box combines the functions of an entry field and a drop-down list (see page 187). It has an entry field and contains a list of choices that you can scroll through to select from to complete the entry field. The list of choices is hidden until you take an action to make the list visible. As an alternative, you can type text directly into the entry field. The typed text does not need to match one of the choices in the list. The width of the input field determines the width of the combination box. If a COMBOBOX field is immediately followed by 3 or more consecutive attributes, then the COMBOBOX will be displayed for the entire length of the field, since the 3 attributes allow space for the COMBOBOX button without overlaying data in the next field. If a COMBOBOX field is not followed by 3 or more consecutive attributes, then the COMBOBOX will be displayed for the length of the field, to avoid overlaying data in the next field, but the COMBOBOX field will scroll to the right so that the user will be able to type in more than enough data to fill the field.

On the host, the application must be made to implement this function. One method to do this is to code the input field with a field-level help panel containing a scrollable list of choices.

The COMBO keyword can have one of the following values:

ON Specifies an input field is to display as a combination box when running in GUI mode.

OFF Specifies an input field is NOT to display as a combination box when running in GUI mode. This is the default setting.

name Specifies a name that is matched with the)LIST section name parameter (see "Defining the LIST Section" on page 221). This name is valid only on a CEF or other input type field. The name is composed of 1 to 8 characters. Alphanumeric characters A-Z, a-z, 0-9, #, $, or @ can be used in the name, but the first character cannot be numeric. Lowercase characters are converted into uppercase equivalents.

Note: The COMBO keyword is supported for any input field type. To keep the following discussion simple, **CEF** is used to mean any input field type, and **SAC** is used to mean any text or output field type.

The COMBO keyword must be used in conjunction with the CSRGRP(x) keyword (see page 187). The CSRGRP(x) keyword must appear on the CEF field that is used to enter the selection on the host, and on the SAC field that identifies the choices in the list. The *x* value is a number that ties the choices to the correct input field, which has the same COMBO keyword and CSRGRP(x) number.

To specify the attributes of a combination box use the following syntax:

```
attribute-char TYPE(input) COMBO(ON|OFF|name) CSRGRP(x) DEPTH(d)
```

where *attribute-char* is the special character or 2-position hexadecimal value that is used to define the field within the panel body section. The *x* in CSRGRP(x) can be a number between 1 and 99. The number is used to group all of the fields with the same value into cursor groups.

The TYPE value must be an input type field. The DEPTH(d) sets the number of rows for the combination box. Values can be from 0 to 99. For example, if you specify DEPTH(8), the combination box contains eight rows of data. If the depth specified is 0, or if the depth is not specified, the default depth is 4.

CSRGRP(*x*)
Enables you to determine which pushbuttons and checkbox fields are grouped together for cursor movement purposes. When pushbuttons or checkboxes are grouped into cursor groups, the cursor up and down keys move the focus through each of the fields within the group. The TAB key moves the focus out of the group, to the next field that is not within this particular group.

To specify the CSRGRP(x) keyword for cursor groups use the following syntax:

```
attribute-char TYPE(PS) CSRGRP(x)
attribute-char TYPE(OUTPUT) PAS(ON) CSRGRP(x)
attribute-char TYPE(CEF) CKBOX(ON) CSRGRP(x)
```

where *attribute-char* is the special character or 2-position hexadecimal value that is used to define the field within the panel body section. The *x* in CSRGRP(x) can be a number between 1 and 99. The number is used to group all of the fields with the same value into cursor groups. If you specify a CSRGRP on a field that is not displayed as a pushbutton, a checkbox, a radio button, listbox, combination box, or drop-down list, then the CSRGRP keyword is ignored.

All pushbuttons and checkbox fields that do not have a CSRGRP defined do not have a cursor group set in GUI mode, which has the same effect as having them all in the same cursor group.

CUADYN(*value*)
Enables you to define dynamic area DATAIN and DATAOUT attributes with CUA attribute characteristics. For more information, see "Specifying Dynamic Areas" on page 205.

DDLIST(ON | OFF | *name*)
Enables you to define choices for a single choice selection list and display the list in a drop-down box in GUI mode. A drop-down list is a variation of a list box (see 193). A drop-down list initially displays only one item until you take action to display the rest of the items in the list.

The DDLIST keyword can have one of the following values:

ON Specifies a single selection list to display as a drop-down list when running in GUI mode.

OFF Specifies a single selection list is NOT to display as a drop-down list when running in GUI mode. This is the default setting.

name Specifies a name that is matched with the)LIST section name parameter (see "Defining the LIST Section" on page 221). This name is valid only on a CEF or other input type field. The name is composed of 1-8 characters. Alphanumeric characters A-Z, a-z, 0-9, #, $, or @ can be used in the name, but the first character cannot be numeric. Lowercase characters are converted into uppercase equivalents.

Note: To keep the following discussion simple, **CEF** is used to mean any input field type, and **SAC** is used to mean any protected text or output type.

The DDLIST keyword must be used in conjunction with the CSRGRP(x) keyword (see page 187). The CSRGRP(x) keyword must appear on the CEF field that is used to enter the selection on the host, and on the SAC field that identifies the choices in the list. The *x* value is a number that ties the choices to the correct input field, which has the same DDLIST keyword and CSRGRP(x) number.

Defining a DDLIST Without a)LIST Section

The following describes how to define a drop-down list using just the attribute keywords DDLIST and CSRGRP. Define the drop-down list by coding the DDLIST(ON) keyword on the CEF field and on the SAC field that identifies the choices that go with the CEF field. The SAC choice fields that have the same keyword settings (DDLIST and CSRGRP) as the CEF field are used to build the list of choices in the list. They are not built into the panel body when the panel is displayed. The fields following the SAC fields should be text or output fields, they are used as the list choice text. If a field following an SAC field is not a text or output field, no entry is built in the list for that field. The data in the drop-down list is displayed in the order that ISPF processes the defined panel body, that is, left to right, and top to bottom.

ISPF initially compares the CEF field with each SAC field for the drop-down list. If a CEF and SAC match is found the drop-down list field is set to the matching SAC choice text field. If no match is found, or if the CEF field is blank, the drop-down list field is set to blank.

To specify the attributes of a drop-down list use the following syntax in the)ATTR section:

```
attr-char TYPE(CEF) DDLIST(ON) CSRGRP(x) WIDTH(w) DEPTH(d)
attr-char TYPE(SAC) DDLIST(ON) CSRGRP(x)
```

Where *attr-char* is the special character or 2-position hexadecimal value used to define the choice entry field, or the SAC field within the panel body section. The other variables listed in the example are:

WIDTH(w)
The value of *w* sets the width of the drop-down list. Values can be from 0 to 99. This parameter is only used when it is specified on a CEF field. If you specify a width, ISPF makes the drop-down list that is displayed the specified width. If you do not specify, or specify a width

of zero, ISPF scans the next field that is not one of the choice numbers or choice text fields for the CEF field to determine the available space for the list. In this case, ISPF sets the width to the smaller value between the available space and the length of the longest choice text string.

This value does not include the DDLIST borders. If you specify WIDTH(5), the DDLIST can contain 5 characters of data. The width you specify should be large enough to hold the longest choice text string. Also ensure that there is enough panel space for it to fit without overlaying other fields on the panel.

Note: Ensure that, from the starting position of the drop-down list, the width that you specify does not extend past the right border of the panel.

DEPTH(d)

The value of *d* sets the number of rows for the list to display. Values can be from 0 to 99. This parameter is only used when it is specified on a CEF field. If you specify a depth, ISPF makes the drop-down list that is displayed the specified depth.

If you specify DEPTH(8), the DDLIST can contain 8 lines of data. If the depth specified is 0, or if the depth is not specified, the default depth is 4.

Example Panel Definition for DDLIST

```
)ATTR
@ TYPE(CEF) DDLIST(ON) CSRGRP(1)
$ TYPE(SAC) DDLIST(ON) CSRGRP(1)
# TYPE(SAC)
)BODY
%-------------------Sample List Panel---------------------------------

+Terminal Characteristics:
  +Screen format
  @Z $1.#Data+   $3.#Max+
     $2.#STD+    $4.#Part+

)END
-------------------------------------------------------------------
```

Defining a DDLIST With a)LIST Section

Another way to define a DDLIST is to build the choices into the)LIST section of the panel. See "Defining the LIST Section" on page 221 for more information about the LIST section.

To specify the attributes of a drop-down list use this syntax:

```
)ATTR
  attr-char TYPE(CEF) DDLIST(name) CSRGRP(x) WIDTH(w) DEPTH(d)
  attr-char TYPE(SAC) DDLIST(ON) CSRGRP(x)
.
.
.
)LIST name
  VAL(value1) CHOICE(choice1)
  VAL(value2) CHOICE(choice2)
```

Where the DDLIST(name) on the CEF field in the)ATTR section matches the name on the)LIST statement. The)LIST section contains the list of choices and

the values for the drop-down list. The data in the drop-down list is displayed in the order in which you define the choices in the)LIST section.

If the choices are also built into the panel body, the SAC attribute must have DDLIST(ON) so that ISPF does not display the choices in the panel body, but uses the choices specified in the)LIST section.

ISPF initially compares the CEF field with each VAL(value) in the named)LIST section. If a CEF and VAL match is found the drop-down list field is set to the matching VAL's choice text. If no match is found, or if the CEF field is blank, the drop-down list field is set to blank.

RECOMMENDATION

Defining drop-down lists is not a trivial task. You might find it less complex if you use the Dialog Tag Language to define panels that contain drop-down lists. Refer to *Dialog Tag Language (DTL) Guide and Reference* for information about the DTL.

When you define drop-down lists, keep the following points in mind:

- The CEF field (or other input field) receives the selection number and the SAC field (or other output or text field) that contains the selection number. The SAC field must be followed by another output or text field with the choice description to be placed in the list.
- The CEF field should not be more than 3 characters long. Only 3 characters are checked and set for CEF fields processed as drop-down lists.
- If the text following the SAC attribute is longer than 3 characters or the CEF field, then the text is truncated to the size of the CEF field, or 3 characters (whichever is smaller when that list choice is selected). Periods at the end of the string are ignored, they are not set into the list entry field with the other text when the choice is selected and the panel is processed.
- If a CEF field has the same CSRGRP value as a previous CEF field, and both of them have the same DDLIST(ON) keyword, then the second CEF field is displayed as an input field and all of the choices with the same keywords are grouped under the first CEF field.
- If a CEF field has a DDLIST(ON) and a CSRGRP value that does not match an SAC field with DDLIST(ON) and a CSRGRP value that comes after it, then the CEF field is displayed as an input field.
- If an SAC field has a DDLIST(ON) and a CSRGRP value that does not match a previous CEF field with DDLIST(ON) and a CSRGRP value, then the SAC field and the description following it do not display.
- If an SAC field is not followed by an output or text field to be used as the list choice text, then the SAC field is not displayed, and there is no entry in the list for that choice.

DEPTH(d)

The value of *d* sets the number of rows for a list box, drop-down list, or combination box to display. Values can be from 0 to 99. This parameter is only used when it is specified on an input field. See the appropriate sections on list boxes, drop-down lists, and combination boxes for more information.

FORMAT(EBCDIC|DBCS|MIX)

For DBCS terminals, the FORMAT keyword specifies the character format for a field.

EBCDIC
: EBCDIC characters only

DBCS
: DBCS characters only

MIX
: EBCDIC and DBCS characters

In a FORMAT(MIX) field, any DBCS character string must be enclosed by a shift-out (hexadecimal 0E) and a shift-in (hexadecimal 0F).

The default value for a TYPE(INPUT) and a TYPE(DATAIN) field is FORMAT(EBCDIC). These two default values can be changed by using the)ATTR statement or the)BODY statement. These values, in turn, can be overridden if explicitly specified on a subsequent statement. For example, the net result of the following two statements is FORMAT(DBCS):

```
)ATTR FORMAT(MIX)
 $ TYPE(INPUT) FORMAT(DBCS)
```

The default value for a TYPE(TEXT) and a TYPE(OUTPUT) field is FORMAT(MIX). The format of a TYPE(TEXT) field cannot be overridden by the execution of an .ATTR or .ATTRCHAR statement. The attempt to do so results in a dialog error.

The pad character for a DBCS field is converted to the corresponding 16-bit character and is then used for padding. Other format fields are padded normally.

The CAPS attribute is meaningful only for EBCDIC and MIX fields. In addition, within a MIX field, the CAPS attribute applies only to the EBCDIC subfields.

GE(ON|OFF)

The GE keyword indicates that a specific character attribute should be preceeded in the order stream by the graphic escape order, provided the terminal supports GE order. The GE order indicates that the character comes from the APL/TEXT character set. This keyword is supported on TYPE(CHAR) within a Dynamic Area, action bar separator lines (TYPE(ABSL)), work area separator lines (TYPE(WASL)), and column headings (TYPE(CH)).

The GE keyword can have one of the following values:

ON
: Specifies that ISPF will place a graphic escape order before the attribute character when building the order stream.

OFF
: The default. Specifies that ISPF will not place a graphic escape order before the attribute character.

If GE(ON) is specified on TYPE(ABSL), TYPE(WASL), or TYPE(CH), and if the characters following these TYPE's in the panel definition are dashes (-) or vertical bars (|), then the appropriate APL character will be used. This results in these panel elements displaying as solid horizontal or vertical lines, instead of broken lines.

Note: If the terminal does not support graphic escape or if you are running under GDDM (i.e., GRINIT service has been issued) then these panel elements will be displayed as coded in the panel definition.

For more information on the GE keyword support on TYPE(CHAR) within a dynamic area, see page "Specifying Character Attributes in a Dynamic Area" on page 150.

HILITE(*value*)
For ISPF-supported terminals with the extended highlighting feature, the HILITE keyword defines the extended highlighting attribute for a field. The *value* can be:

USCORE	Underscore
BLINK	Blinking
REVERSE	Reverse video

No default is assumed if highlighting is not specified. When you are running in GUI mode, the HILITE keyword is ignored.

If highlighting is specified and the panel is displayed on a terminal without the extended highlighting feature, the following occurs:

- If an explicit intensity has also been specified, the highlighting is ignored.
- If an explicit intensity has not been specified for the field, a high-intensity field results. On a 3279-A terminal, there is also color provided by default, as described in Table 7 on page 185.

Examples of Using COLOR and HILITE Keywords

```
@ TYPE(OUTPUT) INTENS(HIGH) COLOR(YELLOW) HILITE(BLINK)
```

the results are as follows:

```
3277,8  –  TYPE(OUTPUT) INTENS(HIGH)
3279-A  –  TYPE(OUTPUT) INTENS(HIGH) *
3279-B  –  TYPE(OUTPUT) COLOR(YELLOW) HILITE(BLINK)
3290    –  TYPE(OUTPUT) HILITE(BLINK)

*  Results in white.
```

INTENS(HIGH|LOW|NON)
Specifies the intensity of the field (HIGH is the default):

HIGH	High-intensity field
LOW	Low-intensity (normal) field
NON	Nondisplay field

You can specify these operands for the basic attribute types (TEXT|INPUT|OUTPUT). NEF is the only CUA panel-element type that supports the INTENS(NON) operand. The remaining CUA panel-element types do not allow the COLOR, INTENS, and HILITE keyword default values to be changed. The NON operand allows you to optionally display comments or directive lines.

For a panel displayed on a color terminal, you can also use the INTENS keyword to generate a default color for the field, as described for the COLOR keyword. INTENS(HIGH) and INTENS(LOW) are ignored for a 3290 terminal and in GUI mode.

JUST(LEFT|RIGHT|ASIS)
Specifies how the contents of the field are to be justified when displayed. JUST 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 type over.

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.

If you omit the JUST parameter, the default is:

- JUST(ASIS) for input or output fields in the)MODEL section of a table display panel
- JUST(ASIS) for DATAIN and DATAOUT fields in dynamic areas
- JUST(LEFT) for all other input or output fields.

LISTBOX(ON | OFF | *name*)

Enables you to define choices for a single choice selection list and display the list in a list box in GUI mode. A list box displays a scrollable list of choices in a box on the display.

The LISTBOX keyword can have one of the following values:

ON Specifies a single selection list to display as a list box when running in GUI mode.

OFF Specifies a single selection list is NOT to display as a list box when running in GUI mode. This is the default setting.

name Specifies a name that is matched with the)LIST section name parameter (see "Defining the LIST Section" on page 221). This name is valid only on a CEF or other input type field. The name can be 1 to 8 characters long. Alphanumeric characters A-Z, a-z, 0-9, #, $, or @ can be used in the name, but the first character cannot be numeric. Lowercase characters are converted into uppercase equivalents.

Note: To keep the following discussion simple, **CEF** is used to mean any input field type, and **SAC** is used to mean any protected text or output type.

The LISTBOX keyword must be used in conjunction with the CSRGRP(x) keyword (see 187). The CSRGRP(x) keyword must appear on the CEF field that is used to enter the selection on the host, and on the SAC field that identifies the choices in the list. The *x* value is a number that ties the choices to the correct input field, which has the same LISTBOX keyword and CSRGRP(x) number.

Defining a LISTBOX Without a)LIST Section

Define the list box by coding the LISTBOX(ON) keyword on the CEF field and on the SAC field that identifies the choices that go with the CEF field. The SAC choice fields that have the same keyword settings (LISTBOX and CSRGRP) as the CEF field are used to build the list of choices in the list. They are not built into the panel body when the panel is displayed. The fields following the SAC fields should be text or output fields, they are used as the list choice text. If a field following an SAC field is not a text or output field, no entry is built in the list for that field.

)ATTR Section

To specify the attributes of a list box use the following syntax in the)ATTR section:

```
attr-char TYPE(CEF) LISTBOX(ON) CSRGRP(x) WIDTH(w) DEPTH(d)
attr-char TYPE(SAC) LISTBOX(ON) CSRGRP(x)
```

Where *attr-char* is the special character or 2-position hexadecimal value used to define the choice entry field, or the SAC field within the panel body section. The other variables listed in the example are:

WIDTH(w)

The value of *w* sets the width of the list box. Values can be from 0 to 99. This parameter is only used when it is specified on a CEF field. If you specify a width, ISPF makes the list box that is displayed the specified width. If you do not specify, or specify a width of zero, ISPF scans the next field that is not one of the choice numbers or choice text fields for the CEF field to determine the available space for the list. In this case, ISPF sets the width to the smaller value between the available space and the length of the longest choice text string.

This value does not include the LISTBOX borders. If you specify WIDTH(5), the LISTBOX can contain 5 characters of data.

DEPTH(d)

The value of *d* sets the number of rows for the list to display. Values can be from 0 to 99. This parameter is only used when it is specified on a CEF field. If you specify a depth, ISPF makes the list box that is displayed the specified depth. If the depth specified is 0, or if the depth is not specified, the default depth is 4.

This value does not include the horizontal scroll bar. If you specify DEPTH(8), the list box can contain 8 lines of data.

Note: Ensure that from the starting position of the List Box, the width specified does not extend past the right border of the panel. Also ensure that from the starting position of the List Box, the depth specified does not extend past the bottom edge of the panel.

Example Panel Definition for LISTBOX

```
)ATTR
@ TYPE(CEF) LISTBOX(ON) CSRGRP(1) DEPTH(4)
$ TYPE(SAC) LISTBOX(ON) CSRGRP(1)
# TYPE(SAC)
)BODY
%-------------------Sample List Panel---------------------------------

+Terminal Characteristics:
  +Terminal Type
  @Z $1.#3277+   $5.#3290A+
     $2.#3277A+  $6.#3278T+
     $3.#3278+   $7.#3278CF+
     $4.#3278A+  $8.#3277KN+

)END
--------------------------------------------------------------------
```

Defining a LISTBOX With a)LIST Section

Another way to define a LISTBOX is to build the choices into the)LIST section of the panel. See “Defining the LIST Section” on page 221 for more information about the LIST section.

To specify the attributes of a list box use this syntax:

```
)ATTR
  attr-char TYPE(CEF) LISTBOX(name) CSRGRP(x) WIDTH(w) DEPTH(d)
  attr-char TYPE(SAC) LISTBOX(ON) CSRGRP(x)
.
.
.
)LIST name
  VAL(value1) CHOICE(choice1)
  VAL(value2) CHOICE(choice2)
```

Where the LISTBOX(name) on the CEF field in the)ATTR section matches the name on the)LIST statement. The)LIST section contains the list of choices and the values for the drop-down list. The data in the drop-down list is displayed in the order in which you define the choices in the)LIST section.

If the choices are also built into the panel body, the SAC attribute must have LISTBOX(ON) so that ISPF does not display the choices in the panel body, but uses the choices specified in the)LIST section.

RECOMMENDATION

Defining list box lists is not a trivial task. You might find it less complex if you use the Dialog Tag Language to define panels that contain list box lists. Refer to *Dialog Tag Language (DTL) Guide and Reference* for information about the DTL.

When you define listboxes, keep the following points in mind:

- The CEF field (or other input field) receives the selection number and the SAC field (or other output or text field) that contains the selection number. The SAC field must be followed by another output or text field with the choice description to be placed in the list.
- The CEF field should not be more than 3 characters long. Only 3 characters are checked and set for CEF fields processed as drop-down lists.
- If the text following the SAC attribute is longer than 3 characters or the CEF field, then the text is truncated to the size of the CEF field, or 3 characters (whichever is smaller when that list choice is selected). Periods at the end of the string are ignored, they are not set into the list entry field with the other text when the choice is selected and the panel is processed.
- If a CEF field has the same CSRGRP value as a previous CEF field, and both of them have the same LISTBOX(ON) keyword, then the second CEF field is displayed as an input field and all of the choices with the same keywords are grouped under the first CEF field.
- If a CEF field has a LISTBOX(ON) and a CSRGRP value that does not match an SAC field with LISTBOX(ON) and a CSRGRP value that comes after it, then the CEF field is displayed as an input field.
- If an SAC field has a LISTBOX(ON) and a CSRGRP value that does not match a previous CEF field with LISTBOX(ON) and a CSRGRP value, then the SAC field and the description following it do not display.
- If an SAC field is not followed by an output or text field to be used as the list choice text, then the SAC field is not displayed, and there is no entry in the list for that choice.

NUMERIC(ON|OFF)

For terminals with the Numeric Lock feature, the NUMERIC attribute keyword

allows users to be alerted to certain keying errors. The NUMERIC attribute keyword is used to specify, for a panel field, whether Numeric Lock is to be activated for data keyed into that field.

ON Specifies that the Numeric Lock feature is to be activated. The terminal keyboard locks if the operator presses any key other than 0 through 9, minus(-), period (.), or duplicate (DUP). ON is valid only for unprotected fields.

OFF Specifies that the Numeric Lock feature is not to be activated. The user can type in any characters. NUMERIC(OFF) is the default value.

On a data-entry keyboard with the Numeric Lock feature, when the user moves the cursor into a field defined by the NUMERIC(ON) attribute keyword, the display shifts to numeric mode. If the user presses any key other than those allowed by the Numeric Lock feature, the DO NOT ENTER message displays in the operator information area and the terminal is disabled. The user can continue by pressing the reset key.

Note: On non-English keyboards with the Numeric Lock feature, the comma sometimes replaces the period as a valid numeric character.

NUMERIC(ON) and SKIP(ON) attributes cannot be specified for the same field. If attempted, ISPF issues an error message.

The NUMERIC(ON) attribute is not supported when GDDM is active.

When running in GUI mode, any panel field defined as NUMERIC(ON) is verified at the workstation. That is, only numeric characters 0 through 9 and special characters comma (,), dash (-), and period (.) are accepted in a numeric only defined field.

OUTLINE([L][R][O][U] | BOX | NONE)

For DBCS terminals, the OUTLINE keyword lets you display lines around any type of field. The keyword parameters specify where the line or lines are displayed.

L Line to the left side of the field
R Line to the right side of the field
O Line over the field
U Line under the field
BOX Line surrounding the field (equivalent to LROU)
NONE
No lines

You can specify any combination of the L, R, O, or U parameters in any order, without intervening blanks.

The default value for OUTLINE is NONE. The default value for TYPE(INPUT) and TYPE(DATAIN) fields can be specified on the)ATTR or)BODY statement, and can be overridden by the OUTLINE keyword. For example:

```
)ATTR OUTLINE(U)
  @ TYPE(INPUT) OUTLINE(BOX)
```

When you are running in GUI mode, the OUTLINE keyword is ignored.

PAD(*char* | NULLS | USER)

Specifies the pad character for initializing the field. This is not valid for text fields. If PAD is omitted, the default is PAD(' ') for output fields.

char Any character, including blank (' '), can be specified as the padding character. If the character is any of the following, it must be enclosed in single quotes:

```
blank < ( + ) ; ¬ , > : =
```

If the desired pad character is a single quote, use four single quotes: PAD('').

NULLS
Nulls are used for padding.

USER Padding character is specified by a user through the ISPF Settings panel.

If the field is initialized to blanks or the corresponding dialog variable is blank, 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, contain the pad character.

Padding and justification work together as follows. At initialization, unless you have specified ASIS, the field is justified 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.

When ISPF processes an input field, it automatically deletes leading or trailing pad characters as follows:

- For a left-justified field, ISPF deletes leading and trailing pad characters.
- For a right-justified field, ISPF deletes leading pad characters and stores trailing pad characters.
- For an ASIS field, ISPF deletes trailing pad characters and stores leading pad characters.

  Regardless of the type of justification, ISPF deletes leading and trailing pad characters for command fields.

In no case does ISPF delete embedded pad characters. It deletes only leading or trailing pad characters.

PADC(*char* | NULLS | USER)
Specifies conditional padding with the specified pad character. The pad character is used as a field filler only if the value of the input or output field is initially blank. The pad character is not displayed in the remaining unfilled character positions if the field has an initial value. Instead, the unfilled positions contain nulls. Otherwise, ISPF treats the PADC keyword like the PAD keyword, including justification and deletion of pad characters before storing variables in the pool.

char Any character, including blank (' '), can be specified as the padding character. If the character is any of the following, it must be enclosed in single quotes:

```
blank < ( + ) ; ¬ , > : =
```

If the desired pad character is a single quote, use four single quotes: PAD('''').

NULLS
Nulls are used for padding.

USER Specifies that a user-defined character be used for padding. You define

the character by using the ISPF Settings panel. PAD and PADC are incompatible. It is not valid to specify both PAD and PADC for the same attribute character.

If PADC is omitted, the default is PADC(USER) for input fields.

PAS(ON|OFF)
PAS is valid for input and output fields only (not for text fields). The point-and-shoot keyword specifies the field as a point-and-shoot field. In GUI mode, output fields specified as point-and-shoot fields are displayed as buttons. The PAS keyword is used in conjunction with the)PNTS point-and-shoot panel section. See "Defining the Point-and-Shoot Section" on page 226 for more information.

For each field on the panel that has been designated as a point-and-shoot field, there must be a corresponding entry in the)PNTS point-and-shoot panel section. If the cursor is placed on a point-and-shoot panel field and the Enter key is pressed, the action associated with the field is executed. In the example below, if the cursor is placed on the point-and-shoot field, BLUE1, and the Enter key is pressed, the variable RED1 is set to RED. In GUI mode, the action is executed when the pushbutton point-and-shoot field is selected. The cursor only remains positioned on the point-and-shoot field if no intermediate panel is displayed and if the dialog does not set the cursor position.

Note: You can use option 0 (Settings) to set the tab key to move the cursor point-and-shoot fields. This changes output fields to input fields, but data is not altered. However, if a variable is used on an output field that is changed to an input field by the tab to point-and-shoot option, and the variable is VDEFINEd to the application, the variable will be truncated. In this case, the application developer should have a temporary panel variable.

ON The field is a point-and-shoot field.

OFF The default. This field is not a point-and-shoot field.

Example:

```
)PANEL
)ATTR
  $ TYPE(PIN)
  } TYPE(PS)
  + TYPE(NT)
  | AREA(SCRL) EXTEND(ON)
  ! TYPE(OUTPUT) PAS(ON) COLOR(RED)
  * TYPE(OUTPUT) PAS(ON) COLOR(BLUE)
  @ TYPE(TEXT) INTENS(LOW) COLOR(RED) PAD(NULLS)
  ø TYPE(TEXT) INTENS(LOW) COLOR(BLUE) PAD(NULLS)
)BODY WINDOW(60,23)
$
%COMMAND ===>_ZCMD
$
$  Press }DEFAULTS$to reinstate defaults
$
+
|S1                                                |
)AREA S1
+                                        +
+                                        +
+     øBLUE  . . . .*BLUE1               +
+     @RED . . . . .!RED1                +
)INIT
```

```
 .cursor = blue1
    &blue1    = ' '
)PROC
 REFRESH(*)
)PNTS
  FIELD(BLUE1) VAR(RED1) VAL(RED)
  FIELD(ZPS00001) VAR(BLUE1) VAL(DEFAULT)
)END
```

RADIO(ON|OFF) CSRGRP(x)

Displays mutually exclusive textual settings choices. These fields must contain at least two choices, one of which is usually selected. A single-choice selection list is the equivalent function on the host. In GUI mode, they appear as radio button groups.

To have a single-choice selection list display as a radio button group, use the RADIO(ON) keyword with the CSRGRP(x) keyword on the CEF type (or other input type) field that is used to enter the selection on the host.

Note: The RADIO keyword is supported for any input, output, or text field type. To keep the following discussion simple, **CEF** is used to mean any input field type, and **SAC** is used to mean any protected text or output type.

For a list of possible selections, attribute type SAC (select available choice) or another text or output field type must be used before the choice selection number. The attribute used for the choice selection number also must have the RADIO(ON) keyword with the CSRGRP(x) keyword. The *x* on the CSRGRP keyword is a number used to identify each radio button group. The CSRGRP number on both the CEF type field and the SAC type field must match. (For more information about CSRGRP, see 187.) The next field must be a text or output field, used as the radio button choice text.

ISPF initially sets the radio button in the group that corresponds to the value in the CEF field. If the CEF field is blank or the value in the field does not correspond with any of the radio button selections, then no radio button is set by default. ISPF then uses the characters following the SAC attribute to set the value into the CEF field with the same CSRGRP(x) number.

The CEF field must be no more than 3 characters, because only 3 characters are checked and set for the CEF fields processed as radio buttons. If the text following the SAC attribute is longer than 3 characters, or longer than the value in the CEF field, then the text is truncated to the size of the CEF field or 3 characters, whichever is smaller when the radio button corresponding to that choice is selected. Periods at the end of the string are ignored.

To specify the RADIO(ON/OFF) CSRGRP(x) keyword for radio buttons, use the following syntax:

```
attribute-char TYPE(CEF) RADIO(ON/OFF) CSRGRP(x)
attribute-char TYPE(SAC) RADIO(ON/OFF) CSRGRP(x)
```

attribute-char

the special character or 2-position hexadecimal value used to define the choice entry field, or the SAC field within the panel body section. The radio button group is defined in the panel body section by using the special character to define the radio button entry field and the radio button choices that go with it.

TYPE(CEF)
field attribute overrides for the CEF fields can be used to set the RADIO(ON) and CSRGRP(x) value for the CEF field.

TYPE(SAC)
or other text or output field type to be used before each of the choice selection numbers.

RADIO(ON|OFF)
ON if the radio button is implemented, OFF if it is not.

CSRGRP(x)
x can be any number from 1 to 99. The number refers to the number of the radio button *group* as a whole, not the individual choices with the radio button group.

For example:

```
)ATTR
@ TYPE(CEF) RADIO(ON) CSRGRP(1)
$ TYPE(SAC) RADIO(ON) CSRGRP(1)
! TYPE(CEF) RADIO(ON) CSRGRP(2)
^ TYPE(SAC) RADIO(ON) CSRGRP(2)
#TYPE(SAC)
)BODY
% -------- Radio Button PANEL ---------- +

+Terminal Characteristics:
  +Screen format  @Z $1.#Data+  $2.#Std+   $3.#Max+   $4.#Part+

  +Terminal Type  !Z ^1.#3277+  ^3.#3278+  ^5.#3290A+ ^7.#3278CF+
                     ^2.#3277A+ ^4.#3278A+ ^6.#3278T+ ^8.#3277KN+
)END
```

Notes about Syntax

- If a CEF field has the same CSRGRP(x) value as a previous CEF field, and both of them have RADIO(ON), then the new CEF field is displayed as an input field.
- If a CEF field has a RADIO(ON) and a CSRGRP(x) value that does not match an SAC with RADIO(ON) and a CSRGRP(x) value that comes after it, then the CEF field is displayed as an input field.
- If an SAC field has a RADIO(ON) and a CSRGRP(x) value that does not match a previous CEF field with RADIO(ON) and a CSRGRP(x) value, then the SAC field is displayed as an output field instead of a radio button.
- If an SAC field is not followed by an output field to be used as the radio button text, then the SAC field is displayed as an output field.
- If the radio button choice text wraps from one row to the next, then the text on the next line is not displayed as part of the radio button choice text, but as normal text.

CAUTION:

- **Radio button groups can appear in a scrollable area, but choices that do not appear in the visible portion of the area are not displayed.**
- **If a radio button group does appear in a scrollable area, and the panel cannot be scrolled to show all of the choices and the CEF field, then it might not be possible to select some of the choices in the radio button group.**
- **If the CEF field is scrolled out of the visible area of a scrollable area, the SAC field and the choice text field that follow it are displayed in the panel body as text or output fields.**

Recommendation

Because of the scrolling restrictions mentioned above, instead of using radio buttons, try using a LISTBOX or DDLIST with the)LIST section for your application.

REP(*character***)**

For DBCS terminals, the REP keyword allows users to view, on panel definitions, the displayable replacements for nondisplayable attribute characters. This provides for the use of a wider range of BODY record attribute characters that can be viewed on panel definitions. These replacement characters are not visible on the actual panel displays.

You can specify any replacement character, but those that must be enclosed in single quotes are as follows: < > ( ) + ; : , = blank.

Replacement characters are defined in the attribute section. Then, in the body section of the panel definition, a record containing only the defined attribute replacement characters is inserted immediately below any field defined by a corresponding statement in the attribute section. Each replacement character must be in the same column position as the attribute character position in the field above.

When the panel definition, for example, is viewed for editing, the data field and the characters that replace the attribute positions are both displayed. However, when the panel is displayed, the record containing the replacement characters is *not* displayed.

Any character immediately above an attribute replacement character in the panel definition is overlaid by the attribute character's hexadecimal code, not by the displayable replacement character.

In the following example, hexadecimal codes 38, 31, 32, and 34 are in the field attribute positions when the panel is displayed. Because these codes are not visible on a display, replacement characters *, !, $, and # are specified for viewing the panel definition.

When the panel is displayed, the attribute position above the asterisk (*) contains hexadecimal 38; the one above the exclamation marks (!) contain hexadecimal 31; the one above the dollar sign ($) contains hexadecimal 32, and the one above the pound sign (#) contains hexadecimal 34. None of these attribute characters is visible on the display, and the panel definition record containing the replacement characters is not displayed.

The field attribute positions on the panel definition can contain any character, illustrated as x in the example below, because they are overlaid by the replacement characters when the panel is displayed.

Example:

```
)ATTR
  38 TYPE(INPUT) FORMAT(DBCS) REP(*)
  31 TYPE(INPUT) FORMAT(EBCDIC) REP(!)
  32 TYPE(TEXT) FORMAT(EBCDIC) REP($)
  34 TYPE(TEXT) FORMAT(MIX) REP(#)

)BODY
  + DBCS input field %===>x VARDBCS +
                           *

  [DBDBDBDBDBDBDBDBDB]===>x VAREBC  +
  #                  $    !
```

Any characters used to replace shift-out or shift-in characters must be less than hexadecimal 40 and must not be hexadecimal 00, 0E, or 0F.

The EXPAND keyword cannot be used for records containing only those characters defined by the REP keyword.

SKIP(ON | OFF)
The SKIP keyword defines the autoskip attribute of the field. It is valid only for text or output (protected) fields (OFF is the default).

ON Specifies that the cursor automatically skips the field. When a character is entered into the last character location of the preceding unprotected data field, ISPF positions the cursor at the first character location of the next unprotected field.

OFF Specifies that the cursor does not automatically skip the field when the condition described for SKIP(ON) occurs.

When you are running in GUI mode, the SKIP keyword is ignored.

TYPE(*value*)
Specifies the TYPE category of the panel element. The default is TYPE(INPUT). The following TYPE values must be coded explicitly; it is not valid to assign any of these values to dialog variables: AB, ABSL, CH, CHAR, CT, DATAIN, DATAOUT, DT, ET, FP, GRPBOX, NT, PIN, PT, RP, SAC, SI, SUC, TEXT, WASL, and WT. For simplicity, the values in examples are shown as literals.

value may be:

Value	Description
AB	AB unselected choices
ABSL	AB separator line
CEF	Choice entry field
CH	Column heading
CHAR	Character attributes in a dynamic area
CT	Caution text
DATAIN	Input (unprotected) field in a dynamic area
DATAOUT	Output (protected) field in a dynamic area
DT	Descriptive text
EE	Error emphasis
ET	Emphasized text
FP	Field prompt

GRPBOX	Group box
INPUT	Input (unprotected) field
LEF	List entry field
LI	List items
LID	List item description
NEF	Normal entry field
NT	Normal text
OUTPUT	Output (protected) field
PIN	Panel instruction
PS	Point-and-shoot
PT	Panel title
RP	Reference phrase
SAC	Select available choices
SC	Selected choice
SI	Scroll information
SUC	Select unavailable choices
TEXT	Text (protected) field
VOI	Variable output information
WASL	Work area separator line
WT	Warning text

Note: TYPE values are grouped into four categories:

- Basic attribute types (TEXT | INPUT | OUTPUT). See "Basic Attribute Types" on page 204.
- Dynamic area types (CHAR | DATAIN | DATAOUT). See "Specifying Dynamic Areas" on page 205.
- CUA panel-element types. See "CUA Panel-Element Types" on page 206.
- Other attribute types. See "Other Attribute Types" on page 208.

UNAVAIL(ON | OFF)

The UNAVAIL attribute keyword is used to show the availability of a choice in conjunction with radio buttons, checkboxes, and pushbuttons.

The UNAVAIL attribute keyword can also be used with the LISTBOX, DDLIST, and COMBO attribute keywords on choices specified in the)LIST section to show the availability of a choice.

In GUI mode, if a LISTBOX, DDLIST, or COMBO choice is set as unavailable, that choice does not appear in the LISTBOX, DDLIST, or COMBO list of choices.

ON Specifies that the choice is *not* available. In GUI mode this means that the choice cannot be selected in the current context. In host mode, you can still select the choice. It is up to the application you are running to display an error message or ignore the choice. You can use the VER statement keywords LISTX or LISTVX to handle an unavailable choice selection.

OFF Specifies the choice is available and can be selected. This is the default setting.

WIDTH(w)

The value of *w* sets the width for a list box or drop-down list. Values can be from 0 to 99. This parameter is only used when it is specified on an input field. See the appropriate sections on list boxes, and drop-down lists for more information.

Basic Attribute Types

For text (protected) fields, the information in the body of the panel following the attribute character is the data to be displayed. Text fields can 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, with trailing blanks stripped, before the panel is displayed.

For input (unprotected) or output (protected) fields in the body of the panel, 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. For input fields, any user-entered information is stored in the variable after the panel has been displayed.

An output field is different from a text field in that an output field has a variable name associated with the field. It also permits padding, capitalization, justification, and refreshing of the data. There is no default attribute character for output fields.

ISPF initializes input fields prior to displaying them. They can be entered (or typed over) by the user. ISPF also initializes output fields prior to displaying them, but output fields cannot be changed by the user. Both input and output fields can have associated caps, justification, and pad attributes. There is no default attribute character for output fields.

The default values for the data-manipulation attribute keywords, by TYPE, are summarized in Table 8.

Table 8. Default Values for Data-Manipulation Keywords

TYPE	CAPS	JUST	PADDING
TEXT	N/A	N/A	N/A
INPUT	ON	LEFT	PADC(USER)
OUTPUT	ON	LEFT	PAD(' ')

The ISPF basic attribute type rules for field types (defined in Table 8) determine which attribute keywords can be used in conjunction with the basic attribute TYPE keywords.

Keyword **Valid For**
CAPS Not valid for text fields
PAD Not valid for text fields
JUST Valid only for input and output fields
ATTN Valid only for text fields
SKIP Valid only for text or output (protected) fields
NUMERIC Valid only for input fields
PADC Valid only for input or output fields
FORMAT(EBCDIC | DBCS | MIX)
EBCDIC
Default value for input fields
MIX Default value for text and output fields
DBCS Valid for text, input, and output fields

Example of Basic Attribute Types: Figure 64 on page 205 shows a panel definition in which an attribute section is included. As previously mentioned, an attribute section is not required in a panel definition if only the default attributes are to be used in the panel body.

```
)ATTR
  * TYPE(TEXT)   INTENS(HIGH) COLOR(WHITE) CAPS(OFF)
  # TYPE(TEXT)   INTENS(HIGH) COLOR(BLUE)  CAPS(OFF)
  @ TYPE(TEXT)   INTENS(LOW)  COLOR(BLUE)  HILITE(REVERSE)
  ? TYPE(TEXT)   INTENS(LOW)  COLOR(TURQ)  CAPS(OFF)
  _ TYPE(INPUT)  INTENS(HIGH) COLOR(YELLOW)
  $ TYPE(INPUT)  INTENS(NON)
  ø TYPE(OUTPUT) INTENS(LOW)  COLOR(TURQ)  CAPS(OFF)
)BODY
* --------------------------@EMPLOYEE RECORD*--------------------------
# SERIAL NO.*===>_SERNUM  +&rbl                                        %
#
#
#   NAME:?&LAST, &FIRST
#
#   ADDRESS:øADDR1                      +
#           øADDR2                      +
#           øADDR3                      +
#           øADDR4                      +
#
#   POSITION:øPOSIT                     +
#
#   YEARS EXPERIENCE:øYRS+
#
#   SALARY:øSALARY +       # PASSWORD*===>$PSW   +
#                              (Password is required for salary)
#
#
* Enter#END*command to terminate application.
#
)PROC
   VER(&SERNUM,NB,NUM)
   .ATTR(.CURSOR) = 'COLOR(RED) HILITE(BLINK)'
)END
```

Figure 64. Attribute Section in a Panel Definition

Specifying Dynamic Areas

TYPE(DATAIN|DATAOUT|CHAR) can be specified for dynamic areas. Use DATAIN and DATAOUT values only for specifying unprotected or protected fields, respectively, within a dynamic area.

You can specify the ATTN, CAPS, COLOR, HILITE, INTENS, JUST, PAD, PADC, and SKIP keywords for DATAIN and DATAOUT fields. You can specify NUMERIC for DATAIN fields. The defaults for CAPS, JUST, and padding are different from those for other panel fields.

The default values for the DATAIN and DATAOUT attribute keywords, by TYPE, are summarized in Table 9.

Table 9. Default Values for DATAIN and DATAOUT Keywords

TYPE	CAPS	JUST	PADDING
DATAIN	OFF	ASIS	PAD(' ')
DATAOUT	OFF	ASIS	PADC(' ')

For more information on TYPE(CHAR) see "Character-Level Attribute Support for Dynamic Areas" on page 150.

CUA Attribute Characteristics in Dynamic Areas: You can define dynamic area DATAIN and DATAOUT attributes with CUA attribute characteristics. You do this with the attribute keyword CUADYN(value) on the TYPE(DATAIN) or TYPE(DATAOUT) attribute statements. DATAIN and DATAOUT fields that you define with the CUADYN(value) keyword are not true CUA attribute fields, but are DATAIN and DATAOUT fields that have taken on CUA attribute characteristics.

The valid values of CUADYN for each TYPE keyword are:

Field Type	Valid Attribute Keyword
DATAIN	CEF, EE, LEF, NEF
DATAOUT	CH, CT, DT, ET, FP, LI, LID, NT, PIN, PT, SAC, SI, SUC, VOI, WASL, WT

The CUADYN(value) keyword is ignored on any type other than DATAIN or DATAOUT. The values allowed on the TYPE(DATAOUT) statement are ignored if specified on the TYPE(DATAIN) statement, and the reverse is also true.

After the DATAIN or DATAOUT attribute is defined with CUA attribute characteristics, the color, intensity, and highlighting of the attribute can only be overridden using the CUA Attribute Color Change utility.

CUA Panel-Element Types

The CUA guidelines define the default colors and emphasis techniques for individual panel elements. The CUA guidelines also request that application users be allowed to change the color and emphasis for individual panel elements. To conform with CUA principles, ISPF provides the following panel-element attributes. The CUA Attribute Change Utility, which is invoked with the CUAATTR command or by selecting the CUA attributes... choice from the Colors pull-down on the ISPF Settings panel, allows you to change the color and emphasis for individual panel elements.

You can define those panel-element attributes that have a TYPE keyword value in the panel attribute section. The panel-element attributes without a TYPE keyword value are used internally by ISPF in response to user interactions.

The following field types of the CUA panel-element attributes play a major role in determining which attribute keywords can be used with the CUA panel-element attribute values.

Field Type	Valid Attribute Keyword
Input, Unprotected	CEF, EE, LEF, NEF
Output, Protected	VOI, LID, LI
Text, Protected	ABSL, CH, CT, DT, ET, FP, NT, PIN, PS, PT, SAC, SI, SUC, WASL, WT
Text, Unprotected	AB, RP

The ISPF CUA attribute type rules for field types (defined in Table 10 on page 207) determine which attribute keywords can be used in conjunction with the CUA panel-element TYPE keywords.

Table 10 on page 207 lists the CUA values for the TYPE keyword. With each TYPE keyword are listed additional attribute keywords and their default values.

Table 10. CUA TYPE Default Keyword Values

TYPE Keywd Value	COLOR *	INTENS *	HILITE *	CAPS	JUST	PAD	PADC	SKIP	NUM-ERIC	FORMAT
AB	WHITE	HIGH	NONE	N/A	N/A	N/A	N/A	N/A	N/A	MIX
CEF	TURQ	LOW	USCORE	OFF	LEFT		B	N/A	OFF	EBCDIC
EE	YELLOW	HIGH	REVRSE	OFF	LEFT		6D	N/A	OFF	EBCDIC
LEF	TURQ	LOW	USCORE	OFF	ASIS		B	N/A	OFF	EBCDIC
NEF	TURQ	LOW [1]	USCORE	OFF	LEFT		B	N/A	OFF	EBCDIC
RP	WHITE	HIGH	NONE	N/A	N/A	N/A	N/A	N/A	N/A	MIX
ABSL	BLUE	LOW	NONE	N/A	N/A	N/A	N/A	OFF	N/A	MIX
CH	BLUE	HIGH	NONE	N/A	N/A	N/A	N/A	OFF	N/A	MIX
CT	YELLOW	HIGH	NONE	N/A	N/A	N/A	N/A	OFF	N/A	MIX
DT	GREEN	LOW	NONE	N/A	N/A	N/A	N/A	OFF	N/A	MIX
ET	TURQ	HIGH	NONE	N/A	N/A	N/A	N/A	OFF	N/A	MIX
FP	GREEN	LOW	NONE	N/A	N/A	N/A	N/A	OFF	N/A	MIX
NT	GREEN	LOW	NONE	N/A	N/A	N/A	N/A	OFF	N/A	MIX
PIN	GREEN	LOW	NONE	N/A	N/A	N/A	N/A	OFF	N/A	MIX
PS	TURQ	HIGH	NONE	N/A	LEFT	B		OFF	N/A	MIX
PT	BLUE	LOW	NONE	N/A	N/A	N/A	N/A	OFF	N/A	MIX
SAC	WHITE	LOW	NONE	N/A	N/A	N/A	N/A	OFF	N/A	MIX
SI	WHITE	HIGH	NONE	N/A	N/A	N/A	N/A	OFF	N/A	MIX
SUC	BLUE	LOW	NONE	N/A	N/A	N/A	N/A	OFF	N/A	MIX
WASL	BLUE	LOW	NONE	N/A	N/A	N/A	N/A	OFF	N/A	MIX
WT	RED	HIGH	NONE	N/A	N/A	N/A	N/A	OFF	N/A	MIX
LI	WHITE	LOW	NONE	OFF	ASIS	B		OFF	N/A	MIX
LID	GREEN	LOW	NONE	OFF	ASIS	B		OFF	N/A	MIX
VOI	TURQ	LOW	NONE	OFF	LEFT	B		OFF	N/A	MIX

Notes:

1. The attribute keywords whose value is denoted with N/A (not applicable) are not valid to use in conjunction with the corresponding TYPE keyword value.
2. It is not valid to use the attribute keywords FORMAT, REP, and OUTLINE with TYPE(AB). If used, the default values remain in effect.
3. You *cannot* change the keyword values for COLOR, INTENS, or HILITE. This is indicated with an * in the preceding table. If you attempt to change these keyword values, you will get an error condition. The exception is the CUA attribute type, NEF. NEF supports the INTENS(NON) keyword value.
4. You can change the default values of COLOR, INTENS, and HIGHLIGHT by using the CUAATTR command or by selecting the CUA attributes... choice from the Colors pull-down on the ISPF Settings panel.
5. The character `B` in the PAD column stands for *blank*. The PAD and PADC keywords are mutually exclusive, so when PAD is set to `B` (blank, X'40') PADC cannot be set. The EE pad character is X'6D', underscore.

1. You may specify the INTENS(NON) keyword with the CUA type NEF.

6. Three keywords not shown on this table are ATTN, REP, and OUTLINE. ATTN always is N/A, REP is defined by the dialog, and OUTLINE is NONE.
7. Another keyword not shown on this table is CKBOX. CKBOX is only used with TYPE(CEF). This keyword is ignored when running in non-GUI mode. For more information about using CKBOX, see the "CKBOX Keyword" on page 185.

Table 11 lists the CUA panel-element attributes that are used *internally* by ISPF in response to user interactions. These panel-element attributes do not have a TYPE keyword, so you cannot code them in the panel attribute section. They are considered as field-type text (that is, protected). The related attribute keywords and their default values are shown for each.

Table 11. Internal Attributes without TYPE Keyword Values

Panel Element Attribute	COLOR	INTENS	HILITE
AB Selected Choices	YELLOW	LOW	NONE
PD Choices	BLUE	LOW	NONE
Function Keys	BLUE	LOW	NONE
Informational Message Text	WHITE	HIGH	NONE
Warning Message Text	YELLOW	HIGH	NONE
Action Message Text	RED	HIGH	NONE

You can change the default values of COLOR, INTENS, and HIGHLIGHT by using the CUAATTR command or by selecting the CUA attributes... choice from the Colors pull-down on the ISPF Settings panel.

Other Attribute Types

The *other attribute types* consist of the Group Box (GRPBOX) and Selected Choice (SC).

Group Box: A *group box* is a rectangle that is drawn around a group of related fields. The upper-left corner of the box contains a label for the group. Group boxes display in GUI mode only.

To specify a group box, use the type keyword **GRPBOX**. Its syntax is:

```
attribute-char TYPE(GRPBOX) WIDTH(wvalue) DEPTH(dvalue)
```

Where:
- *attribute-char* is the special character or 2-position hexadecimal value used to define the group box area within the panel body section. The area is defined by using the special character to position the upper-left corner of the group box in the panel body section.
- *wvalue* is the width of the group box, not including the borders. This value can be 0 to 99. For example, a specification of WIDTH(9) means the box can contain data 9 characters wide.
- *dvalue* is the depth of the group box, including the group box title line. This value can be 0 to 99. A minimum of 2 lines must be defined for the box. The top line is reserved for the label. For example, a specification of DEPTH(5) means the box consists of a group box title and 4 lines of data.

In the panel body section, the name immediately following the special character for the upper-left corner of the group box identifies the dialog variable that

contains the text for the group box label. In Figure 65 on page 210, that name is *gbar*. The name cannot be specified by using a Z-variable placeholder within the panel body.

Some things to remember when defining group boxes are:

- Input/output/text fields should have ending attributes within the group box, or blanks where the box border falls.
- Dynamic areas are allowed within group boxes, and should be entirely contained within the box.
- Group boxes cannot be defined within dynamic areas.
- Dynamic areas and group boxes should not overlap.
- Scrollable areas are allowed within group boxes, and should be entirely contained within the box.
- Group boxes are allowed within scrollable areas, and should be entirely contained within the area.
- Scrollable areas and group boxes should not overlap.
- Group boxes should not be used with graphic areas.
- If the parameters WIDTH and DEPTH are not specified, the group box does not display.
- If you specify WIDTH with no DEPTH, DEPTH(0) is assumed. This means the group box ends at the bottom of the panel.
- If you specify DEPTH with no WIDTH, WIDTH(0) is assumed. This means the group box does not display.
- If the group box DEPTH is coded as zero and the group box is within a scrollable area, the group box expands to the bottom of the scrollable area.
- If the depth of the scrollable area is less than the group box DEPTH, the group box ends at the bottom of the visible scrollable area. The group box DEPTH is expanded when scrolling up, as long as the maximum group box depth has not been reached and the group box title is within the displayed portion of the scrollable area. After the group box title is no longer displayed in the scrollable area, the group box no longer appears.

Note: Even though the type GRPBOX is considered an output field, it maps to the CUA panel-element type Column Heading (CH). Therefore, its color, intensity, and highlight values can only be changed through the CUA Attribute Change Utility.

```
)ATTR
  + TYPE(TEXT)   INTENS(low) SKIP(on)
  % TYPE(TEXT)   INTENS(HIGH) SKIP(on)
  _ TYPE(INPUT)  INTENS(HIGH) CAPS(ON)
  # TYPE(GRPBOX) WIDTH(44) DEPTH(7)
)BODY
* -------------------------Group Box Example--------------------------
%COMMAND ===>_ZCMD
+                                          +
+    #gbar                                 +
+                                          +
+     +Available           Desired+        +
+       +Cruise Control      Sunroof+      +
+       +AM/FM Stereo        AM/FM Stereo  +
+       +Power Brakes+                     +
+       +Sunroof+                          +
+                                          +
+                                          +
)INIT
&zcmd = &z
&gbar = 'Options'
)REINIT
&zcmd = &z
)PROC
)END
```

Figure 65. Group Box Definition

Selected Choice: The Select Choice (SC) type is an output (protected) field to be used in conjunction with the UNAVAIL attribute keyword.

When TYPE(SC) is coded with the UNAVAIL(OFF) attribute, the field has the color, intensity, and highlighting characteristics of TYPE(SAC).

When TYPE(SC) is coded with the UNAVAIL(ON) attribute, the field has the color, intensity, and highlighting characteristics of TYPE(SUC).

You can use field overrides on the choices.

Relationship to Control Variables .ATTR and .ATTRCHAR

This section describes appropriate and inappropriate override conditions for CUA and basic panel-element attributes. See ".ATTR and .ATTRCHAR" on page 269 for information on .ATTR and .ATTRCHAR.

- TYPE

  CUA panel-element attribute TYPE keywords can be overridden by .ATTR or by .ATTRCHAR. You can change the TYPE:
  - From INPUT/CUA input types to OUTPUT/CUA output and input types
  - From OUTPUT/CUA output types to INPUT/CUA input and output types
  - From TEXT/CUA text types to TEXT/CUA text types

  Some exceptions are:
  - Only TYPE keyword values that have a field type of input can be overridden with TYPE(EE)—error emphasis.
  - CUA attribute types AB, RP, ABSL, and PS cannot be overridden, nor can they be used to override text fields.

– TYPE keyword GRPBOX can only be overridden with .ATTR(field), where *field* is the dialog variable name for the group box as specified in the)BODY section.

- COLOR, INTENS, HILITE

  If you change a basic attribute type to a CUA attribute type, the attribute takes on the characteristics of that particular CUA type, including the default COLOR, HILITE, and INTENS keyword values. For example, if you change a TYPE(INPUT) INTENS(HIGH) attribute to TYPE(NEF), the default color for the attribute changes from red to turquoise, the default color of the NEF attribute type. Also, after you change a basic attribute type into a CUA attribute type, the color, highlight, and intensity can only be overridden by using the CUA Attribute Color Change utility.

  For example, hoping to change the TYPE(INPUT) to CUA TYPE(NEF) with the color pink, you enter the following:

  ```
  .ATTR(FIELD1) = 'TYPE(NEF) COLOR(PINK)'
  ```

  The result is that the field is changed to CUA TYPE(NEF), but when the COLOR(PINK) keyword is processed a dialog error message is given stating that the color of the CUA attribute cannot be overridden.

  If you try to enter:

  ```
  .ATTR(FIELD1) = 'COLOR(PINK) TYPE(NEF)'
  ```

  The COLOR(PINK) keyword is processed before the TYPE(NEF) keyword. Thus, no error message is given concerning the changing of the color of a CUA attribute. However, when the TYPE(NEF) keyword is processed, the attribute type is changed to the CUA default color, and subsequent attempts to change the attribute's color, intensity, or highlighting result in a dialog error message.

  If you change a CUA attribute type to a basic attribute type, only the type changes. The other characteristics associated with the type do not change. For example, the color associated with the CUA type does not change unless you specifically override the color using the COLOR keyword. If you change the CUA type ET to basic type TEXT, the color remains turquoise unless you purposely override it.

- CAPS, JUST, PAD, PADC, SKIP, ATTN, NUMERIC, FORMAT, REP, OUTLINE

  If the keyword is applicable on the)ATTR statement, it can be overridden using the attribute override statements. Those panel attribute keywords whose value is denoted as N/A (not applicable) are not valid in attribute override statements.

- CUADYN(value) keyword

  The CUADYN(value) attribute keyword can be used in .ATTRCHAR statements for DATAIN or DATAOUT attribute characters. The keyword values listed in "CUA Attribute Characteristics in Dynamic Areas" on page 206 for DATAOUT attributes can only override DATAOUT attribute characters. Those listed for DATAIN attributes can only override DATAIN attribute characters.

- Input fields with LISTBOX(ON|name) or DDLIST(ON|name)

  You can override input fields with LISTBOX(ON|name) or DDLIST(ON|name) that are coded in the)ATTR section. You do this by using the .ATTR or .ATTRCHAR statements to set LISTBOX, DDLIST, CSRGRP, WIDTH, and DEPTH values for the input field.

- Input fields with COMBO(ON|name)

You can override input fields with COMBO(ON|name) that are coded in the)ATTR section. You do this by using the .ATTR or .ATTRCHAR statements to set COMBO, CSRGRP, and DEPTH values for the input field.

Defining the Body Section

The)BODY (panel body) section of the panel definition specifies the format of the panel as the user sees it. Each record in the body section corresponds to a line on the display.

The section begins with the)BODY header statement, which can be omitted if there are no preceding sections and no change to the default attribute characters. The)BODY header statement and all associated keywords must be specified on the same line. The panel body ends with any of the following statements:

-)MODEL
-)AREA
-)INIT
-)REINIT
-)PROC
-)HELP
-)PNTS
-)LIST
-)END.

```
)BODY
[KANA]
[WINDOW(width,depth)]
[CMD(field name)]
[SMSG(field name)]
[LMSG(field name)]
[ASIS]
[WIDTH(width)]
[EXPAND(xy)]
[DEFAULT(def1def2def3)]
[FORMAT(EBCDIC|DBCS|MIX)]
[OUTLINE([L][R][O] [U]|BOX|NONE)]
```

Notes:

1. There are system-defined (default) areas for the display of messages and the command field. You can specify alternate locations using the WINDOW, CMD, SMSG, LMSG, and ASIS keywords on the)BODY header statement.
2. The WIDTH and EXPAND keywords on the)BODY header statement control the width of a panel. Both keywords are optional. You can specify either or both. However, if the panel definition width is greater than 80 characters, the WIDTH keyword must be used. If the WIDTH keyword is used, the WIDTH variable must be set in the variable pool before the panel is displayed.
3. DEFAULT, FORMAT, and OUTLINE can also be specified on the)ATTR section statement. The values specified on the)BODY section statement take precedence.

where:

KANA
Include the KANA keyword when Katakana characters will appear within the panel and you have not specified an extended code page using the)CCSID section.

WINDOW(*width,depth*)

Identifies the width and depth of the window that the Dialog Manager uses when displaying the panel in a pop-up window. The values do not include the panel borders; the Dialog Manager adds them outside of the dimension of the width and depth values.

Note:

When you are running in GUI mode, the width you specify is respected regardless of whether or not the panel is displayed in a popup window. The depth is honored when the panel is displayed in a popup. If you specify a depth greater than the depth of the panel definition, extra lines are generated to fill the space. Any extendable areas (such as, AREA(DYNAMIC), or AREA(SCRL) with EXTEND(ON)) might be truncated at the popup depth.

For panels not displayed in a popup window, the depth is the minimum of the specified depth and the actual number of)BODY records in the panel definition. Extendable areas are not truncated. That is, the depth expands to the length of the logical screen.

The width that you specify must be a numeric value greater than or equal to the minimum width of 8 characters. The depth that you specify must be a numeric value greater than 0.

Note: The width and depth cannot be specified by a dialog variable.

For panels that are not being displayed in a pop-up window (no active ADDPOP), ISPF validates the width and depth values against the screen size and issues an error message if either of the following is true:

- The width is greater than the current device width.
- The depth is greater than the current device depth.

For help panels and panels that are being displayed in a pop-up window (after ADDPOP service), ISPF validates the width and depth values against the screen size minus the frame and issues an error message if:

- The depth is greater than the screen depth minus 2.
- The depth is less than the screen depth minus 2 and the width is greater than the screen width minus 3.
- The depth is equal to the screen depth minus 2 and the width is greater than the screen width minus 4.

When running in GUI mode, the frame will be what you specified on ISPSTART unless its ADDPOP was specified in a dialog. In this case, the frame is a dialog frame.

The Dialog Manager recognizes the WINDOW keyword for panels displayed in a pop-up window (after an ADDPOP service request has been issued), and when running in GUI mode. If the panel is not being displayed in a pop-up window and you are not in GUI mode, ISPF validates the keyword, but ignores it. If the text on the panel you are defining exceeds the width of the window, the panel fields do not wrap. All fields end at the window width.

Note: Text coded in column 1 of the panel body does not appear when a panel is displayed in a pop-up window. This occurs because ISPF places a field attribute in the column following the pop-up border character, due

to hardware requirements. Without the field attribute after the border character, subsequent panel text would have the attributes (color, intensity, and so on) of the window frame. Therefore, your panel text should be coded so that it does not start in column 1 of the body if you are going to display your panel in a pop-up window.

Attributes coded in column 1 of the panel body overlay the field attributes that ISPF generates following the left side of the window frame. Therefore, an attribute coded in column 1 of the panel will be in effect for subsequent text.

CMD(*field-name*)
Identifies the panel field (variable name) to be treated as the command field. The field type must be a CUA input type. If the CMD keyword is omitted from a)BODY statement, ISPF uses the first input field as a default command field. You can specify that you do not want a command field by using CMD(). Do not use this option for a table display. You must have a command field for a table display.

SMSG(*field-name*)
Identifies the panel field (variable name) where the short message, if any, is to be placed. The field type must be a CUA output type. If the message is longer than the length of this field, the message is placed in a pop-up window. The SMSG keyword does not effect placement of the TOP-ROW-DISPLAYED indicator which is right-justified on the top line of the display, or just below the action bar separator line if an action bar is defined.

LMSG(*field-name*)
Identifies the panel field (variable name) where the long message, if any, is to be placed. The field type must be a CUA output type. If the message is longer than the length of this field, the message is placed in a pop-up window.

Notes:

1. For CMD, SMSG, and LMSG the field-name must be within the)BODY section, not within a scrollable area or table.
2. For long or short messages in pop-up windows, if the message originates from panel processing, as in a verification error message, the message pop-up window is placed adjacent to the field that is the object of the validation.
3. The format of the command, long-message, and short-message fields must not be FORMAT(DBCS). Because a FORMAT(EBCDIC) field does not display DBCS characters correctly, FORMAT(MIX) is recommended.
4. For additional information about the placement of the command and long message fields, see the *ISPF User's Guide*

ASIS
Specifies that the command and long message fields are to appear on the display as specified in the panel definition. When ASIS is specified, any user request, using SETTINGS option 0 or by setting system variable ZPLACE, to reposition the command and long message fields is ignored.

WIDTH(*width*)
The number of columns to use in formatting the panel. *width* can be a constant or a dialog variable, including the system variable &ZSCREENW The specified width must not be less than 80 or greater than the width of the terminal on which the panel is to be displayed. If the WIDTH keyword is not specified, the default is 80.

EXPAND(*xy*)

The repetition delimiter characters. The delimiters can be used on any line within the panel body to enclose a single character that is repeated to expand the line to the required width. The starting and ending delimiter can be the same character. If no delimiters are specified, or if any line does not contain the delimiters, then the line is expanded to the required width by adding blanks on the right. The delimiter characters *cannot* be specified with a dialog variable.

Before the panel is displayed, it is formatted according to the WIDTH and EXPAND keyword values as if the *expanded format* of the body were originally coded in the panel definition. For example:

```
)BODY  WIDTH(&EDWIDTH) EXPAND(//)
+-- &TITLE ---------------------------------/-/----------
%COMMAND ===>_ZCMD     / /             +SCROLL%===>_SCRL +
+
%EMPLOYEE NUMBER:@EMPLN         / /                     @
```

In the title line, hyphens are repeated to expand the line to the width specified by &EDWIDTH The command field and the employee number field would both be expanded with repeated blanks.

If more than one repetition character appears in a line of the panel body, each of the characters is repeated an equal number of times. For example:

```
)BODY  EXPAND(#@)
TUTORIAL #-@ TITLE OF PAGE #-@ TUTORIAL
```

would become:

```
TUTORIAL ------------ TITLE OF PAGE ------------ TUTORIAL
```

ISPF treats as an error a request to display a panel that is wider than the physical screen or current logical screen for a 3290 terminal. ISPF displays a *box* panel indicating the error. For the 3290, if a panel that is wider than 80 characters is being displayed, and the user attempts to divide the screen vertically (SPLITV command), ISPF denies the request and displays an error message. Remember that the panel is displayed as though the *expanded format* of the body were originally coded in the panel definition. Therefore, be careful when expanding text fields that contain substitutable variables, so that meaningful text is not truncated. For example:

```
)BODY  EXPAND(//)
TUTORIAL /-/ &VAR1 /-/ TUTORIAL
```

would become:

```
TUTORIAL ---------------- &VAR1 ---------------- TUTORIAL
```

Then, if &VAR1 had the value ‘ABCDEFG’ when the screen was displayed, the following line would result:

```
TUTORIAL ---------------- ABCDEFG ---------------- TUTORI
```

To avoid this problem, provide a few blanks at the end of the text string, as follows:

```
TUTORIAL /-/ &VAR1 /-/ TUTORIAL      +
```

Table 12 and Table 13 describe the display width, data expansion width (resulting from EXPAND keyword on the)BODY statement), and the pop-up window width based on various WINDOW/WIDTH keyword combinations.

Table 12. Display in Primary Window

WINDOW/WIDTH Combinations	DISPLAY	EXPANSION
no WINDOW, no WIDTH	WIDTH (def. 80)	WIDTH (def. 80)
WINDOW, no WIDTH	WIDTH (def. 80)	WINDOW value
no WINDOW, WIDTH	WIDTH	WIDTH value
WINDOW <= WIDTH	WIDTH	WINDOW value
WINDOW > WIDTH	ERROR	ERROR

Table 13. Display in Pop-up Window

WINDOW/WIDTH Combinations	DISPLAY	EXPANSION	WINDOW
no WINDOW, no WIDTH	WIDTH (def. 80)	WIDTH (def. 80)	(76, 22)
WINDOW, no WIDTH	WIDTH (def. 80)	WINDOW value	WINDOW (w, d)
no WINDOW, WIDTH	WIDTH	WIDTH value	(76, 22)
WINDOW <= WIDTH	WIDTH	WINDOW value	WINDOW (w, d)
WINDOW > WIDTH	ERROR	ERROR	ERROR

Note: ISPF will issue an error message if you attempt to display a panel in a pop-up window where the WINDOW width value is greater than the width of the underlying panel .

DEFAULT(*def1def2def3***)**
You can use the DEFAULT keyword to specify the characters that define a high-intensity text field, a low-intensity text field, and a high-intensity input field, respectively. The value inside the parentheses must consist of exactly three characters, not enclosed in single quotes and not separated by commas or blanks.

The DEFAULT keyword can also be specified on the)ATTR section statement.

FORMAT(EBCDIC | DBCS | MIX)
The default value for a TYPE(INPUT) and a TYPE(DATAIN) field is FORMAT(EBCDIC). These two default values can be changed by using the)ATTR statement or the)BODY statement. These values, in turn, can be overridden if explicitly specified on a subsequent statement. For example, the net result of the following two statements is FORMAT(DBCS):

```
)BODY FORMAT(MIX)
 $ TYPE(INPUT) FORMAT(DBCS)
```

OUTLINE([L][R][O][U] | BOX | NONE)
The default value for OUTLINE is NONE. The default value for TYPE(INPUT) and TYPE(DATAIN) fields can be specified on the)ATTR or)BODY statement and can be overridden by the OUTLINE keyword. For example:

```
)BODY OUTLINE(U)
  @ TYPE(INPUT) OUTLINE(BOX)
```

A Sample Panel Body Section

The sample panel definition, shown in Figure 66, consists of a panel body followed by an)END control statement. It has no attribute, initialization, reinitialization, or processing sections, and uses the default attribute characters.

This data entry panel has 11 input fields (for example, ZCMD and TYPECHG) indicated with the underscore attribute character. It also has a substitutable variable (EMPSER) within a text field. The first two lines of the panel and the arrows preceding the input fields are all highlighted, as indicated by the percent sign attribute characters. The other text fields are low intensity, as indicated by the plus sign attribute characters.

```
)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 66. Sample Panel Definition

Figure 67 on page 218 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.

```
---------------------------  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 67. Sample Panel—When Displayed

Defining the CCSID Section

The)CCSID section identifies the Coded Character Set Identifier used in the panel definition.

)CCSID [NUMBER(*xxxxx***)]**

where:

NUMBER(*xxxxx***)**
The CCSID of the EXTENDED CODE PAGE as defined by Character Data Representation Architecture. Refer to "Supported CCSIDs" on page 318 for which CCSIDs are supported.

The)CCSID section must be the first section in the panel as illustrated in the following example:

```
)CCSID NUMBER(00037)
)PANEL
)BODY
%---------------------- NAME OF PANEL -------------------------------
%COMMAND ===>__ZCMD
   ⋮

)END
```

If the CCSID section is used, the single-byte text characters in the)BODY,)MODEL, or)AREA section of the panel are translated to the equivalent character (or a period if the character does not exist) in the terminal code page for display. ISPF scans the panel for a text attribute, notes the position, and then scans for a non-text attribute. When the non-text attribute is found, ISPF translates the text between the text attribute and the non-text attribute. Thus you must have one text attribute defined prior to any text you want translated. This translation occurs only if the code page indicated by the CCSID is different from the code page of the terminal.

The ISPF program scans the panel record for a text attribute, notes the position, then scans for a non-text attribute. When the non-text attribute is found, ISPF translates the text found between the text attribute and the non-text attribute. Therefore, you must have one text attribute defined prior to any text you want translated.

All characters in the panel source that are not in the)BODY text must be in the Syntactic Character Set:

- A–Z
- a–z
- 0–9
- + < = > % & * ″ ′
- ( ) , _ - . / : ; ?

Note: Lowercase a–z can be used for any CCSID supported by ISPF except the Japanese (Katakana) Extended CCSID 930.

See "Chapter 10. Extended Code Page Support" on page 313

Defining the END Section

The)END section identifies the end of the the panel definition. It is a required section.

```
)END
```

The definition consists only of the)END statement. Any lines placed after the END statement are ignored.

```
)PANEL
)BODY
%---------------------- NAME OF PANEL -------------------------------
%COMMAND ===>__ZCMD
   ⋮

)END
```

Defining the HELP Section

The)HELP (help) section of the panel definition specifies what help panel, if any, is displayed when help is requested for a particular element defined on the panel. Help can be requested for a field, an action bar choice, or a pull-down choice by including a statement in the source panel definition help section. See "Reference Phrase Help" on page 98 for a discussion on requesting help for reference phrases.

```
)HELP FIELD(field-name) PANEL(help-panel-name)
```

where:

FIELD(*field-name***)**
The name of the source panel element (input selection field, action bar choice, and so on) for which a help panel will be displayed when help is requested for that element.

PANEL(*help-panel-name***)**
The name of the help panel associated with the field.

Specifying the Value for the Field-Name and Help-Panel-Name

When modifying or adding statements to the)HELP section of a new or existing source panel, you must adhere to these rules to prevent unexpected results and errors when the source panel is processed.

The field-name must have the following characteristics:
- 1–8 characters in length
- First, or only, character must be A–Z or a–z
- Remaining characters, if any, must be A–Z, a–z, or 0–9.

Lowercase characters are translated to their uppercase equivalents.

The help-panel-name must have the following characteristics:
- 1–8 characters in length
- The first (or only) character must be A–Z or a–z
- Remaining characters must be A–Z, a–z, or 0–9.

Lowercase characters are translated to their uppercase equivalents.

The action bar choice and pull-down choice elements, have no associated field name. ISPF uses the following convention when generating a field-name value for these panel elements.

Action bar choice field-names will have the following format:

ZABC*xx*

where:

ZABC The field-name prefix

xx The number of the action bar choice

Pull-down choice field-names have the following format:

ZPDC*xxyy*

where:

ZPDC The field-name prefix

xx The number of the action bar choice

yy The number of the pull-down choice within this action bar choice

See “Specifying Action Bar Choices in Panel)BODY Section” on page 163 to determine the numbering sequence ISPF uses for these panel elements.

Defining the Initialization Section

The initialization section specifies the initial processing that is to occur prior to displaying the panel.

```
)INIT
```

It begins with the)INIT header statement and ends with either the)REINIT,)PROC,)HELP, or)END header statement. The number of lines allowed in an)INIT section depends upon the storage size available for panel processing at execution time.

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.

See “Formatting Panel Definition Statements” on page 233 for additional information.

Defining the LIST Section

The)LIST (list) section of the panel definition specifies what list choices appear on your screen. It can be useful if the selection list is displayed in a scrollable area and some of the list choices might not be visible. With the)LIST section coded, all of the choices are built into the list box, drop-down list, or combination box even if some are not immediately visible in the scrollable area.

It is used in conjunction with the attribute keywords DDLIST(name), LISTBOX(name), and COMBO(name). These keywords match the list box attributes to the corresponding list choices found in the)LIST list-name section of the panel.

The)LIST section, if you use it, follows the)PNTS section if one exists. It follows the)HELP section if you have no)PNTS section. If you have neither the)PNTS section nor the)HELP section, the)LIST section follows the)PROC section of your panel definition. The)LIST section contains the following parameters when used with list boxes and drop-down lists:

```
)LIST list-name
VAL(value) CHOICE(value)
```

The)LIST section contains the following parameters when used with combination boxes:

```
)LIST list-name
CHOICE(value)
```

where:

list-name
: The name of the list. It must match a LISTBOX(name), DDLIST(name), or COMBO(name) specified on an input field in the)ATTR section. The name can be 1 to 8 characters long. Alphanumeric characters A-Z, a-z, 0-9, #, $, or @ can be used in the name, but the first character cannot be numeric. Lowercase characters are converted to their uppercase equivalents.

VAL(*value*)
: This parameter is used for list boxes and drop-down lists only. It is *not* used for combination boxes. The value can be a variable or text. It must be 3 characters or less (more than three characters are truncated without warning) and is used as the value placed into the CEF field when the choice is selected.

CHOICE(*value*)
: This parameter is used with list boxes, drop-down lists, and combination boxes. The value can be a variable or text. If the value is a variable, the ampersand (&) must be in the first column following the left parenthesis of the CHOICE keyword. The length of the variable data is limited to 99 single-byte characters. If the varible data is longer than 99 bytes, it will be truncated.

  ```
  CHOICE(&var)
  ```

  If the value is a single word text string it is not necessary to enclose it in single quotation marks.

```
CHOICE(3278)
```

If the value is more than a single word of text, the phrase must be enclosed in single quotation marks.

```
CHOICE('3278 terminal type')
```

Literal values can be split between lines by coding a plus sign (+) as the last character on each line that is to be continued. The plus sign is used as a continuation character.

```
CHOICE('This is an example of a continuation +
of the literal string')
```

The)LIST section must contain a list-name. For list boxes and drop-down lists, it also must contain a VAL and a CHOICE for each of the choices to display in the list. Each entry in the)LIST section must contain the keywords in the following order: VAL(value) CHOICE(value). For combination boxes, the list section must contain a CHOICE(value) for each of the choices to display in the list. The data in the lists is displayed in the order in which you define the choices in the)LIST section.

Defining the Model Section

The)MODEL (model) section is used only for table display panels. It defines how each table row is to be formatted. Because the model section is unique to table display panels, it is discussed in “Defining Table Display Panels” on page 134.

Defining the Panel Section

The)PANEL (panel) section specifies the keylist that will be used for the panel, identifies where the keylist is to be found, controls specific CUA display characteristics of the panel, specifies the image that will be used on the panel, and specifies the row and column placement for the upper left-hand corner of the image.

The IMAGE keyword is used to show images on panels in GUI mode. It is ignored in 3270 mode. ISPF supports image files in the *Graphical Interchange Format* (GIF).

ISPF ships image files in sample library SISPSAMP. The panel ISR@PRIM uses three of the GIF image files: ISPFGIFL, ISPFGIFS, and ISPEXIT.

To use images, store the image files on the host in a partitioned data set and allocate this image data set to ddname ISPILIB before invoking ISPF. For more information about allocating this image library, see the section called Allocating Optional Image ISPF Library in the *ISPF User’s Guide*.

Images can be placed on unused panel space. They should not be positioned on text or panel fields. When an image is requested, ISPF does a query and file transfer to download the image specified to the workstation. The image is downloaded to the *image path*, which the user specifies from the GUI Panel Settings window (Option 0). See the *ISPF User’s Guide* for details. If no image path is specified, ISPF downloads the images to the workstation’s working directory.

Note: When using the ISPF Application Server, which involves applets written in JAVA, the image handling is somewhat different for ISPF. In that case, the image is downloaded into workstation memory, not written to the disk. The

path specified is not used.

```
)PANEL [KEYLIST (keylist-name[,keylist-applid,SHARED])]
[IMAGE (image-name,row,col)]
```

where:

KEYLIST

keylist-name
Required when KEYLIST is specified. The keylist name must have the following characteristics:
- 1–8 characters in length
- First, or only, character must be A–Z or a–z
- Remaining characters, if any, must be A–Z, a–z, or 0–9.

Lowercase characters are translated to their uppercase equivalents.

keylist-applid
Optional. Application ID used at run-time to find the keylist. It has a maximum length of 4 characters, the first of which must be alphabetic. Any remaining characters can be alphabetic or numeric.

SHARED
Optional. When specified, ISPF looks *only* at the shared keylist for the panel. If the user issues the KEYLIST OFF or KEYLIST PRIVATE commands, they have no effect; the keylist in xxxxKEYS table allocated to ISPTLIB is used.

IMAGE

image-name
Required when IMAGE is specified. The image-name identifies the image to be displayed. The image-name can be a variable, which should follow ISPF's variable naming conventions.

Note: ISPF downloads images only in panel initialization processing. Variables for images should only be set in the)INIT section of your panel definition. Variables for images in panel sections other than the)INIT section are not supported unless the image exists on the PC Image Path you specifiy.

row,col
The row and column specifiy the starting position, upper left-hand corner, of the image. The row and column can be numeric or variables. Variables for the row and column should follow ISPF's variable naming conventions. If no row or column is specified, you must code commas as place holders, and the row and column will default to 0,0. For example:

```
IMAGE(imagea,,)
```

It is left to the dialog developer to select appropriate row and column values such that the image will display. ISPF checks for valid numeric values 0-9, but does not check for any limits.

When a *keylist-name* is specified without a *keylist-applid*, ISPF searches for the named keylist in the:

- Keylists for the application ID that is currently running

- ISP applid (if not found in the above applid and the name of the application ID is not ISP).

If the KEYLIST keyword is not found on the)PANEL statement, then the default keylist, ISPKYLST, is used.

Before run-time processing, any keylist (other than the default ISPKYLST) referenced in a panel's definitions must have been created and stored. If you add or modify the)PANEL KEYLIST statement in the definition of an existing source panel, you must create the keylist if it does not already exist. New keylists can be created using ISPF option 0 or using the Dialog Tag Language.

Keylist variables

The following variables are used by the keylist function:

ZKLUSE
: Y or N, this variable indicates whether the keylists are being used for an application ID or not. For example, if KEYLIST OFF has been issued, &ZKLUSE is N. This variable is stored in the application profile. The VPUT service can be used by your application to set this value. Putting a value of N in &ZKLUSE to the profile pool is equivalent to issuing the KEYLIST OFF command. Putting a value of Y in &ZKLUSE to the profile pool is equivalent to issuing the KEYLIST ON command.

ZKLNAME
: contains the name of the keylist of the panel currently being displayed. If no keylist is defined for the panel or the keylist is not being used, &ZKLNAME is blank.

ZKLAPPL
: contains the application ID where the keylist of the panel currently being displayed is found. If no keylist is defined for the panel or the keylist is not being used, &ZKLAPPL is blank.

ZKLTYPE
: P or S, this variable indicates that the keylist for the panel currently being displayed is a private (P) copy defined in the profile table, or a shared (S) copy defined in the xxxxKEYS table (where xxxx is the application ID of the keylist (ZKLAPPL)).

ZKLPRIV
: Y or N, this variable indicates that ISPF is to look at both the private and shared keylist (Y, the default) or that it is to look at only the shared keylists (N). This variable is stored in the application profile. The VPUT service can be used by the application to set this value. Putting a value of N in &ZKLPRIV to the profile pool is equivalent to issuing the KEYLIST SHARED command. Putting a value of Y in &ZKLPRIV to the profile pool is equivalent to issuing the KEYLIST PRIVATE command.

  Note: This variable shows and determines where ISPF *looks* for a keylist. &ZKLTYPE is a non-modifiable variable that shows where ISPF found the keylist.

CUA Display Characteristics

The)PANEL section controls specific CUA display characteristics of a panel. Specifying the)PANEL statement in the panel source definition affects the same display characteristics controlled by selecting the Panel display CUA mode option on the ISPF Settings panel. See the *ISPF User's Guide* for more information.

The)PANEL statement controls the following CUA display characteristics:

- Display and placement of the command line and long message text
- Building and display of the named keylist in the Function Key Area (FKA)
- Handling of undefined or null function key definitions
- Execution of the CANCEL and EXIT commands
- Setting of three system control variables that relate to the position of the cursor after panel display.

Command Lines and Long Messages

When the)PANEL section is used, the ISPF default command line placement is at the bottom of the panel (above the function key area, if it is displayed). Long messages are displayed above the command line. To override the ISPF default, go to the ISPF Settings panel and specify Command line placement - ASIS. This setting places the command line and long message as they are specified in your panel definition (usually at the top of the panel). See *ISPF User's Guide* Changes to the)BODY section also affect command line and long message placement. The ASIS keyword on the)BODY section overrides ISPF defaults. The WINDOW keyword also affects the displaying of the command line and long messages. See "Defining the Body Section" on page 212.

You can specify to not have a command line by including the keyword CMD() with no value on the)BODY statement. This is valid only for displaying panels with the DISPLAY service. In this case, the default position of the long message is at the bottom of the panel above the FKA, if it is displayed. Panels (tables) displayed with the TBDISPL service must specify a command area either by coding a CMD() with a value or by coding the system control variable ZCMD in the panel body.

Because the)PANEL statement affects the same display characteristics as if you had selected the Panel display CUA mode option on the ISPF Settings panel, the color and intensity of the short and long messages is affected by the presence of the)PANEL statement. If you specify the LMSG or SMSG keywords on the)BODY statement, you control the color and intensity in which both the short and long messages are displayed, regardless of CUA mode or the presence of a)PANEL statement. Table 19 on page 297 illustrates default message placement.

Keylist Building and Display

The format and display of the named keylist or an ISPF default keylist for a panel containing the)PANEL statement is as follows:

- The maximum number of function keys that can be formatted on each line is displayed.
- Each displayed function key definition appears as *Fnn=label* or *Fn=label* (where *nn* or *n* is the numeric value of the function key).

ISPF attempts to build the FKA with the named keylist or an ISPF default keylist. However, the display of the keylist in the FKA area depends upon the settings of the FKA or PFSHOW commands and the keylist format (SHORT or LONG) specified for the function key definition. The number and set of function keys displayed also varies.

Note: The system control variable ZPFCTL setting is ignored for panel source definitions that contain the)PANEL statement.

Undefined or Null Function Keys

When you press an undefined or null function key, ISPF displays an error message.

CANCEL and EXIT Execution

When the CANCEL or EXIT commands (specified on a function key or entered in a command field) are processed, ISPF returns the entered command in the system control variable ZVERB and sets a return code of 8 from the display service.

If the panel contains an action bar and the cursor is on the action bar, CANCEL moves the cursor to the panel body. ZVERB is not updated.

Setting System Control Variables

When panels with a)PANEL section specified are displayed, ISPF sets the following system control variables:

ZCURFLD
: Name of the field (or list column) containing the cursor when the user exits the panel.

ZCURPOS
: Position of the cursor within the field specified by ZCURFLD when the user exits the panel.

ZCURINX
: Current row number of the table row containing the cursor.

These system variables are stored in the function pool as output variables.

Defining the Point-and-Shoot Section

The)PNTS (point-and-shoot) section of a panel definition specifies what fields, if any, are point-and-shoot fields. Input and/or output fields are specified as point-and-shoot fields by the use of the attribute keyword, PAS(ON). Text fields are specified as point-and-shoot fields by the attribute type keyword, TYPE(PS). For each panel field specified as a point-and-shoot field, there must be a corresponding entry in the)PNTS section. If a field specified as a point-and-shoot field has no corresponding entry in the)PNTS section, no action will be taken if the point-and-shoot field is selected. The examples below show a)PNTS section point-and-shoot phrase definition for input/output fields and for text fields.

Note: You can use option 0 (Settings) to set the tab key to move the cursor point-and-shoot fields. This changes output fields to input fields, but data is not altered. However, if a variable is used on an output field that is changed to an input field by the tab to point-and-shoot option, and the variable is VDEFINEd to the application, the variable will be truncated. In this case, the application developer should have a temporary panel variable.

GUI Mode: If you are running in GUI mode, fields designated as point-and-shoot output and text fields will appear as pushbuttons. Point-and-shoot input fields will appear as selection fields.

Large pushbuttons are point-and-shoot output or text fields which display with a depth greater than one. Large pushbuttons are built by coding the DEPTH keyword on the point-and-shoot statement in the)PNTS panel section.

In GUI mode, images can be displayed on these pushbuttons. The keywords that provide the support for images are DEPTH, IMAGE, IMAGEP, TEXT, and PLACE. These keywords are used in GUI mode and ignored in 3270 mode.

Although you can define images on point-and-shoot output fields and point-and-shoot text fields, if you define an image for a point-and-shoot output field, the field cannot be a Z-variable in the panel body.

You can specify where to place an image on a large pushbutton. It can be above or below the pushbutton text, or to the left or right of the pushbutton text. When you specify the placement of the image to be above or below the text, the image is always centered relative to the text.

ISPF ships sample images in sample library SISPSAMP. The panel ISR@PRIM uses three of the GIF image files: ISPFGIFL, ISPFGIFS, and ISPEXIT.

To use images, store the image files on the host in a partitioned data set and allocate this image data set to ddname ISPILIB before invoking ISPF. For more information about allocating this image library see the section called Allocating Optional Image ISPF Library in the *ISPF User's Guide*.

When an image is requested, ISPF does a query and file transfer to download the image specified to the workstation. The image is downloaded to the *image path* that you specify from the GUI Panel Settings window (Option 0). See the *ISPF User's Guide* for details. If no image path is specified, ISPF downloads the image to the workstation's working directory.

Note: When using the ISPF Application Server, which involves applets written in JAVA, the image handling is somewhat different for ISPF. In that case, the image is downloaded into workstation memory, not written to the disk. The path specified is not used.

```
)PNTS
FIELD(field_name|ZPSxxyyy) VAR(value) VAL(value)
[DEPTH(depth)] [IMAGE(image-name)] [IMAGEP(image-name)]
[TEXT('text')] [PLACE(a,b,l,r)]
```

Note: Each entry in the)PNTS section must contain the keywords in the following order: FIELD, VAR, VAL, [DEPTH]. When defining large pushbuttons or large pushbuttons with images the DEPTH keyword must immediately follow the VAL keyword on the)PNTS entry statement. The remaining keywords, [IMAGE] [IMAGEP], [TEXT], [PLACE], follow the DEPTH keyword. Both the DEPTH keyword and the TEXT keyword must be coded on the PNTS entry for point-and-shoot text fields if you are defining a large pushbutton, or an image for the field.

where:

FIELD(*field_name***|ZPS***xxyyy***)**
The name of the field on the panel that this statement controls.

For point-and-shoot input/output fields, the format is:

FIELD(*field_name***)**
where:

field_name
The name of the field on the panel that this statement controls.

For point-and-shoot text fields, the format is:

FIELD(ZPS*xxyyy***)**
where:

xx 00 for a point-and-shoot field defined in the)BODY section and 01 to 99 for the number of the scrollable area in which the point-and-shoot text field is defined.

Each scrollable area is assigned a sequential number based on its relative position within the panel body. The scrollable area closest to the upper-left corner of the panel body is assigned number 01. Each additional scrollable area, scanning left to right, top to bottom, is assigned the next sequential number. A maximum of 99 scrollable areas in any given panel can contain point-and-shoot text fields.

yyy 001 to 999 for the relative number of the point-and-shoot text field within the panel body or within a particular scrollable area.

A point-and-shoot text field can wrap around multiple terminal lines in panels that are not displayed in a window. A point-and-shoot text field that logically wraps in a pop-up window requires the beginning of each wrapped line to contain a PS field attribute and an entry must exist in the)PNTS section for each wrapped line. This is also true for panels containing the WINDOW() keyword that are not displayed in a pop-up window. The additional)PNTS section entries should result in the same action as the first line of the wrapped text field.

VAR(*value***)**
The name, or a variable containing the name, of the variable to be set when the field named in this)PNTS statement is selected. If the value is a variable, an ampersand (&) must be in the first column following the left parenthesis of the VAR keyword, and it must follow dialog variable naming conventions. If the value is a variable it is limited to the leading ampersand plus 7 characters.

VAL(*value***)**
The value assigned to the variable named in this statement. The value can be a variable or text. If the value is a variable, an ampersand (&) must be in the first column following the left parenthesis of the VAL keyword. The length of the variable data is limited to 255 single-byte characters. If the variable data is longer than 255 bytes, it is truncated. If the value is a variable it is limited to the leading ampersand plus 7 characters.

```
VAL(&var)
```

If the value is a single word text string it is not necessary to enclose it in single quotation marks.

```
VAL(Batch)
```

If the value is more than a single word of text, the phrase must be enclosed in single quotation marks.

```
VAL('List of products')
```

Literal values can be split between lines by coding a plus sign (+) as the last character on each line that is to be continued. The plus sign is used as a continuation character.

```
VAL('This is an example of a continuation +
of the literal string')
```

DEPTH(*depth***)**
The *depth* of the point-and-shoot field (pushbutton). The DEPTH keyword is required and must be specified immediately following the VAL keyword on the)PNTS section statement. ISPF allows *depth* values from zero to sixty-two (0-62). Sixty-two is the maximum screen depth. It is up to the dialog developer to define the *depth* such that other items on the panel body will not be overlaid by the point-and-shoot field (pushbutton). If *depth* is specified as 0, the default depth of two (2) is used. The *depth* can be a variable, whose value is from 0-62.

IMAGE(*image-name***)**
The *image-name* identifies the image to be displayed. The image-name is used when the images are stored on the host in a partitioned data set, with a data set definition of ISPILIB. The *image-name* must follow TSO data set member naming conventions. The *image-name* can be a variable, which should follow ISPF's variable naming conventions.

IMAGEP(*image-name***)**
The *image-name* identifies the image to be displayed, when the point-and-shoot pushbutton is pressed. For example, a pushbutton might normally display a closed door image, but when you press the button, an opened door image could appear. The *image-name* is used when the images are stored on the host in a partitioned data set, with a data set definition of ISPILIB. The *image-name* must follow TSO data set member naming conventions. The *image-name* can be a variable, which should follow ISPF's variable naming conventions.

Note: ISPF downloads images only in panel initialization processing. Variables for images should only be set in the)INIT section of your panel definition. Variables for images in panel sections other than the)INIT section are not supported unless the image exists on the PC Image Path you specifiy.

TEXT(*'text'***)**
The TEXT keyword is required for point-and-shoot text fields. The *text* ties the point-and-shoot text field defined in the panel body with its point-and-shoot entry in the)PNTS section. The *text* must match the text for the point-and-shoot field in the body. If the text in the body contains variables, the *text* of the TEXT keyword must allow for the possible expansion once the variable has been substituted, just as the point-and-shoot text field in the body should. If the *text* consists of more than a single word of text, the phrase must be enclosed in single quotation marks.

PLACE(*a* | *b* | *l* | *r***)**
The values *a* (above), *b* (below), *l* (left), and *r* (right) specify the position of the image relative to the pushbutton text. The PLACE keyword is optional. If not specified, the default image position is above (*a*) the text in the pushbutton. The text for pushbuttons is always centered within the pushbutton. The text for a pushbutton does not wrap, thus one line of text is the maximum. The image is placed either above or below the text, or to

the left or the right of the text. It is up to the dialog developer to allow for space for the pushbutton text and the image. The value for PLACE can be a variable whose value is ***a, b, l,***or ***r***.

Example:

```
)PANEL
)ATTR
  $ TYPE(PIN)
  } TYPE(PS)
  + TYPE(NT)
  | AREA(SCRL) EXTEND(ON)
  ! TYPE(OUTPUT) PAS(ON) COLOR(RED)
  * TYPE(OUTPUT) PAS(ON) COLOR(BLUE)
  @ TYPE(TEXT) INTENS(LOW) COLOR(RED) PAD(NULLS)
  ø TYPE(TEXT) INTENS(LOW) COLOR(BLUE) PAD(NULLS)
)BODY WINDOW(60,23)
$
%COMMAND ===>_ZCMD
$
$  Press }DEFAULTS$to reinstate defaults
$
+
|S1                                                |
)AREA S1
+                                      +
+                                      +
+     øBLUE  . . . .*BLUE1             +
+     @RED . . . . .!RED1              +
)INIT
 .CURSOR = blue1
    &blue1    = ' '
)PROC
 REFRESH(*)
)PNTS
  FIELD(BLUE1) VAR(RED1) VAL(RED)
  FIELD(ZPS00001) VAR(BLUE1) VAL(DEFAULT)
)END
```

Defining the Processing 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)HELP or)END statement. The number of lines allowed in a)PROC section depends upon the storage size available.

)PROC

A statement can be continued over as many lines as necessary as long as it is broken at the end of a word, or a continuation symbol (+) is used within a literal. In menus, the processing section is required and must be in a special format, as described in “Defining Menus” on page 115.

See “Formatting Panel Definition Statements” on page 233 for additional information.

Defining the Reinitialization Section

The reinitialization section specifies processing that is to occur prior to redisplay of a panel. If it is present, it follows the initialization section and precedes the processing section.

```
)REINIT
```

Panel redisplay occurs in either of the following situations:

- Redisplay occurs automatically after the)PROC section has been processed if the .MSG control variable is nonblank and the user has not requested END or RETURN. The .MSG control variable is set automatically if a translation or verification error occurs. It can also be set explicitly by use of an assignment statement in the)PROC section.
- Redisplay occurs if a dialog function invokes the DISPLAY or TBDISPL service with no panel name specified (a blank).

  Note: See *ISPF Services Guide* under the description of the TBDISPL for a description of how redisplay processing for the TBDISPL service differs from that for the DISPLAY service described here.

Processing of the)INIT section is intentionally bypassed when a redisplay occurs. Instead, the)REINIT section is processed. The automatic fetching of variables to be displayed in the panel body is also bypassed on a redisplay. Thus, the panel is redisplayed exactly as the user last saw it, except:

- An error message can appear on a redisplay.
- Field attribute overrides, assignment statements, or REFRESH statements can be used, as noted in the following paragraph.
- A scrollable area can be scrolled to position the cursor or to verify failure.

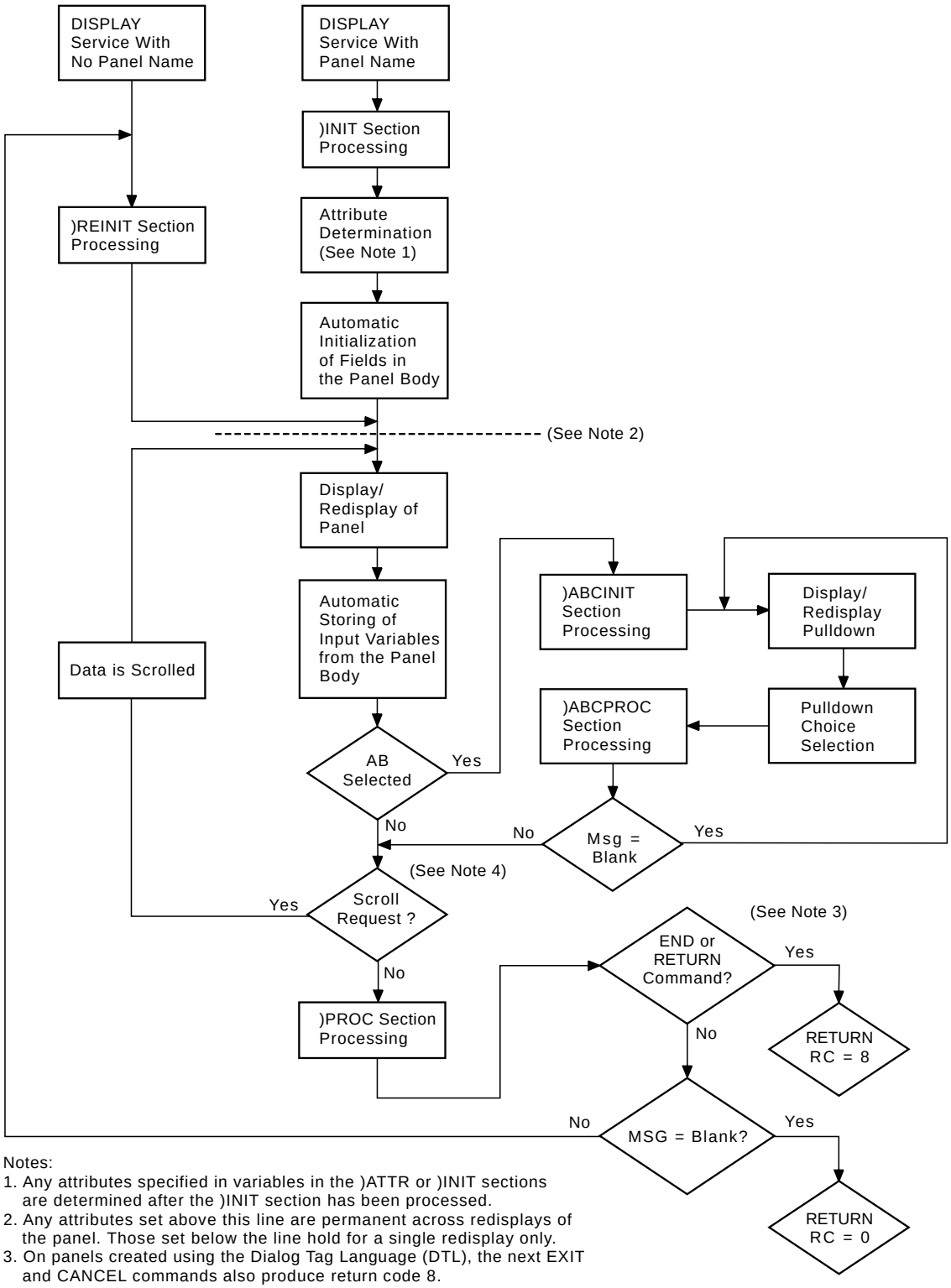
Typically, a)REINIT section contains:

- Field attribute overrides, specified by the .ATTR control variable
- Changes to displayed panel fields, specified by assignment statements and the REFRESH statement.

See “Formatting Panel Definition Statements” on page 233 for additional information.

Figure 68 on page 232 shows panel processing and the point at which attribute settings can be modified for redisplay of a panel.

)REINIT Section



Notes:
1. Any attributes specified in variables in the)ATTR or)INIT sections are determined after the)INIT section has been processed.
2. Any attributes set above this line are permanent across redisplays of the panel. Those set below the line hold for a single redisplay only.
3. On panels created using the Dialog Tag Language (DTL), the next EXIT and CANCEL commands also produce return code 8.
4. The Yes branch is taken if there is a scroll request, a scrollable area is defined for the panel, and the cursor is within a scrollable area.

Figure 68. Panel Processing

Formatting Panel Definition Statements

This section provides reference information for the following panel definition statements:

- Assignment – page 234

  Note: You can use eight built-in functions in an assignment statement:

  - TRUNC (truncate)
  - TRANS (translate)
  - PFK (function key)
  - LVLINE (last visible line)
  - ADDSOSI (add shift-out character)
  - DELSOSI (delete shift-out character)
  - ONEBYTE (convert to a one-byte code)
  - TWOBYTE (convert to a two-byte code)
- ELSE – page 239
- EXIT – page 241
- GOTO – page 241
- IF – page 243
- PANEXIT – page 247
- REFRESH – page 252
- TOG – page 253
- VEDIT – page 255
- VER – page 255
- VGET – page 264
- VPUT – page 266

The following types of data references can appear within panel section statements:

Dialog variable
A name preceded by an ampersand (&)

Control variable
A name preceded by a period (.)

Literal value
A character string not beginning with an ampersand or period. A literal value can be enclosed in single quotes (''). It *must* be enclosed in single quotes if it begins with a single ampersand or a period, or if it contains any of the following special characters:

```
Blank < ( + | ) ; ¬ - , > : =
```

A literal can 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. Trailing blanks are removed from the variable before the replacement. You can use a double ampersand to specify a literal character string starting with, or containing, an ampersand.

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

The Assignment Statement

Assignment statements can be used in the)INIT section to set the contents of dialog variables prior to the automatic initialization of variables in the panel body. Also, they can be used in the)REINIT section prior to redisplay of the panel body. Assignment statements can 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.

```
variable = value
```

where:

value
Specifies the contents of the dialog variable.

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 a single quote. See Chapter 5. Panel Definition Statement Guide for information on 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.

The TRUNC Built-In Function

The TRUNC built-in function can occur on the right side of an assignment statement to cause truncation.

```
variable = TRUNC (variable,value)
```

where:

variable
(Inside the parentheses). Specifies the variable to be truncated.

value
A numeric quantity indicating the length of the truncated result or any special character indicating truncation at the first occurrence of that character.

Examples:

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

In the first example, the contents of variable XYZ are truncated to a length of 3 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 is not stored, nor is it retained in the assignment variable's value. For example:

```
)PROC
  &AAA = TRUNC (&ZCMD, '.')
  &BBB = .TRAIL
```

If variable ZCMD contains `9.4.6`, variable AAA contains `9`. The .TRAIL control variable and variable BBB contain `4.6`. The value of ZCMD remains as `9.4.6`.

Because the control variable .TRAIL is set to blanks before the truncation function is performed, it should not be specified as the truncation variable in the TRUNC statement. For example: &ERROR = TRUNC(.TRAIL,1) would always result in &ERROR being set to blank.

For the TRUNC built-in function, the source and destination variables can be the same. Figure 69 on page 237 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 you to change the valid options for TYPECHG by simply typeing over the first character.

The TRUNC and TRANS built-in functions can 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.

The TRANS Built-In Function

The TRANS built-in function can occur on the right side of an assignment statement to cause translation.

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

where:

variable
(Inside the parentheses). Specifies the variable to be translated.

value,value
Paired values. The maximum number of paired values allowed is 126. The first value in each pair indicates a possible value of the variable, and the second indicates the translated result.

Example:

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

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

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

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

In the first example, if the current value of MOD does not match any of the listed values, a question mark 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.

MSG=value
A message ID. Another option for the anything-else condition is to cause a message to be displayed to the user Typically, this technique is used in the processing section of the panel definition.

Example:

```
&DISP = TRANS (&D 1,SHR 2,NEW 3,MOD MSG=PQRS001)
```

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 PQRS001 is displayed. Message PQRS001 can be an error message indicating that the user has entered an invalid option.

For the TRANS built-in function, the source and destination variables can be the same. Figure 69 on page 237 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 you to change the valid options for TYPECHG by simply typing over the first character.

The TRANS and TRUNC built-in functions can 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 69. Sample Panel Definition with TRANS and TRUNC

The PFK Built-In Function

The PFK built-in function provides function key assignment information by command or key number.

```
variable = PFK(value)
```

where:

value
Either a command or a key number.

Example:

```
&X = PFK (HELP)
&Y = PFK (2)
```

In the first example, the first function key that is assigned to the HELP command is returned in variable X as a character string `PFnn`, where *nn* is the function key number. If CUA mode is set, or the panel has an active keylist, the character string is `Fnn`, where *nn* is the function key number. If the HELP command is not assigned to a function key, a blank value is returned.

In scanning the current function key definitions, the *primary* keys are scanned first, then the *secondary* keys. If KEYLIST OFF has been issued, ISPF searches the ZPF variables. On a 24-key terminal, for example, if both function keys 13 and 1 are assigned to HELP, the function returns `F13`.

In the second example, the command assigned to F2 is returned in variable Y. If no command is assigned to the key requested, a blank value is returned.

The LVLINE Built-In Function

The LVLINE built-in function (used on an assignment statement in the)INIT,)REINIT, or)PROC section) provides the line number of the last visible line within a graphic or dynamic area of the currently displayed panel.

```
variable = LVLINE(value)
```

where:

value
Name of the GRAPHIC or DYNAMIC area. In split-screen mode, this value could be less than the number of lines defined in the area.

This built-in function provides the line number of the last line within a graphic or dynamic area that is visible to the user on the currently displayed panel. The *value* parameter is the name of the graphic or dynamic area. In split-screen mode, this value could be less than the number of lines defined in the area. If the area is defined within a scrollable area, the number returned is the last visible line when the user submitted the panel, even if the user could have scrolled to see more.

Note: When coding the command line after the dynamic area on a non-TBDISPL panel, ISPF might not be able to calculate the LVLINE value correcty based on the location of the command line following the dynamic area, the number of lines after the dynamic area, the function key settings, SPLIT or SPLITV command processing, or other ISPF commands that affect the screen size displayed. To achieve the correct LVLINE value with the command line displayed at the bottom of the ISPF dynamic area panel, the command line will have to be coded above the dynamic area on the panel, ZPLACE set to BOTTOM, and CUA mode set to YES.

Example:

```
&L1 = LVLINE(AREA1)
```

The ADDSOSI and DELSOSI Built-In Functions

These built-in functions are used to add to or delete from a value-string the shift-out and shift-in characters that mark the start and end of a DBCS field, without changing the value of the input string.

```
variable = ADDSOSI(variable name)
variable = DELSOSI(variable name|DBCS literal)
```

where:

variable name
Name of the variable that the function will process.

Examples:

```
&VAR2 = ADDSOSI(&VAR1)
&VAR2; = DELSOSI('[DBDBDBDB]')
```

The bracket characters [and] represent the shift-out and shift-in characters.

The target variable must not contain mixed (DBCS/EBCDIC) data. Only variables, not literals, can be specified with the ADDSOSI function. Variables or literals can be specified with the DELSOSI function. An odd input-value length is not permitted for either function. The input-value length does not include trailing blanks or nulls. Nested built-in functions are not allowed on the DELSOSI function. The ADDSOSI function allows nesting of the TWOBYTE built-in function, described below.

Example:

```
&VARB = ADDSOSI(TWOBYTE(&VARA))
```

Variable VARA is converted to a 2-byte character code and shift-out and shift-in characters are added to the character string. Then, variable VARB is set to the resulting value.

The ONEBYTE and TWOBYTE Built-In Functions

The ONEBYTE function is used to convert a variable from a two-byte character code to the corresponding one-byte code without changing the value of the variable. The TWOBYTE function is used to convert a variable from a one-byte character code to the corresponding two-byte code without changing the value of the variable.

```
variable = ONEBYTE(variable name)
variable = TWOBYTE(variable name)
```

where:

variable name
Name of the variable the function will process.

Examples:

```
&VARA = ONEBYTE(&VARB)
&VARA = TWOBYTE(&VARB)
```

The variable being converted must not contain mixed (DBCS/EBCDIC) data. Only variables, not literals, can be converted. An odd input value length is permitted for the TWOBYTE function, but is not permitted for the ONEBYTE function. The input value length does not include trailing blanks or nulls. Literals cannot be used as input parameters for either function. Nested built-in functions are not allowed on the TWOBYTE function. The ONEBYTE function allows nesting of the DELSOSI built-in function.

Example:

```
&VARB = ONEBYTE(DELSOSI(&VARA))
```

The ELSE Statement

The ELSE statement specifies that alternate processing is to take place when the conditions of the matching IF statement are not satisfied.

```
ELSE
```

The ELSE statement has no parameters. The ELSE statement must be column-aligned with the matching IF statement. Only one ELSE statement is allowed on the same line, even though each can align with a prior IF statement.

ELSE Statement

You can nest IF statements within ELSE statements. The only limitation on the number of nested IF statements is the maximum number of columns available for indented statements due to the panel record length.

The ELSE statement is indentation sensitive. If the conditional expression is true, the ELSE statement that is column-aligned with the IF plus all statements to the right of that column are skipped. Processing continues with the next statement that begins in the same column as the ELSE or in a column to the left of the ELSE.

An example of using the ELSE statement:

```
IF (&DOW = UP)
  &ACTION = SELL
ELSE
  IF (&DOW = DOWN)
    &ACTION = BUY
  ELSE
    &ACTION = HOLD
&DOW = &BEAR
```

In this example, if the value of &DOW is UP, variable &ACTION is set to SELL and processing continues at the statement &DOW = &BEAR The indented processing statements following the first ELSE statement execute if variable &DOW does not have a value of UP. The assignment statement, &ACTION = HOLD, executes only if the value of &DOW is not UP or DOWN.

Figure 70 on page 241 shows a sample panel definition with an IF/ELSE statement pair. The current value of variable PHA is tested for the local area code, 919. If the value of PHA is 919, variable RATE is set to the value of variable &LOCAL If the value of PHA is not 919, variable RATE is set to the value of variable &LONGD

```
)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)
  IF (&PHA = '919')
    &RATE = &LOCAL
  ELSE
    &RATE = &LONGD

)END
```

Figure 70. Sample Panel Definition with IF and ELSE Statement

EXIT and GOTO Statements

Nested IF/ELSE statements can easily become complex, especially since the IF statement is indentation sensitive. The GOTO and EXIT statements allow you to avoid these complexities and achieve enhanced performance during panel processing. You can transfer control back to the user as soon as processing errors are detected.

The GOTO and the EXIT statements are both allowed in the)INIT,)REINIT,)PROC,)ABCINIT, and)ABCPROC sections of the panel source definitions.

EXIT Statement

```
EXIT
```

The EXIT statement has no parameters. When an EXIT statement is encountered during panel processing, ISPF halts processing of the section in which the statement was found and bypasses all remaining statements in that section. Further processing of the panel continues normally.

- Example 1: Simple GOTO/EXIT

```
)PROC
  IF (&CUSTNAME = ' ')
    GOTO NAMERR
    IF (&CUSTNUM = ' ')
      .msg=xxxxx     /* message indicating number is required   */
      EXIT           /* exit )PROC section                      */
```

```
VER (&CUSTNAME,ALPHA,msg=xxxxx)   /* messages specific to     */
VER (&CUSTNUM,NUM,msg=xxxxx)      /* data type - alpha or num */
GOTO NXTSECT
NAMERR:
  .msg=xxxxx       /* message indicating name must be entered */
  EXIT             /* exit )PROC section                      */
NXTSECT:
  zero, one, or more statements
```

In this example, the VER statements are skipped if no values are entered for the CUSTNAME or CUSTNUM variable fields. Processing for the)PROC is halted after the *.msg* variable is set.

- Example 2: Multiple GOTOs

```
)INIT
  &var2 = ' '
  IF (&newcust = ' ')
    GOTO BYPASS
  IF (&newcust = 'renew')
    &var2 = 1
    GOTO NXTCHK1
  IF (&newcust = 'initial')
    &var2 = 2
    GOTO NXTCHK1
  ELSE
    GOTO BYPASS
  NXTCHK1:
  IF (&var2 = 1)
    &var3 = 1
    &var4 = 0
    GOTO NXTSECT
  ELSE
    &var4 = 1
    &var3 = 0
    GOTO NXTSECT
  BYPASS:
    &var3 = 0
    &var4 = 0
  NXTSECT:
    zero, one, or more statements
```

Assuming that the variable NEWCUST was entered and verified to contain one of the two values on a previous panel display, this example illustrates that certain fields on the panel currently being processed will or will not be set depending on the value of NEWCUST.

- Example 3: GOTO Label within IF/ELSE

```
)INIT
  IF (&var1 = ' ')
    GOTO BYPASS
  IF (&var2 = 1)
    &var5 = 1
    &var6 = 0
    BYPASS:
    &var7 = 1
  ELSE
    zero, one, or more statements
```

If variable *var1* is blank, control is transferred to the label BYPASS. Variables *var5* and *var6* are not set and processing will continue as if the IF statement were TRUE. Variable *var7* will be set to 1. The ELSE branch is not executed.

GOTO Statement

```
GOTO  label
```

where:

label
Literal value of the label to which you will branch. The label:
- Must be from 1 to 8 characters in length
- Must begin with an alphabetic character (A–Z, a–z)
- May contain any other alphameric character (A–Z, a–z, 0–9).

The literal value of the label used must be followed by a colon when it appears by itself as a label. For example:
label:

ISPF translates the value for the label to uppercase before it is processed.

There are no indentation restrictions on a GOTO and its corresponding label. They may be at different indentation levels.

ISPF processes the GOTO statement as follows:
- ISPF assumes that transfer of control to the named label is downward.
- ISPF continues processing with the next sequential statement after the first occurrence of the named label.
- ISPF ignores duplicate labels.
- ISPF may transfer control within the IF or ELSE branch of an IF/ELSE statement. If the label is within the IF branch, processing continues with the next statement following the label as if the IF were true. If the label is within the ELSE branch, processing continues with the next statement following the label as if the IF were false.

ISPF issues a severe error message if it does not find a matching label below the GOTO statement and within the same section in which the GOTO statement is coded. The label need not be on a line by itself.

The IF Statement

The IF statement is a valuable tool used to verify a conditional expression. The conditional expression can be as basic as testing the value of a variable or can be expanded to use VER statement constructs and Boolean capabilities. This section first defines the complete syntax of the IF statement. More detailed sections follow to describe:
- Basic IF value testing
- IF statement with VER constructs
- IF statement with Boolean operators

IF statements are valid in the)INIT,)REINIT,)PROC,)ABCINIT, and)ABCPROC panel sections.

```
IF (conditional-expression [ boolean-operator conditional-expression]..)
   ⋮
 [ELSE]
   ⋮
```

where:

conditional-expression
Is either:

Basic value test expression:
variable operator value[,*value*...]

VER statement construct coded without the MSG= parameter:
VER (*variable* [, NONBLANK], *keyword* [, *value1, value2*,...])

Boolean-operator
The character symbol **&** or characters **AND** (AND Boolean operator) or the character symbol **|** or characters **OR** (OR Boolean operator).

ELSE
The optional statement that specifies alternate processing if the IF condition is not satisfied.

Basic IF Value Testing

```
IF (variable operator value [,value ...])
   ⋮
 [ELSE]
   ⋮
```

The parentheses in the syntax contain a conditional expression, in which the operator is expressed in either uppercase character symbols, such as EQ, or in special symbols, such as =. These symbols can be any of:

= or EQ	(equal to)
¬= or NE	(not equal)
> or GT	(greater than)
< or LT	(less than)
>= or GE	(greater than or equal)
<= or LE	(less than or equal)
¬> or NG	(not greater than)
¬< or NL	(not less than).

You can specify comparison against up to 255 values for the EQ (=) and NE (¬=) operators. For the remaining operators, you can specify comparison against only one value.

If you use a character symbol operator, it must be separated from the variable name and comparison value by one or more blanks. For example:

```
IF (&ABC EQ 365)
```

Separation of a special symbol operator from the variable name and comparison value is optional.

```
IF (&ABC = 365)  is the same as  IF (&ABC=365)
```

A compound symbol operator, such as <= or NG, must not contain intervening blanks. For example:

```
<=  cannot be  < =
```

In determining whether the criteria of a conditional expression are met, ISPF uses a numeric compare if the value of the variable and the value being compared are whole numbers between −2147483648 and +2147483647. Thus, if &A is set to +1,

the expression IF (&A=1) is evaluated as being true, using the numeric compare. If the value of the variable and the value being compared are not whole numbers between –2147483648 and +2147483647, ISPF uses a character compare, using the EBCDIC collating sequence to evaluate the IF expression. For both numeric and character compares, trailing blanks are ignored.

Examples of basic value testing:

- IF (&DSN = ‘’) — True if variable DSN is null or contains blanks.
- IF (&OPT EQ 1,2,5) — True if variable OPT contains any of the literal values 1, 2, or 5.
- IF (&A GE &B) — True if the value of variable A is greater than or equal to the value of variable B.
- IF (&A ¬= AAA,BBB) — True if 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 a column-aligned ELSE statement, if one exists, or 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.

IF Statement with VER Constructs

The conditional expression on the IF statement now includes VER statement constructs with one exception: the MSG= parameter is not allowed. The IF conditional-expression evaluates to TRUE (1) for successful verifications and to FALSE (0) for failing verifications. See “The VER Statement” on page 255 for complete explanation of the VER statement. An example of using VER statements with IF statements:

```
IF (VER (valid keyword parameters and values))
   ⋮

ELSE
  .MSG = nld122
  IF (VER (valid keyword parameters and values))
   ⋮
```

The VER statement can be split over more than one line, but the VER statement and the left parenthesis of its keyword parameters must be on the same line. The following example is invalid:

```
IF (VER
 (valid keyword parameters and values))
   ⋮
```

IF Statement and Boolean Operators

You can combine two or more conditional expressions on the IF statement. ISPF evaluates the conditional expressions on the IF statement from left to right, starting with the first expression and proceeding to the next and subsequent expressions on the IF statement until processing is complete.

The use of the **AND** Boolean operator takes precedence over the **OR** Boolean operator as shown in the following examples.

The number of conditional expressions you can specify on the IF statement is limited to 255.

The accepted symbols for the Boolean operators are:

- **&** or **AND** (AND Boolean operator)

  AND processing returns a TRUE result for the IF statement only if all the conditional expressions evaluate as TRUE.
- | or **OR** (OR Boolean operator)

  OR processing returns a TRUE result for the IF statement if any of the conditional expressions evaluate as TRUE. Also, for an IF statement to be evaluated as FALSE, all conditional expressions must be evaluated as FALSE.

  The Boolean operators must be separated by a preceding and following blank or blanks.

Examples of Boolean operators in the IF statement:

- Example 1: Comparison of two expressions using different Boolean operators in two separate IF statements.

  ```
  IF (VER (&vara,NB,ALPHA) & VER (&varb,NB,ALPHA))
     ⋮

  ELSE
    IF (&varc = 123 OR VER (&vard,NB,NUM))
     ⋮
  ```

  The first IF statement will be successful only if both VER expressions are satisfied, while the IF statement under the ELSE will be successful if either of the expressions on the IF statement are satisfied.
- Example 2: Comparison of three expressions using the AND Boolean operator in the same IF statement, with additional OR Boolean operators.

  ```
  IF (VER (&vara,NB,ALPHA) & VER (&varb,NB,ALPHA) &
      &varc = abc,xyz | &vard = 123 | &vard = 456)
     ⋮

  ELSE
    .msg = nld123
  ```

  The IF statement will be successful if the comparisons of the first three expressions evaluate to TRUE, or if expressions four or five evaluate to TRUE.
- Example 3: Comparison of two pairs of expressions using the AND Boolean operator combined on the same IF statement by the OR Boolean operator.

  ```
  IF (VER (&vara,NB,ALPHA) AND &varb = abc OR
      VER (&vara,NB,ALPHA) AND &varb = xyz)
     ⋮
  ```

```
ELSE
  .msg = nld124
  .attr (vara) = 'color(yellow)'
  .attr (varb) = 'color(yellow)'
```

Either of the pairs of expressions must evaluate to TRUE to achieve a successful IF statement.

- Example 4: Comparison of three expressions showing that the AND operator has precedence.

```
IF (Expression-1 OR Expression-2 AND Expression-3)
   ⋮

ELSE
  .msg = nld125
```

Because the IF statement AND Boolean operator has precedence over the IF statement OR Boolean operator, the specifying an IF statement similar to the one above might not give you the results you expected.

If you expected the previous statement to be evaluated like this:

```
IF ( (expression1 OR expression2) AND expression3)
```

You would need to write either two separate IF statements:

```
IF (Expression-1 OR Expression-2)
  IF (Expression-3)
   ⋮

  Else
    .msg = nld126
```

Or two separate comparison pairs:

```
IF (Expression-1 AND Expression-3 OR
          Expression-2 AND Expression-3)
   ⋮

Else
  .msg = nld127
```

The PANEXIT Statement

The ISPF panel user exit provides a way for you to extend the panel language processing of dialog variables. This processing can include operations such as verification, transformation, translation, and formatting of dialog variables passed to the panel user exit routine. Performing these operations in a panel user exit routine reduces the logic required in the ISPF function programs.

Use the PANEXIT statement in a panel's)INIT,)REINIT, or)PROC section to invoke the panel user exit. This statement causes ISPF to branch to the panel user exit routine. When the routine processing completes, control returns to the next sequential panel language statement.

```
PANEXIT ((value,value,...),{PGM,exit-add
                            [,exit-data]
                            [,MSG=msgid]
                          LOAD,exit-mod
                            [,exit-data]
                            [,MSG=msgid]})
```

where:

value
Specifies the names of dialog variables being passed to the exit. The string of values, including the parenthesis, cannot exceed 255 characters. The string of values can be represented by the name of a dialog variable that contains a list of names of variables being passed to the exit routine.

PGM
Keyword that indicates that the exit routine being invoked was loaded when ISPF loaded the application dialog or was loaded from the application. The application passes ISPF the address of the exit routine in exit-add.

exit-add
This is the name of a 4-byte, fixed-format dialog variable that contains the address of the exit routine, which can reside above or below the 16Mb line. The exit routine receives control in AMODE=31 mode. This parameter is used in conjunction with the keyword PGM.

exit-data
This is the name of a 4-byte fixed-format dialog variable that contains a value, such as the address of an information area, to be passed to the exit routine.

msgid
If no message identification is returned to ISPF from the exit routine, this parameter identifies the message to be displayed if a variable fails the exit routine evaluation. If this parameter is not specified, and no message identification is returned from the exit routine, ISPF issues a generic message indicating that the exit routine evaluation failed.

LOAD
Keyword that indicates that the exit routine is to be loaded dynamically. The application passes ISPF the module name of the exit routine that is to be dynamically loaded. The module name is passed in the exit-mod parm.

exit-mod
This parameter identifies the name of the panel user exit routine module that is to be dynamically loaded by ISPF. The panel user exit name can be passed as a literal or as a dialog variable that contains the panel user exit name. This parameter is used in conjunction with the LOAD keyword.

On the PANEXIT statement you can specify that the following be passed to the panel user exit routine:

- A list of dialog variables to be processed by the exit routine in one call. Rules that apply to the variables being passed are:
  - Variable values must be in character format when passed, and must remain in character format.
  - The exit routine can change a variable's value but it cannot change its length. Thus, if a dialog uses the VDEFINE service to define any of these variables to be passed, it should specify the NOBSCAN option. Otherwise, the variable value's length is considered to be the length of the actual data with blanks being ignored.

    For implicitly defined variables, the variable length is considered to be the same as the length of its value.
- A 4-byte area that you can use to pass the address of data to be used by the exit routine.

- The identification of a message to be issued if a variable fails the exit routine evaluation. ISPF uses this value to set the .MSG control variable or, in the case of a panel user exit severe error (RC=20 or invalid value), to set ZERRMSG.

Notes:

1. A panel user exit routine cannot access any dialog variables except those passed on the call.
2. A panel user exit routine cannot issue requests for any ISPF services.
3. ISPF ignores any PANEXIT statement issued from dialog test option 7.2.
4. The PANEXIT statement cannot be issued from a selection panel that initiates a dialog prior to defining the exit address.

Following a successful validation exit, during which one or more dialog variable values are changed, ISPF updates the values for all dialog variable names included on the PANEXIT statement. This allows the exit routine to define dialog variables for cursor field or cursor position, and to return these values to ISPF when an error has been detected.

How to LOAD the Panel User Exit Routine

If the dialog function routine and the panel user exit routine are separate object modules, you can load the panel user exit routine by either:

- Linking the exit routine object module to the dialog function object module containing the display request for the panel from which the PANEXIT statement is issued. Thus, when ISPF loads the application, it also loads the exit routine.
- Loading the exit routine from the application and passing to ISPF the address of the exit routine.
- Letting ISPF load the exit routine dynamically.

Invoking the Panel User Exit Routine

A dialog invokes the panel user exit by issuing the PANEXIT statement from a panel's)PROC,)INIT, or)REINIT section. If the LOAD keyword is specified, ISPF will issue an OS load to bring the load module into virtual storage. ISPF then invokes the exit routine through a call (BALR 14,15). You must use standard OS linkage conventions when invoking the panel user exit. The exit routine (called in AMODE 31) must support 31-bit addressing.

ISPF uses the standard parameter list format to pass parameters. Register one points to a list of addresses; each address points to a different parameter as shown in Figure 71 on page 250. See "Parameters Passed from ISPF to the Panel User Exit Routine" on page 250 for information on these parameters.

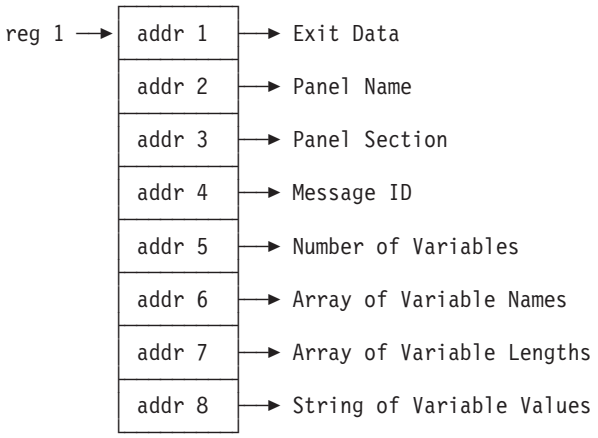


Figure 71. Standard Parameter List Format

The keyword, LOAD, on the PANEXIT panel statement, provides the option of dynamically loading a panel user exit routine. PGM and LOAD are the only valid keywords. PGM indicates that a panel user exit using a compiled source is being invoked. LOAD indicates that the panel user exit routine named by the exit-mod parameter is to be dynamically loaded by ISPF.

ISPF checks the keyword to determine if the panel user exit routine is to be dynamically loaded. If it is, ISPF issues an OS load to bring the load module into virtual storage. The search sequence for link libraries is: job pack area, ISPLLIB, steplib, link pack area, linklib. See *ISPF Services Guide* for further discussion of the search order using LIBDEF.

The panel user exit routine is loaded only once per SELECT level the first time the panel is displayed. The loaded panel user exit routine is not deleted until the SELECT, which first displayed the panel, is terminated.

Parameters Passed from ISPF to the Panel User Exit Routine

Parameters passed to the panel user exit routine are (*in the order passed*):

1. Exit Data
The value of the dialog variable identified on the PANEXIT statement to contain exit data. Its format is a fullword fixed value. If no exit data area is provided, ISPF passes binary zeros.

2. Panel Name
The name of the panel from which the panel user exit is being invoked. Its format is CHAR(8), left-justified in the field. ISPF ignores any changes made to this parameter by the exit routine.

3. Panel Section
A one-character code that identifies the panel section from which the panel user exit is being invoked. Its format is CHAR(1). Its value is:
I for the)INIT section
R for the)REINIT section
P for the)PROC section.

4. Message ID
The identification of the message used to set the .MSG value if the variable evaluation fails. In case of a severe error in the exit routine processing, ISPF uses this value to set variable ZERRMSG. Its format is CHAR(8). When the exit routine is invoked, it contains eight blanks (X'40'). On return to ISPF, if the value in Message ID is not blank, ISPF assumes the value to be a message ID, which must be left-justified in the field.

5. Number of Variables
The dimension of the array of variable names and the array of variable lengths passed to the panel user exit routine. Its format is a fullword fixed value. ISPF ignores any changes made to this parameter by the exit routine.

6. Array of Variable Names
An array of dialog variable names being passed to the panel user exit routine. Each array entry has a format of CHAR(8), left-justified in the array. ISPF ignores any changes made to this parameter by the exit routine.

7. Array of Variable Lengths
An array of dialog variable lengths being passed to the panel user exit routine. Each array entry format is a fullword fixed value. If the exit routine changes any of the variable length values, a severe error results.

8. String of Variable Values
A character buffer of dialog variable values mapped by the array of variable lengths and the array of variable names. The length of the buffer is the sum of the lengths in the array of variable lengths. The exit routine returns updated dialog variable values to ISPF in this buffer.

Return Codes and Error Processing

Return codes, set in the panel user exit routine, recognized by ISPF are:

0 Successful operation.

8 Exit-defined failure. ISPF sets the .MSG control variable and displays or redisplays the panel with the message.

20 (or code unrecognized by ISPF)
Severe error in the exit routine.

For an exit routine return code of 8, ISPF sets the .MSG control variable by using the following search order:

1. If the value in the Message ID parameter is not blank on return to ISPF, that value is used for setting the .MSG control variable.
2. If the value in the Message ID parameter is blank on return, the value (if any) specified for the MSG= keyword on the PANEXIT statement is used for setting the .MSG control variable.
3. If neither the Message ID parameter nor the MSG= keyword has been given a value, the default ISPF exit error message is used for setting the .MSG control variable.

The panel section in which the .MSG control variable is set affects the message display as follows:

-)INIT or)REINIT section — The message is displayed on the panel.
-)PROC section — The panel, including the message to be displayed, is redisplayed.

If the return code from the exit routine is either 20 or not one of the recognized codes, the display service terminates with a severe error condition. ISPF sets the ZERRMSG system variable by using the following search order:

1. If the value in the Message ID parameter is not blank on return to ISPF, it is used for setting the ZERRMSG system variable. This allows the exit routine to define the message to be used in case of a severe error.
2. If the value in the Message ID parameter is blank on return, the value (if any) specified for the MSG= keyword on the PANEXIT statement is used for setting the ZERRMSG system variable.
3. If neither the Message ID parameter nor the MSG= keyword has been given a value, the default ISPF exit error message is used for setting ZERRMSG.

If CONTROL ERRORS CANCEL is in effect, ISPF displays on the severe error panel the message indicated by the value of ZERRMSG.

For a detailed example of using the panel user exit refer to *ISPF Examples*

The REFRESH Statement

The REFRESH statement provides a means to force specified fields in the panel body to be retrieved prior to a redisplay.

```
REFRESH (value, value, ...)
```

where:

value
Name of an input or output field in the panel body.

Typically, when a panel is redisplayed, the automatic fetching of variables that appear in the panel body is bypassed. As a result, all variables are normally displayed as the user last saw them, even though the variable contents can have been changed. REFRESH causes the contents of specified fields to be retrieved and allows the user to see any changes that have occurred since the panel was last displayed.

The REFRESH statement can appear within the)PROC or)REINIT section of a panel definition. ISPF flags it as an error if it appears in the)INIT section. When this statement is encountered, the specified input/output fields within the panel body are retrieved from the corresponding dialog variables prior to redisplay of the panel.

A value of * indicates that *all* input/output fields on the panel are retrieved. You can omit the parentheses if only one field is refreshed.

- Example 1:

```
)PROC
   ⋮

 IF (.MSG ¬= '')
   &STMT = 'Correct invalid field and press Enter key'.
 IF (.MSG = ' ')
   &STMT = ' '
 REFRESH STMT
```

If the panel is displayed again and if the control variable .MSG is set to non-blank in the)PROC section, the panel field STMT is reset to `Correct the ... Enter key`. Otherwise, the field is set to blank.

- Example 2:

```
)REINIT
  REFRESH(SEL, RENAME)
```

  Both panel fields SEL and RENAME are reset with their current values prior to any redisplay.

- Example 3:

```
)REINIT
  REFRESH(*)
```

  All of the panel fields are reset to their current values.

- Example 4:

```
)REINIT
  REFRESH(&RVARS)
```

  The variable RVARS will contain a list of one or more panel fields to be refreshed.

A field that is refreshed on the screen remains unchanged for multiple redisplays unless it is again refreshed.

The TOG Statement

Use the TOG statement to alternate the value of a variable between two values.

```
TOG(mode,fld,&variable[,value1,value2])
```

where:

mode
Mode in which TOG is to function:
- S—single, used for pull-downs and single-choice selection fields.
- M—multiple, used for multiple choice selection fields.

fld Panel field used to determine whether &variable alternates.

&variable
Variable whose value may alternate between value1 and value2.

value1
Value &variable receives if &variable is not equal to value1. The default is 0. Value1 can be a dialog variable or literal.

value2
Value &variable receives if &variable is equal to value1. The default is 1. Value2 can be a dialog variable or literal.

Examples:

```
Value1 = 0
Value2 = 1

IF &variable = Value2
   &variable = Value1
ELSE
   &variable = Value2
```

The statement accepts numeric or alphabetic values. A numeric compare is performed on numeric data. When scan encounters a comma (even if it is followed immediately by an another comma or a right parenthesis) it assumes a value is given. The TOG value will be assigned a blank in this case. For example:

```
TOG(S,fld1,test,)   value1 = ' ' value2 = 1
TOG(S,fld1,test,,)  value1 = ' ' value2 = ' '
TOG(S,fld1,test)    value1 = 0   value2 = 1 (both will use defaults)
```

If the TOG is in single mode, a check is made to determine if the data has been modified. If it has been modified, then the TOG is performed.

If the TOG is in multiple mode, and a check determines that the data has been modified, then:

- If the field contained a character at the last display and it has not been changed to a blank, the TOG is not performed.
- If the field contained a blank and now contains a character, the TOG is performed.

  This is to ensure the selection is not deselected by a different character. Only by blanking the field should the variable be deselected.

The following TOG statement example in Figure 72 uses both single and multiple mode combinations. The single mode TOG statements are prefaced with IF statements and are performed based on the IF statement condition. The multiple mode TOG statements are not conditional. They are performed with each pass through this processing section.

```
)PROC
IF ( &CLS = 1 )
   TOG (S,CLS,&CHSPORT,'0','1')
IF ( &CLS = 2 )
   TOG (S,CLS,&CHSEDAN,'0','1')
IF ( &CLS = 3 )
   TOG (S,CLS,&CHLUXRY,'0','1')
IF ( &PERFMOD ^= ' ' )
   &PERFMOD = '/'
   &PERFORM = 'MODERATE'
ELSE &PERFORM = '0'
TOG (M,PERFMOD,&CHPERFO,'0','1')
IF ( &PERFSUP ^= ' ' )
   &PERFSUP = '/'
   &PERFORM = 'SUPER'
ELSE &PERFORM = '0'
TOG (M,PERFSUP,&CHSUPER,'0','1')
IF ( &PERFULT ^= ' ' )
   &PERFULT = '/'
   &PERFORM = 'ULTRA'
ELSE &PERFORM = '0'
TOG (M,PERFULT,&CHULTRA,'0','1')
)END
```

Figure 72. TOG Statement Example

The VEDIT Statement

The VEDIT statement identifies the variables on which ISPF must do mask validation. The VEDIT statement should precede all other)PROC statements that involve variables, such as the VER statement or the VPUT statement. It must precede any statements that refer to a VMASKed variable. A VEDIT statement

must be coded for all masked variables defined in the panel. An example is shown in Figure 73.

```
VEDIT (variable [,MSG=value])
```

where:

variable
Specifies the name of a dialog variable, whose value is to be verified against the mask pattern specified by the VMASK service.

MSG=value
Optional. Can be set to a message ID in the processing section to cause a message to be displayed.

```
)ATTR  DEFAULT(%+_)
   @ TYPE(INPUT)  INTENS(LOW)
)BODY
%-------------------------------TEST PANEL----------------------------
%COMMAND ===>_ZCMD
%
+   PHONE  %===>@CVAR        + (999)999-999
+   TIME   %===>@FVAR +        HH:MM
+
+
+
+
+      Press%ENTER+to leave this panel
)INIT
)PROC
 VEDIT (CVAR)
 VEDIT (FVAR)
)END
```

Figure 73. VEDIT Example

The VER Statement

Use the verify statement, VER, to check that the current value of a variable meets some criteria. Typically, it is used in the processing section to verify the data stored in a dialog variable. Verification of an input variable value is performed after the value has been stored in the variable pool. The current rules for padding, justification, and VDEFINE apply to the value stored in the pool. ISPF provides several types of verification, as described below.

```
VER (variable [NONBL] keyword [,MSG=value])
                              ALPHA
                              ALPHAB
                              BIT
                              DBCS
                              DSNAME
                              DSNAMEQ
                              EBCDIC
                              ENUM
                              FILEID
                              HEX
                              INCLUDE[IMBLK] value1[,value2]
                              LEN,relational-operator,expected-length
                              LIST,value1[value2...]
                              LISTV,varlist
                              LISTVX,varlist
                              LISTX,value1,value2,...
                              MIX
                              NAME
                              NUM
                              PICT,string
                              RANGE,lower,upper
```

where:

variable
Name of the variable to be checked.

NONBLANK
Optional keyword. Specifies that the variable must contain a value and not all blanks. NONBLANK, or NB, can be specified with another type verification, such as ALPHA, NUM, or HEX. Do this by specifying the NONBLANK keyword 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)
```

If the variable does not meet the verification criteria, ISPF displays a message. The message can 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's)PROC and)REINIT statements is performed.

keyword
Specifies the verification criteria. One of the following keywords must be specified:

ALPHA
The variable must contain only lowercase or uppercase alphabetic characters (A–Z, a–z, #, $, or @). Blanks are not allowed.

ALPHAB
The variable must contain only lowercase or uppercase alphabetic characters (A–Z or a–z). Blanks are not allowed.

BIT
The variable must contain all zeros and ones.

DBCS
The variable must contain only valid DBCS characters.

DSNAME
The variable must contain a valid TSO data set name. A data set name qualifier must begin with an alphabetic character (A–Z, $, @, or #). The remaining characters must be either uppercase alphanumeric or a hyphen (-). A period is used to connect each qualifier in the data set name.

ISPF first determines if the TSO/E NOPREFIX PROFILE option is in use. If it is, ISPF does use a prefix in the calculation of the data set length. A maximum of 44 characters can be entered for a data set name, if that data set name is enclosed in quotes. If the TSO/E NOPREFIX PROFILE option is in use, a maximum of 44 characters can be entered for a data set name when it is not enclosed within quotes. If the TSO/E NOPREFIX PROFILE option is not in use, a maximum of 42 characters can be entered for a data set name, not enclosed in quotes. ISPF uses the minimum data set prefix of two characters (one character and a period separator) during its calculation of the data set name length.

DSNAMEQ
This parameter provides the same function as DSNAME with the additional feature that if the TSO data set name starts with a quotation mark and no ending quotation mark is found, DSNAMEQ adds the ending quotation mark for you.

EBCDIC
The variable must contain only valid EBCDIC characters.

ENUM
The variable can contain, in addition to numeric characters:
- Plus sign (+)
- Negative number indicators
- Delimiter symbols
- Decimal symbol (.)
- Certain national language decimal symbol (,).

ISPF ignores leading blanks. Blanks between characters (except the French language delimiter) and trailing blanks are not allowed. This includes blanks between leading or trailing signs and the adjacent character. Use of any characters other than those listed just above results in ISPF issuing an appropriate error message.

The ENUM parameter allows verification of a numeric variable that has been expressed in a more natural style. ISPF verifies variable values for correct decimal and comma notation plus correct sign placement.

Negative number indicators include a leading or trailing minus sign and a number enclosed by parentheses. The decimal and delimiter symbols can vary according to national language. The negative number indicators are common to all national languages.

Use of delimiter symbols is optional. However, if they are used, ISPF validates the delimiter symbols beginning at the left-most symbol that it finds in the variable being verified. In case of an invalid placement or omission of a delimiter symbol, ISPF issues an appropriate error message.

Use of the decimal symbol is optional. A maximum of one decimal symbol is allowed. If used, the decimal must be correctly placed in relation to any delimiter symbols used. Delimiter symbols are not allowed to the right of a decimal symbol. In case of an invalid placement of a decimal symbol, ISPF issues an appropriate error message. Table 14 illustrates decimal and delimiter symbol use for each of the national languages supported by ISPF.

Table 14. Decimal and Delimiter Symbols

Language	Whole	Fractional
Danish	999,999.88	0.789
English	999,999.88	0.789
French	999.999,88	0,789
German	999.999,88	0,789
Italian	999.999,88	0,789
Japanese	999,999.88	0.789
Korean	999,999.88	0.789
Portuguese	999.999,88	0,789
Spanish	999.999,88	0,789
Traditional Chinese	999,999.88	0.789
Simplified Chinese	999,999.88	0.789
Swiss-German	999.999,88	0,789

The variable being verified can contain leading blanks. Any trailing blanks in the variable's value in the variable pool cause a verify error condition. Trailing blanks result from defining the variable by using the VDEFINE service with the NOBSCAN option specified. These trailing blanks are not overlaid when the variable is updated by a panel operation if the corresponding panel field has a justification attribute of LEFT or ASIS.

Note: ISPF treats fields containing the non-numeric characters allowed when using VER ENUM as character fields. To use these fields in numeric operations, an installation can need to provide a routine to convert the fields from character to numeric data. The ISPF VDEFINE exit routine is one option available for incorporating these conversion routines.

Table 15 shows examples of results when verifying variable values (English) with the ENUM keyword specified.

Table 15. Verifying Variable Values with the ENUM Keyword Specified

Value	Results	Reason
+2574	Valid	Leading plus sign is allowed
-2574	Valid	Leading minus sign allowed
25.74	Valid	Decimal allowed
.2574	Valid	Leading decimal allowed
2,574	Valid	Delimiter character allowed (but not required)
(2,574)	Valid	Alternate method of showing a negative value allowed

Table 15. Verifying Variable Values with the ENUM Keyword Specified (continued)

Value	Results	Reason
2574-	Valid	Trailing minus sign allowed
2574+	Invalid	Trailing plus sign not allowed
-2574-	Invalid	Double negative indication not allowed
(2,574)	Invalid	Two errors; blanks not allowed between either sign indicator and the adjacent character
35,543785	Invalid	If used, the delimiter character must be inserted at every appropriate point (35,543,785)
4,5932.673	Invalid	Delimiter must be positioned in relation to decimal (45,932.673)
33.452.78	Invalid	Only one decimal allowed in numeric field
8.364,798	Invalid	Delimiter not allowed to right of decimal

FILEID
The variable must contain a valid file ID in CMS syntax. The file name and file type, if given, must be from 1–8 alphanumeric characters, including A–Z, 0–9, $, #, @, +, - (hyphen), : (colon), and _ (underscore). 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 fileid can be an asterisk (*) or a string of characters followed by an asterisk. For example:

`tr* status`
All files having a file name beginning with the letters `tr` and having a file type of status.

`* exec` All files having a file type of exec.

HEX
The variable must contain only hexadecimal characters (0–9, A–F, a–f).

INCLUDE [IMBLK] *value1*[,*value2*]
Defines a list of value parameters, each specifying the character types a verify field is allowed to contain.

IMBLK
Optional positional subparameter. Indicates that the variable is allowed to contain imbedded blanks. Any leading or trailing blank characters are ignored.

value1,value2
Specifies ALPHA, ALPHAB, or NUM; at least one value must be specified. The specification of two different values are combined and indicate to ISPF that the field can contain data of either type. ISPF issues an error message if more than two values are specified.

Example:

```
)PROC
   VER (&vara,NB,INCLUDE,IMBLK,ALPHAB,NUM,MSG=NSL001)
   VER (&varb,NB,INCLUDE,IMBLK,NUM,MSG=NSL002)
   VER (&varc,NB,INCLUDE,ALPHA,NUM,MSG=NSL003)
   ⋮
```

This example illustrates that the variable *vara* can contain any alphabetic (A–Z or a–z) or numeric character as well as imbedded blanks; *varb* can

contain numeric characters only and imbedded blanks; and variable *varc* can only contain alphabetic characters (A–Z, a–z, #, $, or @) and/or numeric characters (0–9), but no imbedded blanks.

LEN,relational-operator,expected-length
The length of the variable (number of characters) must satisfy the condition expressed by the relational operator and expected length.

You can use the LEN function in a panel's)INIT,)REINIT, or)PROC section to verify the number of characters (bytes) in a variable that is currently residing in the variable pool.

For DBCS character strings the number of bytes in the string is twice the number of characters.

relational-operator
Valid relational operators are:

= or EQ
Equal to

< or LT
Less than

> or GT
Greater than

<= or LE
Less than or equal

>= or GE
Greater than or equal

¬= or NE
Not equal

¬> or NG
Not greater than

¬< or NL
Not less than.

You can specify the relational operator either as a special symbol (=, <, and so forth) or as a character symbol (EQ, LT, and so forth) expressed in uppercase. A relational operator can be expressed either as a literal value (remember to enclose special symbol values in quotes) or as a dialog variable containing the value.

expected-length
The expected-length operand is a positive number having a maximum of 5 characters, with which ISPF compares the number of characters in the variable data. Like the relational operator, the expected-length operand can be expressed as a literal value or as a dialog variable containing the value.

Example:

```
VER (&NAME,LEN,'<=',8)
```

This statement verifies that the number of characters defining the value of variable &NAME is less than or equal to 8.

Example:

```
VER (&NAME,LEN,NG,&SIZE)
```

This statement verifies that the number of characters defining the value of variable &NAME is not greater than the value of dialog variable &SIZE

When input fields are stored in their corresponding dialog variables, any keyed leading or trailing pad characters associated with right or left justification of the variable field are deleted prior to being stored.

The length of a variable, used by ISPF for comparison, is the total number of characters in the variable as it is currently stored in the variable pool. Thus, for a variable created using the VDEFINE service with NOBSCAN specified, any trailing blanks are included in the length value used for comparison.

If a variable has been defined using the VDEFINE service but currently has no value, ISPF uses a length value of zero for comparison.

LIST,value1,value2, ...
The variable must contain one of the listed values. The maximum number of listed values allowed is 100.

LISTV,varlist
Allows the use of a variable containing a list of values to be used for variable field verification.

varlist
When defined within the panel, this is the name of a variable, preceded by an &, that contains a list of values that will be compared to the value contained in the verify variable. The *varlist* variable can contain up to 100 values. Each value in the *varlist* variable must be delimited by a comma or at least one blank. A value in the *varlist* variable containing any of the following special characters should be enclosed in single quotes (' '):

Blank < (+ |) ; ¬ - , > : =

To specify the ampersand character in a value contained in the *varlist* variable, or a period in a value contained in the *varlist* variable when it immediately follows a dialog variable name, you must double these characters. To specify the single quote character in a value contained in the *varlist* variable, use two single quote characters enclosed within single quotes (''').

If the varlist is set in the dialog, use the notation that is correct for the programming language used to code the dialog.

Example:

```
)PROC
  ⋮

  VER (&areacode,NONBLANK,LISTV,&varlist,MSG=NSL011)
  ⋮
```

The variable specified in the VER LISTV variable parameter must be set before being referenced in the statement. (The variable used in the previous example could have been assigned the following values in the)INIT section of the panel definition.)

```
&varlist ='919 914 212'
```

Note: To have quotes as part of an assignment, you must double the number of quotes used in each previous layer. For example:

```
&list1 = 'one o''ne' yields one o'ne
&list2 = 'two t''''wo' yields two t''wo
```

LISTVX,varlist
The *varlist exclude* (LISTVX) keyword enables you to use a variable containing a list of values that the field variable must *NOT* contain. The use of this parameter implies that the keyword *non-blank* is used. The varlist follows the same rules as the varlist for LISTV.

LISTX,value1,value2,...
The *list exclude* (LISTX) keyword enables you to list values that the field variable must *NOT* contain. The use of this parameter implies that the keyword *non-blank* is used. The maximum number of listed values allowed is 100.

MIX
The variable must contain all valid DBCS, EBCDIC, shift-in, and shift-out characters.

NAME
The variable must contain a valid name, following the rules of member names, up to eight alphanumeric characters (A–Z, #, $, @, 0–9). The first character must be alphabetic (A–Z, $, @, or #).

NUM
The variable must contain all numeric characters (0–9). However, leading blanks are acceptable.

PICT,string
The variable must contain characters that match the corresponding type of character in the picture string. The *string* parameter can be composed of the following characters:

C — any character
A — any alphabetic character (A–Z, a–z, #, $, @)
N — any numeric character (0–9)
9 — any numeric character (same as `N`)
X — any hexadecimal character (0–9, A–F, a–f)

In addition, the string can contain any special characters that represent themselves. For example:

```
VER (xxx,PICT,'A/NNN')
```

In this example, the value must start with an alphabetic character, followed by a slash, followed by 3 numeric characters. The length of the variable value and the picture string must be the same. Trailing blanks are not included.

RANGE,lower,upper
The variable must contain all numeric characters (0–9). It can also contain a leading plus (+) or minus (–). Its value must fall within the specified lower and upper limits, which can be either positive or negative. The length of the specified variable is limited to 16 digits, in addition to the plus or minus sign. Further, the *lower* and *upper* parameters can consist of no more than 16 digits each, in addition to the plus or minus sign, if used. Any characters in excess of the 16 allowed are truncated.

MSG=value
value contains the message issued if the current value of the variable does not meet the criteria being checked.

For all tests except NONBLANK, a blank value is acceptable. That is, if you enter a value, or leave a nonblank initial value unchanged, it must conform to the specified condition. If a variable value is stored as all blanks, the value passes any verification test except NONBLANK.

Figure 74 on page 264 shows a sample panel with VER statements to verify that information entered 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 and a length of 3.
- The PHNUM (local number) field contains 3 numeric characters followed by a hyphen, followed by 4 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 is 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,ALPHAB)
  VER (&FNAME,ALPHAB)
  VER (&I,ALPHAB)
  VER (&PHA,LEN,'=',3)
  VER (&PHA,NUM)
  VER (&PHNUM,PICT,'NNN-NNNN')

)END
```

Figure 74. Sample Panel Definition with Verification

The VGET Statement

The VGET statement copies variables from the shared or application profile variable pool.

VGET *name-list* **[ASIS|SHARED|PROFILE]**

where:

name-list
Specifies one or more dialog variables, separated by commas or blanks, whose values are to be copied from the shared or application profile pool. The names are passed in standard name-list format. A name-list of more than one name must be enclosed in parentheses.

ASIS Variable values are to be copied from the shared variable pool, if found there; otherwise, they are to be copied from the application profile pool. ASIS is the default value.

SHARED
Variable values are to be copied from the shared variable pool.

PROFILE
Variable values are to be copied from the application profile variable pool. ISPF deletes any shared pool variables having the same name, even if they do not exist in the application profile pool.

Note: Specifying a nonmodifiable variable in a VGET statement in a selection panel results in a severe error.

DISPLAY Service Panel

When processing a DISPLAY or TBDISPL service request, ISPF normally searches for dialog variable values in the order:
1. Function pool
2. Shared pool
3. Application profile pool

To give you control over the pool from which ISPF retrieves variable values, the VGET statement in a panel's)INIT,)REINIT, or)PROC section allows you to specify that ISPF is to copy one or more variable values from either the shared pool or application profile pool to the function pool. If one or more of these variables already exist in the function pool, their values are updated with the values of the corresponding variables accessed by the VGET statement. Any of these variables that do not exist in the function pool are created and updated with the values of those accessed by the VGET statement.

Example:
```
)PROC
   VGET (XYZ ABC) PROFILE
```

This VGET statement in a panel's)PROC section causes the current values for variables XYZ and ABC to be copied from the profile pool and updated in the function pool and used as the variable values for display of a panel field. If XYZ and ABC do not already exist in the function pool, they are created then updated.

SELECT Service Panel

At the time ISPF processes a SELECT service request, there is no function pool. Therefore, ISPF normally searches for dialog variable values in the order:
1. Shared pool
2. Profile pool

When specified on a selection panel, the VGET statement functions as follows:
- If the variable value is taken from the profile pool, the shared pool value, if it exists, is deleted.
- Otherwise, the VGET statement has no effect.

Further processing of the variables on the selection panel, other than that by the VGET statement, is described in *ISPF User's Guide*

The following is an example of a VGET statement on a selection panel, where the specified variable exists in both the shared and profile pools:
```
VGET FNAME PROFILE
```

This statement causes ISPF to retrieve the current value of variable FNAME from the profile pool and display it in the corresponding panel field. Any updates to the variable are made to the profile pool. ISPF deletes the variable from the shared pool.

The VPUT Statement

While variables entered from a panel are automatically stored in the function variable pool, variables can also be stored in the shared and profile variable pools by VPUT statements used in the)INIT,)REINIT,)ABCINIT,)ABCPROC, or)PROC sections of the panel definition.

```
VPUT name-list [ASIS|SHARED|PROFILE]
```

where:

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.

ASIS Specifies that the variables are to be copied to the pool in which they already exist 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 only to the shared pool.

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

PROFILE
Specifies that the variables are to be copied to the application profile pool. Any shared pool variables with the same names are deleted.

Example:

```
)PROC
  VPUT (XYZ ABC) PROFILE
```

This statement 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 command procedure except that the ISPEXEC command verb is omitted.

Using ISPF Control Variables

Control variables are used to control and test certain conditions pertaining to the display of a panel or message. Only those that apply to displays are discussed in this section. They can be used only in the)INIT,)REINIT, and)PROC sections of a panel definition.

This section describes the following control variables:

- .ALARM – page 268
- .ATTR – page 269
- .ATTRCHAR – page 269
- .AUTOSEL – page 272

- .CSRPOS – page 272
- .CSRROW – page 273
- .CURSOR – page 273
- .HELP – page 275
- .MSG – page 275
- .PFKEY – page 277
- .RESP – page 277
- .TRAIL – page 278
- .ZVARS – page 278.

Control variables are automatically reset to blank when the panel display service first receives control. If .MSG, .CURSOR, and .CSRPOS 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 placement. Under certain conditions, processing of the initialization section is bypassed.

Once .CURSOR, .CSRPOS, .MSG, and .RESP have been set to nonblank by panel processing, they retain their initial values until the panel is displayed, or redisplayed, at which time they are reset.

The control variables

```
.ALARM
.AUTOSEL
.CURSOR
.HELP
.MSG
.PFKEY
.RESP
```

have a length of 8 bytes. When set in an assignment statement to a longer value, the value is truncated. If these control variables are tested in a conditional expression, the compare value (literal or dialog variable) must not be longer than 8 bytes.

Figure 75 on page 268 shows an example in which both .HELP and .CURSOR have been set in the)INIT section of the panel definition.

```
)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
  .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,ALPHAB)
  VER (&FNAME,ALPHAB)
  VER (&I,ALPHAB)
  VER (&PHA,NUM)
  VER (&PHNUM,PICT,'NNN-NNNN')

)END
```

Figure 75. Sample Panel Definition with Control Variables

.ALARM

The .ALARM control variable can be set in an assignment statement within the)INIT,)REINIT, or)PROC sections to control the terminal alarm.

```
.ALARM = value
```

where:

value
YES, NO, a blank, or null.

YES Causes the terminal alarm to sound when the panel is displayed.
NO Causes the terminal alarm to be silent when the panel is displayed.
blank Causes the terminal alarm to be silent when the panel is displayed.
null Causes the terminal alarm to be silent when the panel is displayed.

Note: *value* can also be a variable containing the value YES, NO, a blank or null.

Examples:

```
.ALARM = YES
.ALARM = &ALRM
```

In the first example, the .ALARM setting is YES, which causes the terminal alarm to sound when the panel is displayed. In the second example, the alarm setting can be turned on (YES) or off (NO) according to the current value of the variable ALRM. If the panel is displayed with a message that has .ALARM = YES, the alarm sounds regardless of the setting of .ALARM within the panel assignment statement.

Control variable .ALARM can also appear on the right side of an assignment statement. For example:

```
&ALRM = .ALARM
```

might be used to save the setting of .ALARM in variable ALRM.

.ATTR and .ATTRCHAR

.ATTR

The .ATTR control variable can be set in the)INIT,)REINIT, or)PROC section to allow attributes to be changed on a field basis.

```
.ATTR (field) = 'keyword (value),keyword (value)....'
```

where:

field
Can be:

- The name of any input or output field that occurs in the panel body or area section.
- The .CURSOR control variable, which indicates the field where the cursor is currently positioned.
- The name of a dialog variable, preceded by an ampersand. The variable must contain the name of an input or output field that occurs in the panel body, .CURSOR, or a blank.

keyword (value)
A legitimate attribute keyword and value for that attribute.

Examples:

```
.ATTR (.CURSOR) = 'COLOR(YELLOW) HILITE(REVERSE)'
.ATTR (&FLD)    = 'HILITE(&HLTE)'
.ATTR (&FLD)    = 'PAS(ON)'
```

In the first example, the color and highlighting of the field containing the cursor is overridden. In the second example, the name of the field whose highlighting attribute is to be overridden is found in dialog variable FLD and the highlighting value is in variable HLTE.

Overriding the cursor field (.CURSOR) and the alternate long or short message field attributes can be particularly useful if the panel is being redisplayed because of a translation or verification failure. When such a failure occurs, the cursor is automatically placed on the field in error and the message ID to be displayed is automatically placed in the message area.

For example, if SMFIELD is specified on the)BODY statement as the alternate short message field, a)REINIT section could be specified as follows:

```
)REINIT
 .ATTR (.CURSOR) = 'COLOR(RED) HILITE(USCORE)'
 .ATTR (SMFIELD) = 'HILITE(BLINK)'
```

This will cause the field in error to be redisplayed in red and underscored, and the error message to blink.

Only the specified attributes are overridden. Any other attributes associated with the field remain in effect.

When a field attribute is overridden in the)INIT section of a panel, the override remains in effect if the panel is redisplayed (unless the attribute is again overridden by another statement in the)REINIT section). However, an attribute override in the)PROC or)REINIT section of the panel remains in effect only for a single redisplay of that panel, should a redisplay occur. This allows one field at a time to be highlighted as errors are found. Once the user corrects the error, the field reverts to its normal attributes.

.ATTRCHAR

The .ATTRCHAR control variable can be set in the)INIT,)REINIT, or)PROC section to override attributes for all fields related to an existing attribute character.

```
.ATTRCHAR(<char)='keyword(value),keyword(value) ....'
```

where:

char
Can be:

- One of the special characters, one-digit character, or two-digit hexadecimal codes used to denote attribute characters within the panel.
- The name of a dialog variable, the value of which must contain an attribute character, two-digit hexadecimal code, or a blank.

char follows the rules for literals. That is, it must be enclosed in single quotes if it contains any of the special characters listed in “Using Variables and Literal Expressions in Text Fields” on page 113.

keyword (value)
A legitimate attribute keyword and value for that attribute.

When a field attribute is overridden in the)INIT section of a panel, the override remains in effect if the panel is redisplayed unless the attribute is again overridden by another statement in the)REINIT section. However, an attribute override in the)PROC or)REINIT section of the panel remains in effect only for a single redisplay of that panel, should a redisplay occur.

See “Relationship to Control Variables .ATTR and .ATTRCHAR” on page 210 for a description of appropriate and inappropriate override conditions for CUA and basic panel-element attributes.

Using .ATTR and .ATTRCHAR with Table Display Panels

The effect that an attribute override has on a table display panel depends on whether the override is permanent (overridden in the)INIT section) or temporary

(overridden in the)REINIT or)PROC section). If the attribute override for a field or attribute character in the scrollable section of a panel is:

- Permanent, the override for the specified field or character is effective for every model set displayed
- Temporary, the override for the specified field or character is effective for only the last selected model set processed

Any scrolling activity performed when temporary overrides are in effect causes the affected attributes to be cleared, including any temporary overrides in the fixed portion of the panel, and the original attributes to be put into effect. In addition, if a table is redisplayed after model sets have been selected and a scroll has taken place, any .ATTR or .ATTRCHAR temporary overrides are not put into effect.

Things to Remember When Using Attribute Override Control Variables

- The .ATTR or .ATTRCHAR control variable cannot appear on the right side of an assignment statement.
- Several characteristics (for example, INTENSITY, COLOR, and CAPS) can be changed with one attribute override statement. However, only one field can be changed by a single .ATTR statement, and only one attribute character or hexadecimal code can be changed by a single .ATTRCHAR statement.
- The TYPE keyword can be overridden by .ATTR or .ATTRCHAR. You can change the TYPE:

  ```
  from INPUT/CUA input types    to OUTPUT/CUA output types
  from OUTPUT/CUA output types  to INPUT/CUA input types
  from TEXT/CUA text types      to TEXT/CUA text types
  from DATAIN                   to DATAOUT
  from DATAOUT                  to DATAIN

  Exceptions:  CUA TEXT types AB, ABSL, PS, RP
  ```

  However, if you attempt to change the TYPE of a field from TEXT to INPUT, a dialog error will result.

  See “Relationship to Control Variables .ATTR and .ATTRCHAR” on page 210 for a description of appropriate and inappropriate override conditions for CUA and basic panel-element attributes.
- The command field or scroll amount field cannot be changed to TYPE(OUTPUT) by an attribute override assignment.
- The first .ATTR assignment that is encountered within a panel section for a particular field is the one that is honored. Subsequent .ATTR assignments for that field are ignored. In the following example, FIELD1 will be blue and FIELD2 will be yellow:

  ```
  )INIT
     .ATTR(FIELD1) = COLOR(BLUE)
     .ATTR(FIELD2) = COLOR(YELLOW)
     .ATTR(FIELD1) = COLOR(RED)
  ```
- Similarly, the first .ATTRCHAR assignment that is encountered within a panel section for a particular attribute character or hexadecimal code is the one that is honored.
- Be careful when overriding the pad character. Since the string of overridden attribute keywords is in quotes, the new pad character must be specified either without quotes or in double quotes, as follows:

```
.ATTR(FIELD1) = 'PAD($)'
.ATTR(FIELD2) = 'PAD(''*'')'
```

- If both an .ATTRCHAR assignment and an .ATTR assignment apply to the same field, the .ATTR assignment takes precedence.

  Example:

```
)BODY
   ⋮

%===>_FIELD1             +
)INIT
 .ATTRCHAR(_) = 'COLOR(YELLOW)'
 .ATTR(FIELD1)  = 'COLOR(WHITE)'
)REINIT
 IF (.MSG ¬= ' ')
    .ATTR(FIELD1) = 'COLOR(RED) HILITE(BLINK)'
    .ATTRCHAR(_) = 'COLOR(BLUE)'
)PROC
 VER(&FIELD1,NB)
)END
```

  When this panel is initially displayed, FIELD1 will be white and all other input fields will be yellow. If the panel is redisplayed with a message, FIELD1 will be blinking red and all other input fields will be blue. If the panel is redisplayed without a message, FIELD1 will revert to white, and all other input fields will revert to yellow.

.AUTOSEL

The .AUTOSEL control variable is used in conjunction with table display panels to specify auto-selection.

```
.AUTOSEL = YES | NO
```

where:

YES
Indicates that if the CSRROW parameter or control variable is specified, the row is to be retrieved even if the user did not explicitly select the row. This is called auto-selection.

NO
Indicates that if the CSRROW parameter or control variable is specified, the row is to be retrieved only if the user explicitly selects the row by entering data in the corresponding model set on the screen.

If the CSRROW parameter or control variable is not specified, .AUTOSEL is ignored. .AUTOSEL can be set in the)INIT or)REINIT section. Any assignment made to .AUTOSEL in the)PROC section is ignored.

.CSRPOS

The .CSRPOS control variable can be set in the)INIT or)REINIT section and controls where in a field the cursor is to be set.

```
.CSRPOS = integer
 variable = .CSRPOS
```

where:

integer
Specifies the position in the field to which the cursor is set. This position applies regardless of whether the cursor placement was specified using the CURSOR calling sequence parameter, the .CURSOR control variable in the)INIT or)REINIT section, or the default cursor placement. If cursor-position is not specified or is not within the field, the default is one, the first position of the field.

The .CSRPOS control variable can appear on the right side of an assignment statement, making it act like a function. Thus, the cursor field name and its position within a field can be stored in variables.

Example:

```
&CPOS  = .CSRPOS
```

In the preceding statement, the position (an integer value) of the cursor within the input or output field or area is returned in variable CPOS.

.CSRROW

The .CSRROW control variable is used in conjunction with table display panels.

```
.CSRROW = CRP-number
 variable = .CSRROW
```

where:

CRP-number
Table current-row-pointer number corresponding to the model set on the display where the cursor is to be placed. If the specified row does not have a corresponding model set displayed on the screen, the cursor is placed at the command field. The row will be auto-selected under either of the following conditions:

- If the CSRROW parameter is specified on the TBDISPL service either without AUTOSEL(NO) being specified on TBDISPL or .AUTOSEL(NO) specified as a panel definition statement.
- If the .CSRROW control variable is specified as a panel definition statement either without AUTOSEL(NO) being specified on TBDISPL or .AUTOSEL(NO) specified as a panel definition statement.

The .CSRROW control variable can appear on the right side of an assignment statement, making it act like a function. Thus, the table row number corresponding to the model set on the display where the cursor is to be placed can be stored in a variable.

Example:

```
&CROW = .CSRROW
```

.CURSOR

The .CURSOR control variable can be set in the)INIT or)REINIT section to control the placement of the cursor.

```
.CURSOR = string
 variable = .CURSOR
```

where:

string
A character string that matches a field name or a DYNAMIC or GRAPHIC area name in the panel body. Its value cannot be a character string that matches a scrollable area name, but it can be a character string that matches a field name within the scrollable area.

Example:

```
.CURSOR = DSN
```

This example causes the cursor to be placed at field DSN. This variable is automatically set to the field last referred to whenever .MSG is set explicitly or indirectly by TRANS or VER statements. The .CURSOR control variable overrides any cursor position specified on the DISPLAY or TBDISPL service request.

Note: In GUI mode, .CURSOR can be set only to an input or pushbutton (point-and-shoot) field. If the application attempts to set the cursor to any other field, ISPF ignores the placement and uses the default cursor placement.

The .CURSOR control variable can appear on the right side of an assignment statement, making it look like a function.

Example:

```
&CNAME = .CURSOR
```

If the control variable .CURSOR is not explicitly initialized, or if it is set to blank, the initial field where the cursor is positioned (default placement) is determined as follows:

1. 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; that is, the corresponding dialog variable must be null or blank.
   - It must not have a field name of ZCMD.
2. If the stated criteria is not met in the panel body, the scrollable areas are searched using the same criteria.
3. If the criteria is still not met, the cursor is placed on the first input field in the panel body or scrollable area, usually the command field.
4. If the panel has no input fields, the cursor is placed at the upper-left corner of the screen.

The cursor is automatically placed at the beginning of the field that was last referred to in any panel definition statement when a message is displayed because of:

- A verification failure that sets .MSG
- A .MSG=value condition in a TRANS
- An explicit setting of .MSG

Examples:

```
&XYZ = TRANS (&A ... MSG=xxxxx)
&A = TRANS (&XYZ ... MSG=xxxxx)
VER (&XYZ,NONBLANK)  VER (&B,ALPHA)
```

Assume that field XYZ exists in the panel body, but there are no fields corresponding to variables A or B. In all the preceding examples, the cursor would be placed on field XYZ if a message is displayed.

.HELP

The .HELP control variable can be set in the initialization section to establish a tutorial (extended help) panel to be displayed if the user enters the HELP command.

```
.HELP = panelname
variable = .HELP
```

where:

panelname
Name of the tutorial panel to be displayed.

Example:

```
.HELP = ISPTE
```

This example causes tutorial panel ISPTE to be displayed when the user enters the HELP command.

The .HELP control variable can appear on the right side of an assignment statement, making it act like a function.

.HHELP

The .HHELP control variable can be set in the initialization section to establish a tutorial (extended help) panel to be displayed if the user enters the HELP command from within HELP.

```
.HHELP = panelname
```

where:

panelname
Name of the tutorial panel for help to be displayed.

Example:

```
.HHELP = ISP00006
```

This example causes tutorial panel ISP00006 to be displayed when the user enters the HELP command from HELP. This also happens to be the default setting. The Dialog Tag Language generates the setting .HHELP = ISP00006 for any help panels it builds.

.MSG

The .MSG control variable can be set to a message ID, typically in the processing section, to cause a message to be displayed.

```
.MSG = msgid
variable = .MSG
```

where:

msgid
The message ID of the message to be displayed.

Example:

```
.MSG = ISPE016
```

This variable is automatically set by use of the MSG=value keyword on a TRANS statement if there is no match with the listed values, or on a VER statement if the verification fails.

The .MSG control variable can appear on the right side of an assignment statement, making it act like a function.

.NRET

On enabled panels, the .NRET key retrieves the library names from the current library referral list or data set, or workstation file name from the current data set referral list. Unlike some othe dot variables, .NRET **can** be assigned multiple times in panel logic.

```
.NRET = ON|OFF|DSN|LIB
```

where:

ON
Sets the NRETRIEV command table entry active.

OFF
Sets the NRETRIEV command table entry inactive.

DSN
Tells ISPF that the NRETRIEV command retrieved a name from the current data set referral list.

LIB
Tells ISPF that the NRETRIEV command retrieved a name from the current library referral list.

Other values are reserved by ISPF. No messages are given in case of an assignment that is not valid.

When .NRET is used as the source for an assignment statement it always returns a null.

The user is responsible for assigning NRETRIEV to a PF key. NRETRIEV is normally inactive but can be made active by using the .NRET=ON assignment in the)INIT and)REINIT section of a panel. If it is turned on, .NRET=OFF **must** be executed in the)PROC section of the panel. Failure to turn off .NRET in the)PROC section of the panel can lead to errors when the NRETRIEV key is pressed on subsequent panels.

NRETRIEV sets the following variables in the FUNCTION pool:

Variable	Function
ZNRPROJ	Project name

Variable	Function
ZNRGRP1	First group name
ZNRGRP2	Second group name
ZNRGRP3	Third group name
ZNRGRP4	Fourth group name
ZNRTYPE	Type name
ZNRMEM	Member name
ZNRODSN	Other data set name
ZNRVOL	Volume associated with the other data set name
ZNRLIB	Successful library retrieve (YES or NO)
ZNRDS	Successful data set retrieve (YES or NO)
ZNRWSN	Workstation name indicator for other data set name (H = Host, W = Workstation)

.PFKEY

The .PFKEY control variable is set to a value that reflects the function key pressed by a user while the panel is being displayed.

```
.PFKEY = value
variable = .PFKEY
```

where:

value
The function key (F01–F24) pressed by a user.

The value of .PFKEY can be examined in the)PROC section of the panel and copied into dialog variables through use of assignment statements. If no function key is pressed by the user, .PFKEY contains blanks. .PFKEY is blank during processing of the)INIT and)REINIT sections.

The .PFKEY control variable can appear on the right side of an assignment statement, making it act like a function.

.RESP

The .RESP control variable indicates *normal* or *exception* response on the part of the user.

```
.RESP = ENTER | END
variable = .RESP
```

where:

ENTER
Normal response. ISPF always sets .RESP to ENTER unless the user enters an END or RETURN command.

END
Exception response. ISPF sets .RESP to END if the user enters an END or RETURN command.

.RESP Control Variable

The value in .RESP can be tested in the processing section to determine the user's response.

Example:

```
IF (.RESP = END)
```

Setting .RESP in the)INIT or)REINIT section of the panel definition has no effect if a message is being displayed.

The)INIT or)REINIT section can be coded with the following statements to assure that the panel is not displayed, regardless of whether a message was specified on the DISPLAY service request.

Example:

```
)INIT   or   )REINIT
  IF (.MSG ¬= &Z)
    .MSG = &Z
  .RESP = END
```

This variable can be set in a panel processing section to force an END or ENTER response. This can be particularly useful if a verification has failed (or .MSG was set) and you want that panel to be redisplayed with the message even if the user entered END or RETURN.

The .RESP control variable can appear on the right side of an assignment statement, making it act like a function.

.TRAIL

The .TRAIL control variable contains the *remainder* following a truncate (TRUNC) operation.

```
variable = .TRAIL
```

where:

variable
Assigned the value in .TRAIL.

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

The .ZVARS control variable can be set in the initialization section to a list of variable names that correspond to Z place-holders in the body and/or model lines.

```
.ZVARS = var | '(varlist)'
variable = .ZVARS
```

where:

var
Name that corresponds to a Z place-holder.

varlist
One or more variable names that correspond to Z place-holders.

The .ZVARS control variable can appear on the right side of an assignment statement, making it act like a function.

Using Z Variables as Field Name Place-Holders

In the body and area sections of a panel definition and in the model lines for a table display panel, the name of an input or output field can be represented by the single character Z. This serves as a place-holder; the actual name of the field is defined in the initialization section of the panel definition.

Use of place-holders 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. The names are in 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 place-holder that appears in the body or model lines. The second name in the list corresponds to the second Z, and so forth.

In the example shown in Figure 76, the input field labeled TYPE is 1 character long and the next two input fields are each 2 characters long. The names of these three fields are TYPFLD, LNGFLD, and OFFSET, respectively.

```
)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 76. Example of Z Variable Place-Holders

The name list assigned to .ZVARS must be enclosed in single quotes because the list contains special characters (parentheses) and blanks. As with other name lists, either commas or blanks can be used to separate the names in the list. .ZVARS can also be set to a dialog variable that has a valid name list as its value. For example:

```
.ZVARS = &NLIST
```

where the value of &NLIST is (TYPFLD LNGFLD OFFSET). See “Defining the Area Section” on page 169 for the description of how to use Z place-holders in scrollable panel areas.

Chapter 7. ISPF Help and Tutorial Panels

Online help and tutorial panels are a set of panels that a developer can include to provide online information for an application user. Help and tutorial panels can contain information that is helpful to a first-time user. They also can instruct a user to take specific actions based on a particular condition that has occurred during the application processing.

All ISPF help panels that are created using the Dialog Tag Language display in a pop-up window. ISPF help panels created using the ISPF panel source statements and containing the WINDOW keyword on the panel's)BODY statement also display in a pop-up window. If field-level help is being displayed, the ISPF help facility attempts to position the pop-up window relative to the object field.

The width and depth values specified on the HELP tag or on the WINDOW keyword must be valid for the device on which these help panels are displayed. Refer to the *ISPF Dialog Tag Language Guide and Reference* for details on the HELP tag. See page 212 for details on the WINDOW keyword.

You can provide the following types of help or tutorial panels. The ISPF tutorial is shipped with the product.

Extended help (panel help)
Provides general information about the contents of a panel. The information in extended help can be an overall explanation of items on the panel, an explanation of the panel's purpose in the application, or instructions for the user to interact with the panel.

See the description of the .HELP variable in ".HELP" on page 275 for more information.

Field-level help
Provides help panels for fields defined on an application panel.

When the user enters the HELP command, ISPF displays the help panel defined for the field on which the cursor is located.

You may define field-level help for action bar choices and pull-down choices, as well as for fields within the panel body. If you are creating panels with field level help using Dialog Tag Language, refer to the *ISPF Dialog Tag Language Guide and Reference* for a description of the tag attributes you should use. Otherwise, for further information about defining the)HELP section of the panel, refer to "Defining the HELP Section" on page 219.

HELP FOR HELP
Provides help for using the help or tutorial facility.

Keys help
Provides a brief description of each key defined for a panel. See "Keys Help" on page 98 for more information on keys help.

Message help
Provides help for ISPF messages. See "How to Define a Message" on page 292 for more information.

© Copyright IBM Corp. 1980, 1999

Reference phrase help
Provides help for reference phrases. See "Reference Phrase Help" on page 98 for more information.

Tutorial
Describes the ISPF product. The tutorial is shipped with the ISPF product. See "The ISPF Tutorial Panels" on page 285 for more information.

TUTOR command
Provides a direct path to specific tutorial panels, in effect indexing Help hierarchies by panel identifiers.

Processing Help

You can request help from an application panel or a help panel. You can also specify a keylist to be associated with a help panel.

Help Requests from an Application Panel

When the user enters the HELP command, ISPF displays a help or tutorial panel according to the following sequence.

1. When a short message appears on an application panel and the user requests HELP, ISPF displays the long message.
2. If a long message is on the screen and the user requests HELP, ISPF checks to see if message help is defined.
   - If message help is defined, ISPF displays that panel. If the user requests help from the message help panel, the Help Tutorial panel is displayed.
   - If message help is not defined, ISPF checks to see if field-level help is defined for the field on which the cursor is located.
     - If field-level help is defined, ISPF displays that panel. If the user requests HELP from the field-level help panel, the Help Tutorial panel is displayed.
     - If field-level help is not defined, ISPF checks for panel help.
       - If panel help is defined, ISPF displays that panel. If the user requests HELP from the panel help panel, the Help Tutorial panel is displayed.
       - If panel help is not defined, ISPF displays the first panel within the application's tutorial.
3. When an application panel has been displayed and the user requests HELP, ISPF checks to see if field-level help is defined for the field on which the cursor is located.
   - If field-level help is defined, ISPF displays that panel. If the user requests HELP from the field-level help panel, the Help Tutorial panel is displayed.
   - If field-level help is not defined, ISPF checks for panel help.
     - If panel help is defined, ISPF displays that panel. If the user requests HELP from the panel help panel, the Help Tutorial panel is displayed.
     - If panel help is not defined, ISPF displays the first panel within the application's tutorial.

Figure 77 on page 283 illustrates the panel flow for help according to the ISPF search sequences.

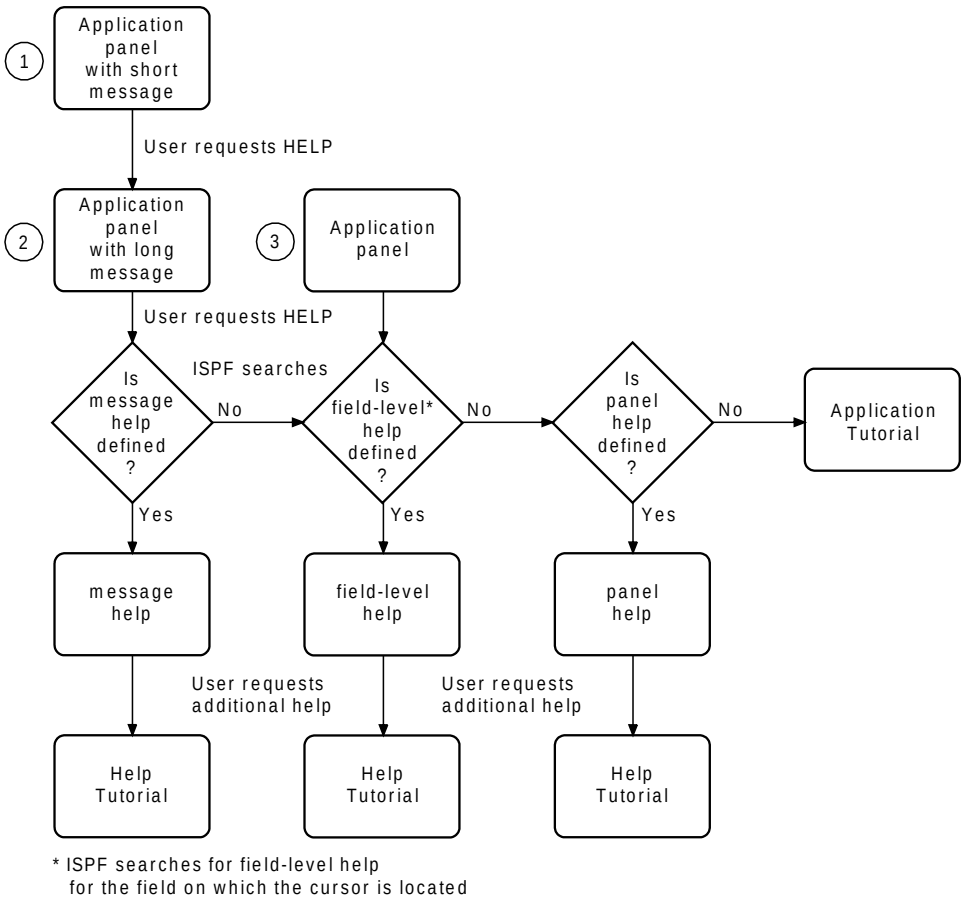


Figure 77. Help Panel Flow

Keys Help Request from an Application Panel

When an application panel is displayed and the user requests KEYSHELP, ISPF displays the keys help panel (provided that keys help is defined).

If the panel contains a short message, and/or long message and the user requests KEYSHELP, ISPF displays the keys help panel without following the search sequence as illustrated in Figure 77.

Extended Help Request from an Application Panel

When an application panel is displayed and the user requests EXHELP, ISPF displays the extended help panel (provided that extended help is defined).

If the panel contains a short message, and/or long message and the user requests EXHELP, ISPF displays the extended help panel without following the search sequence as illustrated in Figure 77.

Help Available from a Help Panel

The following list describes the ISPF help facilities available when a help panel or tutorial panel is displayed.

- If the user requests HELP from any help or tutorial panel, ISPF displays the help for help panel defined by the .HHELP control variable. If the variable is not defined, then ISPF displays the Help Tutorial panel.
- If the user requests EXHELP from any help or tutorial panel (except from the extended help panel), ISPF displays extended help.

  Note: If the user requests EXHELP from the extended help panel, ISPF issues a message stating that extended help is currently displayed.
- If the user requests KEYSHELP from any help or tutorial panel (except the keys help panel), ISPF displays keys help.

  Note: If the user requests KEYSHELP from the keys help panel, ISPF issues a message stating that keys help is currently displayed.
- If the help panel contains a reference phrase, and the user requests HELP while the cursor is positioned on a reference phrase, ISPF displays the reference phrase help panel defined. When a reference phrase help panel is cancelled, the help panel from which reference phrase help was requested is redisplayed. All other help facilities are available from a reference phrase help panel.

Ending Help

When the user requests END or EXIT from any help panel (except the Help Tutorial panel), ISPF returns to the original application panel. If the user requests END or EXIT from the Help Tutorial panel, ISPF returns to the previous panel.

If the user requests CANCEL from any help or tutorial panel, ISPF returns to the previous panel.

ISPF Default Keylist for Help Panels

You can specify a keylist to be associated with a help panel by using the keylist attribute on the HELP tag (DTL) or by using the)PANEL statement in your panel definition. If you do not specify a keylist, ISPF uses the keys defined for ISPHELP to display in the function area of the help panel when it is displayed.

The key settings and forms for ISPHELP are shown in Table 16. Refer to the *ISPF User's Guide* for more information on keylists.

Table 16. ISPHELP Key Settings

Key	Command	Form
F1	HELP	Short
F2	SPLIT	Long
F3	EXIT	Short
F5	EXHELP	Short
F6	KEYSHELP	Short
F7	UP	Long
F8	DOWN	Long
F9	SWAP	Long
F10	LEFT	Long
F11	RIGHT	Long
F12	CANCEL	Short

The ISPF Tutorial Panels

A tutorial panel is a special type of panel that is processed by the ISPF tutorial program. This program invokes the panel display service to display the panel.

A user invokes the ISPF program that displays tutorial panels in four ways:

- As an option from a menu
- Directly or indirectly from any non-tutorial panel by entering the HELP command or by pressing the function key assigned to the HELP command.
- By selecting a choice from a Help pull-down
- Through the use of the TUTOR command

Transfer into and out of the tutorial using the HELP command is transparent (no action required) to ISPF functions.

ISPF tutorial panels are arranged in a hierarchy. Generally, this hierarchy is a table of contents, each succeeding level of which contains a more detailed list of topics. When the tutorial is entered from a menu, the first panel to be displayed is usually 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 you were doing when help was requested.

When viewing the tutorial, you can select topics by entering a selection code or by simply pressing Enter to view the next topic. On any panel, you can 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
TOC or T	To display the table of contents
INDEX or I	To display the tutorial index

Note: If you enter the UP command after viewing a portion of a tutorial sequentially and if you do not select a new topic from the displayed list, you can resume the tutorial at the next sequential topic on the list by entering the NEXT command or by pressing Enter.

You can use the following keys whenever you are in the tutorial:

ENTER	To display the next sequential page or scroll a scrollable help panel
HELP	To redisplay this page for *help* information
END	To terminate the tutorial
UP	To display a higher level list of topics (rather than typing UP)
DOWN	To skip to the next topic (rather than typing SKIP)
RIGHT	To display the next page (rather than pressing Enter) or to scroll a scrollable help panel
LEFT	To display the previous page (rather than typing BACK) or to scroll a scrollable help panel

When running under tutorial and trying to scroll past the end of the scrollable area, a message will be displayed indicating that no more information is available in the scrollable area. If RIGHT or ENTER is pressed again, ISPF will follow the normal tutorial flow and display the next help panel if one has been defined. The same is true when scrolling to the TOP of the scrollable AREA; a message indicating that no more information is available will be displayed, and if LEFT is pressed, the previous tutorial panel will be displayed if one has been defined.

Cursor positioning usually defines which scrollable area will be scrolled. However, when in tutorial, if the cursor is not within a scrollable area, the first area defined in the)BODY section will be scrolled. The LEFT and RIGHT commands should be included in any keylist specified for a scrollable help panel.

If you issue the HELP command while viewing a tutorial, ISPF displays a tutorial panel that contains a summary of commands that are available to the tutorial user.

When you end the tutorial, using the END or RETURN command, the panel from which you entered the tutorial is displayed again.

The name of the top panel must be specified by dialog variable ZHTOP. 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 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 "Example of a Primary Option Menu" on page 127.

The index is optional. It is a collection of panels in which topics are arranged in alphabetic order. You can jump to the index from any point by using the INDEX command. The index need not be connected to the main tutorial hierarchy. It can be a topic that you can select from the main table of contents or other panels. A list of the last 20 tutorial panels displayed, including the current panel, is maintained by ISPF. You should issue the TOP or INDEX command instead of the BACK command if you want to view panels displayed before the last 20 panels.

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 can 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 Enter to view the next topic.

The maximum number of entries allowed is 100.

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 can 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 if it 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.

Variables SEL, UP, and CONT are provided for compatibility with the previous SPF product. Use of variable names ZSEL, ZUP, and ZCONT is recommended.

A panel cannot have both a continuation panel and selectable topics. However, the last panel in a sequence of continuation panels can have selectable topics.

Help/tutorial panels can contain variables so that dialog information, including information entered by a user, can be displayed on the help panel. Function variables, as well as shared and profile variables, can be displayed.

Figure 78 shows a sample hierarchy of tutorial panels. Panels A and B have three selectable topics each. Panels C and D2 have two selectable topics each. 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 78, assuming that panel A is the highest-level table of contents, the viewer can get to A from any point by issuing the TOC command. A viewer currently on panel F1, F2, or F3 can return to panel B by issuing the BACK command. Then, from B, the SKIP command would take the viewer to panel C. If the user enters the TUTOR command along with a panel identifier parameter, a specific tutorial panel within the Help hierarchy is displayed. From that point on, any movement within the hierarchy is the same as if the user had reached the panel by any other means.

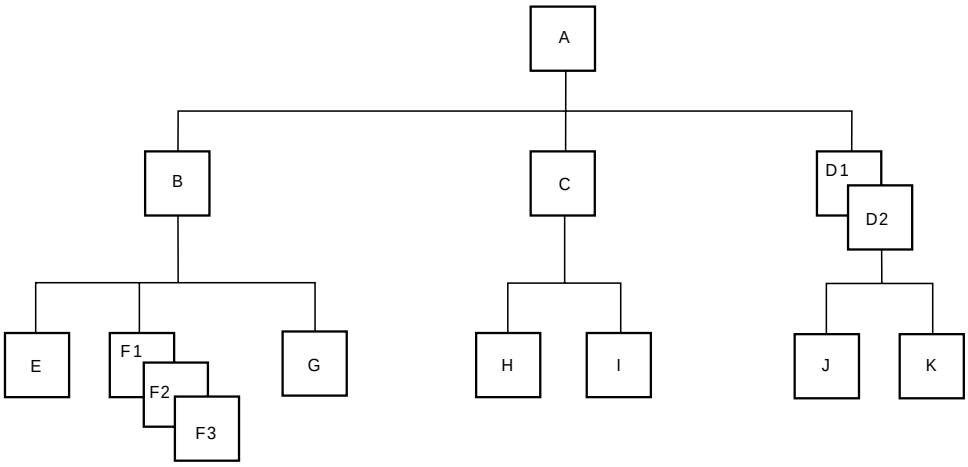


Figure 78. Sample Tutorial Hierarchy

Two sample tutorial panels are shown in Figure 79 and Figure 80 on page 289 . These are assumed to be panels B and F2, respectively, in the hierarchy in Figure 78 on page 287.

```
%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 can 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 79. Sample Tutorial Panel Definition (Panel B)

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 Enter to view each topic. The order in which topics are presented when Enter is pressed is the same as the order in which they appear in the TRANS function. If option 3 is selected, pressing the Enter key does not display the other topics.

In panel B, the name of the parent panel (A) is stored in variable ZUP.

```
%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 80. Sample Tutorial Panel Definition (Panel F2)

Panel F2 (Figure 80) 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.

If you call ISPTUTOR from an edit macro, be sure to save and restore the environment at that point. For example:

```
ISREDIT MACRO
ISPEXEC CONTROL DISPLAY SAVE
ISPEXEC SELECT PGM(ISPTUTOR) PARM(panel-id)
ISPEXEC CONTROL DISPLAY RESTORE
EXIT
```

Chapter 8. Defining Messages

This chapter describes how to create and change ISPF messages. You can create messages in two ways:

- Using the existing message definition.
- Using the MSG and MSGMBR tags of the Dialog Tag Language (DTL). Refer to the *ISPF Dialog Tag Language Guide and Reference* for more information about these tags.

ISPF message definitions are stored in a message library and displayed by using the DISPLAY, TBDISPL, or SETMSG service, written to the ISPF log file by the LOG service, or copied to variables specified in a GETMSG service request. You create or change messages by editing directly into the message library. ISPF interprets the messages during processing. No compile or preprocessing step is required.

Note: When not in TEST mode, the most recently accessed message definitions are retained in virtual storage for performance reasons. If you have modified a message, using TEST mode will ensure that the updated version of the message will be picked up by ISPF services. See "ISPF Test and Trace Modes" on page 25 for more information.

Several messages can be within each member of the message library. When using the PDF editor to create a message file, prevent numbers from appearing in the file by specifying NUMBER OFF.

The member name is determined by truncating the message ID after the second digit of the number.

For example:

Message ID	Member Name
G015	G01
ISPE241	ISPE24
XYZ123A	XYZ12
ABCDE965	ABCDE96
EMPX214	EMPX21

All messages that have IDs beginning with the characters `G01`, for example, must be in member G01. Figure 81 on page 292 shows an example of a member in the message library. This member contains all message IDs that begin with `EMPX21`.

© Copyright IBM Corp. 1980, 1999

```
EMPX210   'INVALID TYPE OF CHANGE' .HELP=PERSO33   .ALARM=YES
'TYPE OF CHANGE MUST BE NEW, UPDATE, OR DELETE.'

EMPX213    'ENTER FIRST NAME'       .HELP=PERSO34  .ALARM=YES
'EMPLOYEE NAME MUST BE ENTERED FOR TYPE OF CHANGE=NEW OR UPDATE.'

EMPX214    'ENTER LAST NAME'        .HELP=PERSO34  .ALARM=YES
'EMPLOYEE NAME MUST BE ENTERED FOR TYPE OF CHANGE=NEW OR UPDATE.'

EMPX215    'ENTER HOME ADDRESS'     .HELP=PERSO35  .ALARM=YES
'EMPLOYEE NAME MUST BE ENTERED FOR TYPE OF CHANGE=NEW OR UPDATE.'

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

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

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

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

Figure 81. Sample Messages

How to Define a Message

Messages generally should appear in collating sequence by message ID. Each message within the library consists of two required lines and (optionally) additional long message lines. The additional lines can contain up to 512 bytes of long message text. Figure 82 illustrates the syntax for defining messages.

```
Line 1:
msgid ['short message'][.HELP=panel|*][.ALARM=YES|NO]
[NOKANA|KANA][.WINDOW=RESP|NORESP|LRESP|LNORESP]
[.TYPE=NOTIFY|WARNING|ACTION|CRITICAL]
Line 2:
'long message' [+]

Additional long message text lines – optional

Line 3:
['long message' [+] ]
Line 4:
['long message' [+]  ]
Line n:
['long message'     ]
```

Figure 82. Example Syntax for Defining Messages

msgid

Required. Each message is referred to by a message identifier (ID). A message ID can be four to eight characters long. It 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. Use the message ID suffix if more than 10 messages are to be included in one member.

short message

Optional. If a short message is specified on an ISPF panel, it is displayed first (before the long message). Its maximum length is 24 bytes. The short message is displayed in a pop-up window if the text is longer than will fit in the short message area or if you defined a message window using the .WINDOW keyword for the message. Otherwise, the short messages are right-justified and displayed, with a high intensity attribute, either:

- At the right end of the first line on the screen, if an action bar is not defined
- At the right end of the line following the action bar

If the user enters the HELP command, the long message is displayed, with a high intensity attribute. If the user enters the HELP command again, 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 under "Defining the Body Section" on page 212.

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.

Note: For long or short messages in pop-up windows, if the message originates from panel processing, such as a verification error message, the message pop-up window is placed adjacent to the field that is the object of the validation.

.HELP=panel | *

Optional. (Can be abbreviated to **.H**) If the user enters tutorial mode, the panel name specified by .HELP is the first tutorial page displayed. If .HELP=* is specified, the first tutorial page is the one specified in the panel definition, that is, the panel on which this message is being displayed. The default is *.

NOKANA|KANA

Optional. The NOKANA keyword allows messages to contain lowercase characters, and still display correctly on a Katakana terminal. Because hexadecimal codes for some lowercase characters overlap those of some Katakana characters, they would display as meaningless characters on a Katakana terminal. If the NOKANA keyword is present in a message definition, ISPF translates any lowercase message characters to uppercase before displaying the message on a Katakana terminal.

In summary, if the terminal is Katakana, and:

- KANA is specified, all characters are left as is.
- NOKANA is specified, lowercase characters are translated to uppercase.
- If neither KANA nor NOKANA is specified, all characters are left as is.

If the terminal is not Katakana, and:

- KANA is specified, lowercase characters are displayed as periods
- NOKANA is specified, all characters are left as is.
- If neither KANA nor NOKANA is specified, all characters are left as is.

Notes:

1. On non-Katakana terminals, the KANA keyword can be used to display overlapping Katakana characters as periods rather than as meaningless lowercase characters.
2. On Katakana terminals, the NOKANA keyword is necessary in messages containing lowercase English characters.
3. See "Chapter 10. Extended Code Page Support" on page 313 for the discussion of the treatment of the KANA or NOKANA keywords if a CCSID is specified.

.ALARM=YES | NO
Optional. (Can be abbreviated to **.A**) If .ALARM=YES is specified, the audible alarm sounds when the message displays. If .ALARM=NO is specified, the alarm does not sound unless .ALARM is set to YES in the panel definition. The default is NO.

.WINDOW=RESP | NORESP | LRESP | LNORESP
Optional. (Can be abbreviated to **.W**) The .WINDOW keyword tells ISPF to display the message in a message pop-up window.

.WINDOW=RESP (R is a valid abbreviation for RESP) requests ISPF to display both long and short messages in a message pop-up window that requires the user to press Enter before data can be entered into the underlying panel. The user cannot enter data or interact with the underlying panel until Enter (or some other attention key) is pressed.

.WINDOW=NORESP (N is a valid abbreviation for NORESP) requests ISPF to display both long and short messages in a message pop-up window that does not require direct user response. The user can enter data into the underlying panel while this message is being displayed.

.WINDOW=LRESP (LR is a valid abbreviation for LRESP) requests ISPF to display only long messages in a message pop-up window that requires the user to press Enter before data can be entered into the underlying panel. The user cannot enter data or interact with the underlying panel until Enter (or some other attention key) is pressed.

.WINDOW=LNORESP (LN is a valid abbreviation for LNORESP) requests ISPF to display only long messages in a message pop-up window that does not require direct user response. The user can enter data into the underlying panel while this message is being displayed.

The MSGLOC parameter on the DISPLAY, TBDISPL, and SETMSG services controls the placement of the message pop-up window. For messages that originate from panel processing, such as a verification error message, the message pop-up window is placed adjacent to the field which is the object of the validation. The window placement will be such that it does not overlay the object field, if possible. If no correlation can be made between the validation and a field (such as when the variable being validated is not a panel field name), the message pop-up window is displayed at the bottom of the logical screen or below the active pop-up window, if one exists. See *ISPF User's Guide* for a complete description of the MSGLOC parameter.

.TYPE=NOTIFY | WARNING | ACTION | CRITICAL
Optional. (Can be abbreviated to **.T**) The .TYPE keyword in the message

definition identifies the particular type of message. There are four types of messages, NOTIFY, WARNING, ACTION, and CRITICAL. N, W, A, and C are valid abbreviations.

Table 17 summarizes the characteristics of the different types of messages.

Table 17. Message Characteristics

Type	Color	Intensity	Placement	Response	Alarm
NOTIFY	White	High	Message area or pop-up window	Optional	Off
WARNING	Yellow	High	Message area or pop-up window	Optional	On
ACTION	Red	High	Message area or pop-up window	Optional	On
CRITICAL	Red	High	Pop-up window	Required	On

The .TYPE keyword overrides any .ALARM value that can be specified and a .TYPE=CRITICAL message is always displayed as though .WINDOW=RESP was specified. The color and highlighting characteristics defined above apply to messages displayed in the default short/long location and a pop-up message window. The dialog application controls the field attributes for alternate message location fields.

long message

Required. If a short message is not specified, the long message is automatically displayed first, with a high intensity attribute, in the long message area or in a message pop-up window. The long message is displayed in a pop-up window if the text is longer than will fit in the long message area, if you defined a message window using the .WINDOW keyword for the message, or if you have selected this option on the Settings panel.

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 under “Defining the Body Section” on page 212.

The maximum length of the long message text is 512 bytes. If the message text is greater than 512 bytes, it will be truncated. Messages greater than 78 bytes require multiple long message lines. The continuation of the long message text into additional lines is indicated by one or more spaces following the ending quote (’) followed by a plus (+) sign. For example:

```
ISPX001 'short message text'
'Long message text' +
' continued over ' +
'multiple lines.  The maximum length is ' +
'512 bytes.'
```

For the best results, use the fewest number of message lines possible.

```
ISPX001 'short message text'
'Long message text continued over multiple lines.  The maximum' +
' length is 512 bytes.'
```

Consecutive SOSI characters resulting from multiple lines of DBCS data are automatically removed. For example,

```
'Long messageSDBS' +
            O  I
'SCSSdata.'
 O  I

Result:  Long messageSDBCSSdata.
                    O    I
```

The ending SI in the first record and the beginning SO in the second record are automatically removed.

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.

The long message text will be written to multiple records if the text is greater than 78 characters.

Existing dialogs which have VDEFINEd the system variable ZERRLM as 78 characters should be updated to VDEFINE this variable as 512 characters.

Note: For long or short messages in pop-up windows, if the message originates from panel processing, such as a verification error message, the message pop-up window is placed adjacent to the field which is the object of the validation.

Message Display Variations

The following tables show various message display situations and the effect of the .TYPE keyword and the PANEL DISPLAY CUA MODE field on the color and highlighting of the message text. The variations are dependent on whether or not you used the Dialog Tag Language (DTL) or the panel definition statements to define your panels.

Note: If you are running in GUI mode, messages that would appear in a pop-up window in non-GUI mode will be displayed in a message box. The message box will include the appropriate icon as defined by CUA guidelines:

- .TYPE=NOTIFY produces an *i* in a circle, the international symbol for information
- .TYPE=WARNING produces an exclamation point (!)
- .TYPE=ACTION or .TYPE=CRITICAL produces a red circle with a diagonal line across it

If your dialog application panels are generated using the DTL, the dialog manager displays the messages as shown in Table 18 on page 297.

Table 18. Message Display Using DTL

Message Definition	Text	Intensity
.TYPE=NOTIFY .ALARM=YES|NO	White	High
.TYPE=WARNING .ALARM=YES|NO	Yellow	High
.TYPE=ACTION .ALARM=YES|NO	Red	High
.TYPE=CRITICAL .ALARM=YES|NO	Red	High
.TYPE not specified .ALARM=NO	White	High
.TYPE not specified .ALARM=YES	Yellow	High

If your application panels are generated from the panel definition statements and you use the default message placement, the dialog manager displays the messages as documented in Table 19.

Table 19. Message Display Using Panel Definition Statements

Message Definition	Text	Intensity
.TYPE=NOTIFY .ALARM=YES|NO	White	High
.TYPE=WARNING .ALARM=YES|NO	Yellow	High
.TYPE=ACTION .ALARM=YES|NO	Red	High
.TYPE=CRITICAL .ALARM=YES|NO	Red	High
.TYPE not specified .ALARM=NO CUA mode=YES	White	High
.TYPE not specified .ALARM=YES CUA mode=YES	Yellow	High
.TYPE not specified .ALARM=NO CUA mode=NO	White	High
.TYPE not specified .ALARM=YES CUA mode=NO	White	High

If you define your panels using the panel definition statements and you use an alternate message placement, the dialog (using the field attributes) controls the message text color and highlighting.

Messages Tagged with CCSID

An ISPF message can be defined with .CCSID=xxxxx where xxxxx is the CCSID of the EXTENDED CODE PAGE as defined by Character Data Representation Architecture. Refer to "Supported CCSIDs" on page 318 for which CCSIDs are supported.

Panels or messages tagged with the CCSID keyword invoke the TRANS service. The ***to*** CCSID is the value in ZTERMCID. This value is filled in during ISPF initialization as the result of the terminal query done by ISPF. The ***from*** CCSID is the CCSID entered following the CCSID keyword.

If the CCSID keyword is used, the characters in the message are translated to the equivalent characters in the terminal code page for display. This translation occurs only if the terminal has returned information to allow ISPF to determine its CCSID and only if the code page indicated by the CCSID is different from the code page of the terminal.

Note: The same CCSID is used for all messages within a message member. Therefore, this keyword should be in the first record and start in the first column of the message member. If the .CCSID keyword is not in the first record or does not start in the first column of the first record, it is ignored and character translation does not occur.

```
.CCSID=xxxxx
ISPX001 'short message text'
'Long message text' +
' continued over ' +
'multiple lines.  The maximum length is ' +
'512 bytes.'
```

All characters in the message member which are not short or long message text must be in the Syntactic Character Set:

- A–Z
- a–z
- 0–9
- + < = > % & * ″ ′
- ( ) , _ - . / : ; ?

The beginning and ending inhibited character tables are enhanced to include characters from the extended code pages for the supported Asian Pacific languages in formatting message text. The CCSID of the message is used to determine which tables to use. If no CCSID is specified, the session language ID and terminal type determine the tables used. See "Chapter 10. Extended Code Page Support" on page 313 and "Message Pop-Up Text Formatting".

Modeless Message Pop-Ups

ISPF allows you to cancel a modeless message pop-up by positioning the cursor within the bounds of the message pop-up and requesting CANCEL or ENTER. This allows you to remove the message pop-up without submitting the underlying panel for processing.

For the cursor to be within the bounds of the message pop-up, it must be inside the window frame of the message. Placing the cursor on the message window frame does not result in the message window being cancelled. Note that asynchronous command processing is not suspended when the cursor is placed inside a message window. Therefore, commands such as PRINT and SPLIT are executed when typed on the command line and Enter pressed, even if the cursor is placed inside a modeless message pop-up window.

The HELP command will not display *message help* for a message window that has been cancelled.

Message Pop-Up Text Formatting

The message text is retrieved from the message member. If it is more than one line (that is, if ISPF finds at least one blank and a plus sign following the closing quote) then the lines are concatenated together, including blanks within or at the end of the text. Trailing blanks are stripped from any variable values before the values are substituted into the text string.

The width of the message pop-up window is determined based on the location where the window will be placed. If the message is displayed as a result of a panel verification error, the message pop-up is displayed relative to the field in error. If the MSGLOC parameter is specified on the DISPLAY or SETMSG service, the message pop-up is displayed relative to the specified field name. If the MSGLOC parameter is not specified, the message pop-up will be displayed at the bottom of the logical screen or below the active ADDPOP pop-up window, if one exists.

The width of the window will be the width from this determined location to the right edge of the screen. Note that this width will vary based on the screen size the user is running with.

ISPF determines if the message text is to be formatted according to English rules or Asian rules based on the type of data in the message text, MIXED or EBCDIC, together with the message CCSID or the current ISPF session language variable, ZLANG.

If the data contains double-byte characters and the message CCSID is 00930, 00933, 00935, 00937, or 00939, the Japanese (Katakana), Korean, Simplified Chinese, Traditional Chinese, or Japanese (Latin) text formatting rules are used, respectively. If the data contains double-byte characters and the message does not have a CCSID or the CCSID is not 00930, 00933, 00935, 00937, or 00939 and the ZLANG value is JAPANESE, CHINESET, CHINESES, or KOREAN, the Japanese, Traditional Chinese, Simplified Chinese, or Korean text formatting rules are used, respectively. If the data contains double-byte characters and the message does not have a CCSID, or if the message CCSID is not 00930, 00933, 00935, 00937, or 00939, or if the ZLANG is not JAPANESE, CHINESET, CHINESES, or KOREAN, the Japanese text formatting rules are used by default.

If the data is all single-byte data and there is no CCSID for the message, ISPF determines if the application is running on a Japanese Katakana terminal and if the NOKANA keyword was specified on the message definition. If so, ISPF uses the English formatting rules. If NOKANA was not specified, ISPF uses the Japanese Katakana formatting rules. If the application is not running on a Katakana terminal and there is no CCSID for the message, ISPF uses the English formatting rules.

English Rules for Message Text Formatting

Message text exceeding the width of the message window is wrapped to the next line. The text is split at blanks only. If a word is longer than the message window width, the window is expanded to the width of this word. However, if a word exceeds the maximum window size (screen width minus 3), the word will be split and continued on the next line. Once the message formatting is complete, the message pop-up window width will be decreased to the length of the longest line, excluding trailing blanks.

Asian Rules for Message Text Formatting

Some characters should not be placed at the beginning of a line, and some should not be placed at the end of a line. These beginning-and-ending inhibited characters are different among the languages, yet the required process is the same. Thus, ISPF uses the same text formatting process for the Asian languages, but it uses a different beginning-and-ending inhibited character table for each language. The CCSID of the message is used to determine which tables to use. If no CCSID is

specified, the session language ID and terminal type determine the tables used. See "Chapter 10. Extended Code Page Support" on page 313.

The message text is first split into *words*. A SBCS *word* is delimited by blanks, or SO/SI characters. Then any beginning inhibitors are stripped from the beginning of the word and treated as separate words, and any ending inhibitors are stripped from the end of the word and treated as separate words.

Adjoining DBCS alphanumeric characters (that is, Ward 42 characters) are treated as one DBCS *word*. Then any beginning inhibitors are stripped from the beginning of the word and treated as separate words, and any ending inhibitors are stripped from the end of the word and treated as separate words. All other non-Ward 42 double-byte characters are treated as separate DBCS words.

If a word is longer than the message window width, the window is expanded to the width of this word. However, if a word exceeds the maximum window size (screen width = 3), the word will be split and continued on the next line. If the text consists of mixed data and does not fit in one line within the specified width, the first position will always be reserved for an SO character (if first word is double-byte) or for a blank (if the first word is single byte). This will allow the text to be aligned properly.

Words that exceed the width of the message window are wrapped to the next line according to following rules:

```
        ... CE_1 CE
CB CB+1 ...
```

CE_1	CE	CB	CB+1	Process
any	B,X	B	X,E	Backward
E	E	X,B	X,E	Backward
X,B	E	any	any	Forward
X,B	X	B	B	Forward
any other				No process

where:

CE–1 and CE	Last two words that fit on line
CB and CB+1	First two words on next line
E	Ending inhibitor
B	Beginning inhibitor
X	Neither
Forward	Move CE to next line
Backward	Move CB to previous line
No process	Split as is.

Note: If words CE or CB are single-byte words and are more than one character, or if CE or CB are double-byte words and are more than one double-byte character, no special processing is used; the line is split as is.

SBCS and DBCS blanks that end or begin a line will be deleted.

Substitutable Parameters in Messages

A substitutable parameter, a dialog variable name preceded by an ampersand (&), can appear anywhere within the short and long message text. For example:

```
'Volume &VOL not mounted'
```

Substitutable parameters can 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 can also be used to specify the value of .TYPE and .WINDOW.

Substitutable parameters in messages are normally replaced with values immediately before the message displays. If the message is specified for display by using the SETMSG service, substitutable parameters are replaced during SETMSG processing. When the GETMSG service is invoked, substitutable parameters are replaced at the time of the GETMSG call. After substitution of the variables, the short message is truncated to 24 characters and the long message is truncated to 512 characters.

Syntax Rules for Consistent Message Definition

The following rules apply to the syntax of messages as they appear in the message library (Figure 81 on page 292):

- 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 can separate the two-line message specifications within the member.
- Comments can precede or follow a two-line message specified within a member. A comment begins with the characters /* starting in column one.
- In the first line, the fields must be separated by at least one blank. One or more blanks can optionally occur on either side of an equal sign (=).
- The short message, if specified, and the long message must each be enclosed in single quotes (’). If the short message is omitted, the enclosing single quotes are also omitted.
- Within the short or long message text, any non-alphanumeric character can 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 single quote that marks the end of the message.
- 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*, the

  ```
  '&V.DEF' yields 'ABCDEF'
  ```
- A single ampersand followed by a blank is interpreted as a literal ampersand character, not the beginning of a substitutable variable. An ampersand followed by a nonblank is interpreted as the beginning of a substitutable variable.
- A double ampersand can be used to produce a character string starting with an ampersand. The double character rule also applies to single quotes within the

delimiting single quotes 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 single quotes
..  yields  .  immediately following a variable name.
```

DBCS-Related Variables in Messages

The following rules apply to substituting DBCS related variables in messages. These rules also apply to file skeletons and file-tailoring operations.

- If the variable contains MIX format data, each DBCS subfield must be enclosed with shift-out and shift-in characters.

  Example:

  ```
  eeee[DBDBDBDBDB]eee[DBDBDB]
  ee... represents a field of EBCDIC characters
  DBDB... represents a field of DBCS characters
  -[ ]- represent shift-out and shift-in characters.
  ```

- If the variable contains DBCS format data only, the variable must be preceded by the ZE system variable, without an intervening blank.

  Example:

  ```
  ...text...&ZE&DBCSVAR..text...
  ```

- If the variable contains EBCDIC format data and is to be converted to the corresponding DBCS format data before substitution, the variable must be preceded by the ZC system variable, without an intervening blank.

  Example:

  ```
  ...text...&ZC&EBCSVAR..text...
  ```

The ZC and ZE system variables can only be used for the two purposes described above.

Chapter 9. Defining File-Tailoring Skeletons

ISPF skeleton definitions are stored in a skeleton library and accessed through the ISPF file-tailoring services. You create or change skeletons by editing directly into the skeleton library. ISPF interprets the skeletons during execution. No compile or preprocessing step is required.

The description below of skeleton formats applies only to new format skeletons used with ISPF file-tailoring services. There are two types of records that can appear in the skeleton file:

Data records
A continuous stream of intermixed text, variables, and control characters that are processed to create an output record.

Control statements
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 nonblank character in column 2, are interpreted as control statements. A)DEFAULT control statement can be used for assigning different special characters for syntactical purposes. The available control statements are:

-)BLANK
-)CM
-)DEFAULT
-)DOT
-)ENDSEL
-)ENDDOT
-)IM
-)SEL
-)SET
-)TB
-)TBA

Considerations for Data Records

Input records can have a maximum length of 255 bytes. For fixed-length records, the last eight character positions are considered to be a sequence number. The character preceding the last eight characters is considered to be the last input column. Variable-length input records are scanned up to the end of the record.

If variable substitution results in an output record larger than the logical record length of the output file, file tailoring terminates and a message is displayed.

Any blank data records in the input data are deleted from file-tailoring output. However, the)BLANK control statement can be used to produce blank lines in the output file.

The following characters have special meanings:

© Copyright IBM Corp. 1980, 1999

Right Parenthesis—)

Defines the following:

- The start of a control statement when placed in column 1 and followed by a nonblank character in column 2.
- The start of a data record when placed in column 1 and followed by a blank in column 2.

Question Mark—?

Indicates a continuation record when placed in the last input column of the record that is to be continued. The question mark is used as a continuation character when more than one input record maps to a single output record. If any character other than a question mark appears in the last input column of an input record, it is copied to that column of the output record. Continuation is not permitted for variable length input records.

The following control characters also have special meaning:

Ampersand (&)

Indicates the start of a variable name. The value of the corresponding dialog variable is substituted in the output record. A value of all blanks is treated as null. Blank ø < (+ | & ! *) ; ¬ - / , % _ > : ' = " implicitly delimit the end of a variable name.

Period (.)

Causes the value of the variable to be concatenated with the character string following the period when used at the end of a variable name. For example, if variable *V* has the value *ABC*, then:

```
"&V.DEF"  yields "ABCDEF"
```

Exclamation point (!)

Serves as the default tab character for the)TB and the)TBA control statements. The file-tailoring tabbing function works either similarly to that of a typewriter tabbing operation, or you can specify in the)TB syntax that tabbing is not to take place if a tab stop is sensed at the same record position as the tab character.

Less-than (<), vertical bar (|), and greater-than (>)

Specify, respectively, the beginning, middle, and end of a conditional substitution string. For example:

```
<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.

Example:

An input skeleton contains:

```
)SET I = &Z
)SET J = VALUE_OF_J
)SET K = VALUE_OF_K
FIRST CONDITIONAL SUBSTITUTION RESULT: <&J|&K>;
SECOND CONDITIONAL SUBSTITUTION RESULT: <&I|&J>;
```

After processing, the file-tailoring output file contains:

```
FIRST CONDITIONAL SUBSTITUTION RESULT: VALUE_OF_J
SECOND CONDITIONAL SUBSTITUTION RESULT: VALUE_OF_J
```

Two consecutive control characters

Two consecutive control characters in the input record result in one control character being placed in the output record:

```
&&  yields  &
!!  yields  !
<<  yields  <
||  yields  |
>>  yields  >
```

and

```
..  yields  . immediately following a variable name
```

Seven characters,), &, ?, !, <, |, and >, can be overridden with the)DEFAULT control statement. If those characters are overridden, the specified characters are substituted in the list above.

Note: File tailoring treats an ampersand-blank combination in the input record as an invalid variable name.

Considerations for Control Statements

Control statements cannot be continued to a second line. The maximum length of a control statement is determined by the record length of the skeleton.

The general format of a control statement, which must begin in column 1, is:

```
)control-word   parameter1 parameter2  ...  parameter31
```

where each parameter represents a name, value, operator, or keyword.

All control words must be entered in uppercase.

The parameters must be separated by one or more blanks, and cannot contain embedded blanks. A parameter can 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 before the control statement is evaluated. 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, described above.

The following sections describe the specific control statements:

)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. Since the NT parameter causes the data to be imbedded as it is, without any control character or control statement processing, using the NT option improves performance.

The optional OPT parameter indicates that the skeleton is not required to be present in the skeleton library. 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.

)DEFAULT abcdefg

The seven characters represented by `abcdefg` override the use of the), &, ?, !, <, |, and > characters, respectively. Exactly seven characters must be specified.

If you are using a non-U.S. keyboard, refer to "Appendix A. Character Translations for APL, TEXT, and Katakana" on page 327 for text keyboard character translations.

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. If the)DEFAULT statement is used to change defaults during an imbed, it is only in effect for that imbed level. It does not apply to deeper or previous imbed levels. The DEFAULTS will not be in effect for any skeletons that are imbedded but will be in effect for any data in the skeleton after the)IM.

Example 1: This example demonstrates that defaults changed using)DEF do not take effect in imbedded skeletons.

An FTINCL of the following skeleton, which changes the variable name control character, &,; to the ø sign:

```
)DEFAULT )ø?!<|>
)SET A = USERNAME
A: øA
)IM SKEL2
A: øA
```

imbeds SKEL2 that contains:

```
AA: øA
AA: &A
```

and results in the following data in the output data set:

```
A: USERNAME
AA: øA
AA: USERNAME
A: USERNAME
```

Example 2: This example demonstrates that defaults changed in an imbedded skeleton are not passed back to the skeleton doing the)IMBED.

An FTINCL of the following skeleton:

```
)SET A = USERNAME
A: øA
)IM SKEL3
A: øA
```

imbeds SKEL3 which changes the variable name control character & to the ø sign:

```
)DEFAULT )ø?!<|>
AA: øA
AA: &A
```

and results in the following data in the output data set:

```
A: øA
AA: USERNAME
AA: &A
A: øA
```

Example 3: The following example demonstrates how to use the NT parameter to prevent tailoring from occurring when imbedding a file. Using NT eliminates having to change defaults in the imbedded skeleton when it contains default control characters.

An FTINCL of the following skeleton, which imbeds a skeleton with the NT parameter:

```
)SET A = LBL1
&A:
)IM SKEL4 NT
  GO TO &A
```

imbeds SKEL4 that contains:

```
IF (&A < 0) | (&A > 10) THEN
  &A = 0
ELSE
```

and results in the following data in the output data set:

```
LBL1:
IF (&A < 0) | (&A > 10) THEN
  &A = 0
ELSE
  GO TO LBL1
```

)TB value1 ... value16	(standard tabbing)
)TB value1[A] ... value16[A]	(alternate tabbing–designated positions)
)TBA value1 ... value16	(alternate tabbing–all positions)

An exclamation point (!) is used as the default tab character for the)TB control statement. It tabs the output record to the next tab stop and fills the intervening spaces with blanks. The next character following an exclamation point in the input record is put at the tab stop location in the output record. Up to 16 tab stops can be specified. A tab stop specifies a tab position in the output record, and must be in the range 1-255. The default is one tab stop at location 255.

When you use the *standard* tabbing syntax, *)TB value1 ... value16*, and the tab stop value equals the current output position, the tabbing skips to the next tab stop value that is greater than the current output position. The input character following the tab character is then inserted into the position skipped to in the output record.

When you use *alternate* tabbing syntax, specified with an 'A' in the)TB tabbing syntax, and the tab stop value equals the current output position, the input character following the tab character is inserted into the current position in the output record. This allows you to write to the current position of the output record if a tab character in the input record is encountered at the same time as a tab stop is encountered in the output record.

The way you specify alternate tabbing syntax on the)TB control statement determines whether only designated or all tab stop values are affected, even if the tab stop value equals the current position in the output record when a tab character is encountered in the input record. If you specify:

```
)TB value1A ... value16A
```

only the tab stop values to which the character A is appended selectively cause tabbing to stop in any of those positions. If you specify:

```
)TBA  value1 ... value16
```

any tab stop value that equals the current position in the output record when a tab character is encountered in the input record causes tabbing to stop.

Be sure the character that you append for alternate tabbing is an *uppercase* A. Appending an A to the)TB control word ( )TBA) has the same effect as appending an A to all individual tab stop values. When you use the)TBA control word, appending an A to an individual tab stop value has no additional effect.

Example 1: This example uses the standard tabbing syntax – *)TB value1 ... value16*.

An input skeleton file contains:

```
)TB 5 10 20
!ABCDE!F
```

After processing, the file-tailoring output record contains:

Positions 1–4 contain the blanks inserted by the first tab operation.
Positions 5–9 contain ABCDE. Standard tabbing occurs between E and F because tab stop 10 is at the same (not greater than) position of the output record at which the tab character is encountered in the input record.
Positions 10–20 contain blanks inserted by the second tab operation.
Position 20 contains F.

Example 2: This example uses alternate tabbing syntax for designated tab positions – *)TB value1[A]...value16[A]*.

An input skeleton file contains:

```
)TB 5 10A 20
!ABCDE!F
```

After processing, the file-tailoring output record contains:
Positions 1–4 contain the blanks inserted by the first tab operation.
Positions 5–10 contain ABCDEF. F immediately follows E because alternate tabbing is specified for tab position 10. This allows tabbing to stop in the current output record position (10) when the tab character was encountered in the input record.

Example 3: This example uses the alternate tabbing syntax for all tab positions–*)TBA value1 ... value16*.

An input skeleton file contains:

```
)TBA 3 6 10
!ABC!DEF!GH
```

After processing, the file-tailoring output record contains:
Positions 1–2 contain the blanks inserted by the first tab operation.
Positions 3–5 contain ABC. D immediately follows C because alternate tabbing is specified and a tab stop is set at the current output position (6).
Positions 6–8 contain DEF.
Position 9 contains a blank inserted by normal tabbing.
Positions 10–11 contain GH.

)BLANK [number]

The specified number of blank lines are placed in the output file at the point where the)BLANK statement is encountered. If the number parameter is omitted, the default value is 1. The number parameter can be specified as a symbolic variable.

Examples:

```
)BLANK

)BLANK &SPACER
```

The first example inserts a single blank line into the output file. In the second example, the number of blank lines inserted into the output file is equal to the current value of the variable SPACER.

)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)SEL nesting are permitted. The list of records must end with an)ENDSEL statement.

Any of the other control statements can be used between the)SEL and the)ENDSEL control statements. For example, if you want to write information from a table only if variable ABC is set to the name of that table, specify:

```
)SEL &ABC='TABNAME'
)DOT TABNAME
   &FNAME &LNAME
)ENDDOT
)ENDSEL
```

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 can 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). ISPF evaluates connected expressions from left to right and evaluates the connectors with equal priority.

Examples:

```
)SEL  &COND = YES
)SEL  &TEST1 ¬= &Z   |  &ABC = 5
)SEL  &TEST1 ¬= &Z  &&  &ABC = 5
```

)DOT table-name)ENDDOT

Note: The)DOT command parameter *table-name* must be in uppercase for use with ISPF table services.

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 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)DOT nesting are permitted. The same table cannot be processed recursively. The list of records must end with the)ENDDOT statement.

Any of the other control statements can be used between the)DOT and the)ENDDOT control statements. For example, if you want to take information from table ABC, and write any blank table row as a blank line, specify:

```
)DOT ABC
)SEL &LNAME=&Z
)BLANK 1
)ENDSEL
  &FNAME &LNAME
)ENDDOT
```

If the table was already open, it remains open after file tailoring with the CRP positioned at TOP. If it was not open, it is opened automatically and then closed upon completion of file tailoring.

)SET variable = expression

)SET allows a value to be assigned to a dialog variable. The variable name should not be preceded by an ampersand, unless the variable name is itself stored as a variable. A blank is required between the variable and the equal sign and between the equal sign and the expression. 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 (-).

To assign a null value to a dialog variable, use the system variable &Z

Example:

An input skeleton file contains:

```
)SET A = 1
)SET B = 2
)SET C = &A + &B
)SET D = &Z
A is &A, B is &B, C is &C, D is &D
```

The resulting output file contains:

```
A is 1, B is 2, C is 3, D is
```

)CM comment

The statement is treated as a comment. No tailoring is performed, and the record is not placed in the output file. The)N comment statement of Program Development Facility edit models is not a valid control statement for file tailoring. The control statement terminates file tailoring.

Sample Skeleton File

Figure 83 shows a sample skeleton file. The sample skeleton refers to several dialog variables (for example, ASMPARMS, ASMIN, and MEMBER). 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, that is, 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-and-go step. An imbed statement, `)IM`, is used to bring in a separate skeleton for the load-and-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 83. Sample Skeleton File

DBCS-Related Variables in File Skeletons

The following rules apply to substituting DBCS related variables in file skeletons. These rules also apply to messages and file-tailoring operations.

- If the variable contains MIX format data, each DBCS subfield must be enclosed with shift-out and shift-in characters.

  Example:

  ```
  eeee[DBDBDBDBDB]eee[DBDBDB]
  ee... represents a field of EBCDIC characters
  DBDB... represents a field of DBCS characters
  -[ ]- represent shift-out and shift-in characters.
  ```

- If the variable contains DBCS format data only, the variable must be preceded by the ZE system variable, without a n intervening blank.

  Example:

  ```
  ...text...&ZE&DBCSVAR..text...
  ```

- If the variable contains EBCDIC format data and is to be converted to the corresponding DBCS format data before substitution, the variable must be preceded by the ZC system variable, without an intervening blank.

  Example:

  ```
  ...text...&ZC&EBCSVAR..text...
  ```

The ZC and ZE system variables can be used only for the two purposes described above. For file skeleton definition and file tailoring, these two variables can be used only between)DOT and)ENDOT statements. When variable substitution causes a subfield-length of zero, the adjacent shift-out and shift-in characters are removed.

Chapter 10. Extended Code Page Support

EXTENDED CODE PAGE support allows panels, messages, and variable application data to be displayed correctly on terminals using any of the supported code pages. For example, a German panel can be displayed on a French Country Extended Code Page (CECP) terminal, with all common characters displayed correctly. Any characters in the panel that do not exist in the terminal code page are displayed as periods (.).

ISPF supports the EXTENDED CODE PAGES listed in "Supported CCSIDs" on page 318. CCSID stands for Coded Character Set IDentifier. The CCSID is a short identifier, representing a code page and character set combination. An extended CCSID has the same code page as its base CCSID, but has a larger character set.

Translating Common Characters

ISPF translates common characters from EXTENDED CODE PAGES to the code page of the terminal for panel)BODY,)MODEL, and)AREA text, if the panel is tagged with a CCSID, and for the long and short message text if the message member is tagged with a CCSID.

The TRANS service is provided to allow the application to translate variable application data from one CCSID to another CCSID (see *ISPF Services Guide*).

In a panel tagged with a CCSID, all characters that are not)BODY,)MODEL, and)AREA text and all characters in variable names within the)BODY,)MODEL, and)AREA text of a tagged panel and within the message text of a tagged message member must be in the syntactic character set:

- A–Z
- a–z
- 0–9
- + < = > % & * " '
- ( ) , _ - . / : ; ?

Note: Lowercase a–z can be used for any CCSID supported by ISPF except the Japanese (Katakana) Extended CCSID 930.

If an EXTENDED CODE PAGE is specified and the terminal code page and character set is one of those recognized by ISPF, all displayable code points are available for display (no displayable code points are invalidated by ISPF).

If an EXTENDED CODE PAGE is not indicated in a panel or message member, a base character set and code page is assumed based on the terminal type specified in option 0 (see *ISPF User's Guide*).

Z Variables

The following Z variables are available for code page processing:

ZTERMCP
: Terminal code page. Returned as a 4-digit decimal number (4 characters).

© Copyright IBM Corp. 1980, 1999

ZTERMCS
Terminal character set. Returned as a 4-digit decimal number (4 characters).
ZTERMCID
Terminal CCSID. Returned as a 5-digit decimal number (5 characters).
ZERRCSID
Contains the 5-digit decimal CCSID of a dialog error message, or blanks if the error message is not tagged with a CCSID. Returned as a 5-digit decimal number (5 characters).

If an extended code page is specified for a panel or message and the terminal code page cannot be determined, there is no transformation of characters.

Table 20 illustrates when characters will be transformed for Extended Code Page support and when they will not be transformed:

Table 20. Character Transformation Table

	Terminal Query Reply CP/CS Valid for ISPF	**Terminal Query Reply CP/CS Not Returned**	**Terminal Query Reply CP/CS Invalid for ISPF**
CCSID Tag Present	Characters transformed	Characters not transformed	Characters not transformed
No CCSID Tag Present	Characters not transformed	Characters not transformed	Characters not transformed

For DBCS languages, the beginning and ending inhibited character tables are enhanced to include characters from the extended code pages for the text formatting of messages and panels.

Panels Tagged with CCSID

Panels can be defined with a)CCSID section and the NUMBER(xxxxx) keyword where xxxxx is the CCSID of the extended code page as defined by Character Data Representation Architecture. The)CCSID section must be the first section in the panel. See “Defining the CCSID Section” on page 218.

Messages Tagged with CCSID

An ISPF message can be defined with .CCSID=xxxxx. See “Messages Tagged with CCSID” on page 297.

GETMSG Service

The GETMSG service can be called with a CCSID parameter. If the message is tagged with a CCSID, the CCSID will be returned; otherwise, blanks will be returned.

TRANS Service

Users can call the TRANS Service in ISPF to translate variable data specified by the user from one CCSID to another CCSID. The *to* and *from* CCSIDs are also specified by the user in the TRANS call (see *ISPF Services Guide*). For a list of the EXTENDED CODE PAGE translate tables provided by ISPF, see "Extended Code Page Translate Tables Provided by ISPF" on page 323.

ISPccsid Translate Load Modules

The ISPccsid translate load modules provide ISPF with the information needed to translate data from one CCSID to another. There is one ISPccsid translate load module for each of the supported CCSIDs. The name (or alias for those ISPccsid modules provided by ISPF) of each CCSID translate load module is made up of the 5-digit CCSID, prefixed with ISP. For example, load module ISP00111 supports translation of the CCSID 00111. Each CCSID translate load module must contain two translate tables. The required translate tables permit data to be translated between the respective CCSID and CCSID 00500. Additionally, each CCSID load module can contain up to 256 pairs of optional direct translate tables. ISPF will use direct translate tables when available. Otherwise, ISPF translates through CCSID 00500. Translating through CCSID 00500 can result in valid characters being lost. This is due to CCSID 00500 not having all possible code points defined.

ISPccsid Translate Load Module Generation Macro

An assembler macro that permits the user to generate customized ISPccsid translate load modules is supplied with ISPF. The macro also allows the user to add direct translate tables to the ISPccsid translate load modules ISPF supplies with the product.

Only the values for the hex digits X'40' through X'FE' are defined in a given translate table. These are the only code points that vary from CCSID to CCSID.

The assembler macro is:

```
ISPCCSID  CCSID=nnnnn,TO=to-address,FROM=from-address
```

ISPCCSID Macro

The initial ISPCCSID macro usage identifies the CCSID associated with the particular ISPccsid translate load module and provides addresses of the *to* and *from* CCSID 00500 translate tables.

Subsequent usage of the ISPCCSID macro in a particular ISPccsid translate load module generation identifies the CCSID and translate table addresses of optional direct *to* and *from* translate tables.

Description of Parameters

nnnnn
Required parameter. The nnnnn value is a 5-digit decimal (5 characters) number that specifies a CCSID number. The nnnnn value on the first or only ISPCCSID macro definition is the CCSID associated with the ISPccsid translate load module. The nnnnn value on other than the first ISPCCSID macro

definition is the CCSID associated with direct *to* and *from* translate tables. Assembly errors will occur if this parameter is not 5 digits.

to-address
Required parameter. On the first or only ISPCCSID macro definition, this parameter specifies the address of the translate table that converts data from the CCSID associated with the respective ISPccsid translate load module to CCSID 00500. On subsequent ISPCCSID macro definitions within the same ISPccsid translate load module, it specifies the address of the translate table that converts data from the CCSID associated with the respective ISPccsid translate load module to the CCSID specified on this ISPCCSID macro definition.

from-address
Required parameter. On the first or only ISPCCSID macro definition, this parameter specifies the address of the translate table that converts data from CCSID 00500 to the CCSID associated with the respective ISPccsid translate load module. On subsequent ISPCCSID macro definitions within the same ISPccsid translate load module, it specifies the address of the translate table that converts data from the CCSID specified on this ISPCCSID macro definition to the CCSID associate with the respective ISPccsid translate load module.

ISPccsid Translate Load Module Definition Examples

Each ISPccsid translate load module must be compiled separately using assembler H (or functional equivalent). Figure 84 shows an example of a basic translate model, and Figure 85 shows an example of a translate model with two direct CCSID entries.

```
          ISPCCSID CCSID=00111,TO=TRTO500,FROM=TRFR500
*
*
TRTO500   DC    XL191'...            00111 TO 00500
TRFR500   DC    XL191'...            00111 FROM 00500 (00500 TO 00111)
          END
```

Figure 84. Basic ISP00111 Translate Module.

```
          ISPCCSID CCSID=00222,TO=TRTO500,FROM=TRFR500
          ISPCCSID CCSID=00333,TO=TRT00333,FROM=TRF00333
          ISPCCSID CCSID=00444,TO=TRT00444,FROM=TRF00444
*
*
TRTO500   DC    XL191'...            00222 TO 00500
TRFR500   DC    XL191'...            00222 FROM 00500 (00500 TO 00222)
*
*
TRT00333  DC    XL191'...            00222 TO 00333
```

Figure 85. ISP00222 Translate Module with Two Direct CCSID Entries

KANA and NOKANA Keywords

If a CCSID is specified, the KANA (panels and messages) and NOKANA (messages) keywords are ignored by ISPF. Panels and messages specifying the Japanese (Katakana) Extended CCSID (CCSID=00930) are handled as follows regardless of whether or not KANA or NOKANA (for messages) keywords are specified:

- If the terminal code page is the base Katakana code page, all characters in the panel)BODY,)MODEL, or)AREA text or short and long message text, except lowercase English characters, are left as is. Because the base Katakana code page does not support lowercase English characters, all lowercase English characters are translated to uppercase English characters. All other parts of the panel or message must be in the syntactic character set, excluding characters a–z.
- If the terminal code page is non-Katakana, all lowercase English characters in the)BODY,)MODEL, or)AREA text or short and long message text in a panel or message that has been tagged with the extended Katakana code page (CCSID=05026) are translated to the equivalent lowercase English characters in the terminal code page for display. All Katakana characters are displayed as periods (.). For example, the lowercase a, which is X'62' in the extended Katakana code page, is translated to X'81' (lowercase a) in the U.S. English code page. The Katakana character which is X'81' is translated to a period (X'4B') in the U.S. English code page. All other parts of the panel or message must be in the syntactic character set, excluding characters a–z.

Character Translation

Table 21 illustrates the character translation from the extended Katakana code page and from the extended Japanese (Latin) code page (if CCSID=00930 or CCSID=00939 is specified in a panel, message, or in the TRANS service) to the U.S. English (CECP and base) code page, to the extended and base Katakana, and to the Japanese (Latin) Extended code pages for code points X'81', X'62' and X'59'.

Table 21. Character Translation from Extended Katakana Code Page

Destination Code Page	Source CCSID=00930	Translation	Source CCSID=00939	Translation
Base Katakana (base code page)	X'81' X'62'	X'81' X'C1'	X'81' X'59'	X'C1' X'81'
Extended Katakana (CCSID=00930)	X'81' X'62'	X'81' X'62'	X'81' X'59'	X'62' X'81'
U.S. English CECP and Non-CECP Japanese (Latin) Non-Extended	X'81' X'62'	X'4B' X'81'	X'81' X'59'	X'81' X'4B'

Table 21. Character Translation from Extended Katakana Code Page (continued)

Destination Code Page	Source CCSID=00930	Translation	Source CCSID=00939	Translation
Japanese (Latin) Extended (CCSID=00939)	X'81' X'62'	X'59' X'81'	X'81' X'59'	X'81' X'59'

Code Points

Character Translation

X'81' A Katakana character in the Katakana code pages and is lowercase a in the U.S. English (CECP and base) and Japanese (Latin) (Extended and base) code pages.

X'62' Lowercase a in the extended Katakana (CCSID=00930) code page, a Katakana character in the extended Japanese (Latin) (CCSID=00939) code page, and is an unknown character in the U.S. English, base Japanese (Latin), and base Katakana code pages.

X'59' A Katakana character in the Japanese (Latin) Extended (CCSID=00939) code page, and an unknown character in the other code pages.

X'C1' Uppercase A and X'4B' is a period (.) in all of the previously mentioned code pages.

Supported CCSIDs

The CCSIDs listed in Table 22 are supported for panels and messages that specify an EXTENDED CODE PAGE and for the TRANS service.

Table 22. Extended CCSID1 Supported

CCSID	Character Set	Code Page	Country/Language
00037	697	37	U.S.A. Canada Netherlands Portugal Brazil Australia New Zealand
00273	697	273	Austria Germany
00277	697	277	Denmark Norway
00278	697	278	Finland Sweden
00280	697	280	Italy
00284	697	284	Spain L.A. Spanish
00285	697	285	United Kingdom
00297	697	297	France
00420	235	420	Arabic
00424	941	424	Hebrew

Table 22. Extended CCSID1 Supported (continued)

CCSID	Character Set	Code Page	Country/Language
00500	697	500	Switzerland Belgium
00838	1176	838	Thailand
00870	959	870	Latin-2
00871	697	871	Iceland
00875	923	875	Greece
00880	960	880	Cyrillic
01025	1150	1025	Cyrillic
01026	1126	1026	Turkey

Table 23. Extended CCSID1 Supported (EURO)

CCSID	Character Set	Code Page	Country/Language
01140	695	1140	U.S.A. Canada Netherlands Portugal Brazil Australia New Zealand
01141	695	1141	Austria Germany
01142	695	1142	Denmark Norway
01143	695	1143	Finland Sweden
01144	695	1144	Italy
01145	695	1145	Spain L.A. Spanish
01146	695	1146	United Kingdom
01147	695	1147	France
01148	695	1148	Switzerland Belgium
01149	695	1149	Iceland

The extended CCSIDs shown in Table 23 and Table 24 on page 320 are supported for the TRANS service, and also with the use of the CCSID keyword in panels and messages. These are the mixed SBCS/DBCS CCSIDs for these languages.

Japanese (Katakana) and Simplified Chinese EXTENDED CODE PAGES are not supported on any terminal, but these CCSIDs are supported by ISPF for the TRANS service and for tagging panels and messages.

Note: Although the following CCSIDs represent both SBCS and DBCS character sets and code pages, only the SBCS character set and code page are involved in the EXTENDED CODE PAGE support in ISPF.

Table 24. Extended SBCS and DBCS CCSIDs Supported

CCSID	Character Set	Code Page	Country
00930	1172	290	Japanese (Katakana)
00939	1172	1027	Japanese (Latin)
00933	1173	833	Korean
00935	1174	836	Simplified Chinese
00937	1175	037	Traditional Chinese

Base Code Pages for Terminals

Translation to base character sets and code pages is supported for panels, messages, and the TRANS service. See "Base CCSIDs" on page 322.

Direct translation between each base code page and its EXTENDED CODE PAGE is provided. Also, direct translation between both base and extended Japanese (Katakana) and both base and extended Japanese (Latin or English) is provided. All translation between the single-byte EXTENDED CODE PAGES for the double-byte languages and the CECP code pages is through CCSID 00500.

Adding Translate Tables for Extended Code Page Support

You can add code pages to be used for messages and panels that specify code page and for the TRANS service by creating the following translate tables using the sample assembler module ISPEXCP as an example. (ISPEXCP is provided in the SYS1.SAMPLIB library in the MVS environment.) The tables to translate between the new code page and CCSID 00500 are needed to reduce the number of translate tables necessary to translate characters between the new code page and any other supported (or added) code page. For example, to translate characters from a panel with CCSID=xxxxx to a terminal with CCSID=yyyyy, the characters in the panel are first translated to CCSID 00500 and then from CCSID 00500 to CCSID yyyyy for display on the terminal.

Note: The translate tables for the CCSIDs listed in Table 22 on page 318 and Table 24 are provided and shipped with ISPF. Also, see "Extended Code Page Translate Tables Provided by ISPF" on page 323.

Any translate tables that are added must be named ISPnnnnn, where nnnnn is the CCSID. The translate tables should include code points X'40' through X'FE'.

- The following example illustrates the translation to CCSID 00500 from CCSID xxxxx, where xxxxx is the CCSID for the new code page. This CCSID must be different from any of the supported CCSIDs previously listed, and should be a CCSID defined in the Character Data Representation Architecture. In Figure 86 on page 321, xxxxx is 00037.

```
Table      Hexadecimal Code              Position
--------   ------------------------     -------------
TO_500     DC X'4041424344454647'       (X'40' to X'47')
           DC X'4849B04B4C4D4EBB'       (X'48' to X'4F')
           DC X'5051525354555657'       (X'50' to X'57')
           DC X'58594F5B5C5D5EBA'       (X'58' to X'5F')

                     . . .

           DC X'78797A7B7C7D7E7F'       (X'78' to X'7F')
           DC X'8081828384858687'       (X'80' to X'87')

                     . . .

           DC X'E8E9EAEBECEDEEEF'       (X'E8' to X'EF')
           DC X'F0F1F2F3F4F5F6F7'       (X'F0' to X'F7')
           DC X'F8F9FAFBFCFDFE'         (X'F8' to X'FE')
```

Figure 86. Translation to CCSID 00500 from CCSID XXXXX

- Figure 87 illustrates the translation to CCSID xxxxx from CCSID 00500, where xxxxx is the CCSID for the new code page. This CCSID must be different from any of the supported CCSIDs listed above, and should be a CCSID defined in the Character Data Representation Architecture. In this example, xxxxx is 00037.

```
Table      Hexadecimal Code              Position
--------   ------------------------     -------------
FROM_500   DC X'4041424344454647'       (X'40' to X'47')
           DC X'4849BA4B4C4D4E5A'       (X'48' to X'4F')
           DC X'5051525354555657'       (X'50' to X'57')
           DC X'5859BB5B5C5D5EB0'       (X'58' to X'5F')

                     . . .

           DC X'78797A7B7C7D7E7F'       (X'78' to X'7F')
           DC X'8081828384858687'       (X'80' to X'87')

                     . . .

           DC X'E8E9EAEBECEDEEEF'       (X'E8' to X'EF')
           DC X'F0F1F2F3F4F5F6F7'       (X'F0' to X'F7')
           DC X'F8F9FAFBFCFDFE'         (X'F8' to X'FE')
```

Figure 87. Translation to CCSID XXXXX from CCSID 00500

- Optionally, any number of pairs of *to* and *from* tables can be provided for direct translation from the new CCSID to and from another CCSID.

The assembler macro, ISPCCSID, is supplied with ISPF to allow you to generate custom ISPxxxxx translate load modules (where xxxxx is the new CCSID). Calls to this macro must also be coded for the To_500 and From_500 tables and any *to* and *from* tables for direct translation. The load module must either have the name ISPxxxxx (where xxxxx is the new CCSID) or an alias of ISPxxxxx. See "ISPccsid Translate Load Modules" on page 315, "ISPccsid Translate Load Module Generation Macro" on page 315, and "ISPCCSID Macro" on page 315.

Note: New translate tables can still be added based on terminal type as described in *ISPF Planning and Customizing* for untagged messages and panels.

Direct *to* and *from* translate tables can be added for direct translation (to prevent possible loss of characters through CCSID 00500 for character sets other than 697). Additional direct translation tables can also be added to the extended code page translate tables provided by ISPF. The direct translation CCSID must be one of the CCSIDs supported by ISPF, or added by the user. If the CCSID of the terminal is the same as the CCSID in any of the direct translation tables, those tables are used. Otherwise, the To_500 and From_500 tables are used to translate through CCSID 00500.

Note: Both *to* and *from* translate tables must be provided for direct translation tables as well as CCSID 00500 tables, even though there may be no translation needed. For example, to translate from a base CCSID to an extended CCSID for the same code page, all characters will translate to themselves.

Base CCSIDs

The CCSIDs for the BASE CODE PAGES supported by ISPF (that include mixed SBCS/DBCS CCSIDs for the DBCS languages) are listed in Table 25.

Table 25. Base CCSIDs Supported

CCSID	Character Set	Code Page	Country/Language
00803	1147	424	Hebrew (Old)
00931	101	037	Japan (English)
04369	265	273	Germany and Austria
04371	273	275	Brazil
04373	281	277	Denmark and Norway
04374	285	278	Finland and Sweden
04376	293	280	Italy
04380	309	284	L.A. (Spanish Speaking)
04381	313	285	U.K. English
04393	1129	297	France
04934	938	838	Thailand
04966	959	870	Latin-2
04976	960	880	Cyrillic
05029	933	833	Korean
05031	936	836	Simplified Chinese
05033	101	037	Traditional Chinese
08229	101	037	U.S. English and Netherlands
08476	650	284	Spain
09122	332	290	Japan (Katakana)
41460	904	500	Switzerland
45556	908	500	Switzerland

Note: Although the CCSIDs for the DBCS languages (Japanese, Korean, and Chinese) represent both SBCS and DBCS character sets and code pages, only the SBCS character set and code page are involved in the EXTENDED CODE PAGE support in ISPF.

Extended Code Page Translate Tables Provided by ISPF

The translate tables provided by ISPF that can be updated by the user are as follows:

- ISPSTC1 (CCSID=00037 / 01140 U.S.A., Canada, Netherlands, Portugal, Brazil, Australia, New Zealand)
- ISPSTC2 (CCSID=00273 / 01141 Austria and Germany)
- ISPSTC3 (CCSID=00277 / 01142 Denmark and Norway)
- ISPSTC4 (CCSID=00278 / 01143 Finland and Sweden)
- ISPSTC5 (CCSID=00280 / 01144 Italy)
- ISPSTC6 (CCSID=00284 / 01145 Spain and Spanish-Speaking)
- ISPSTC7 (CCSID=00285 / 01146 United Kingdom)
- ISPSTC8 (CCSID=00297 / 01147 France)
- ISPSTC9 (CCSID=00500 / 01148 Switzerland and Belgium)
- ISPSTC10 (CCSID=00939 Japan (Latin))
- ISPSTC11 (CCSID=00930 Japan (Katakana))
- ISPSTC12 (CCSID=00933 Korea)
- ISPSTC13 (CCSID=00935 Simplified Chinese)
- ISPSTC14 (CCSID=00937 Traditional Chinese)
- ISPSTC15 (CCSID=00870 Latin-2)
- ISPSTC16 (CCSID=00880 Cyrillic)
- ISPSTC17 (CCSID=01025 Cyrillic)
- ISPSTC18 (CCSID=00420 Arabic)
- ISPSTC19 (CCSID=00424 Hebrew)
- ISPSTC20 (CCSID=00838 Thai)
- ISPSTC21 (CCSID=00871 / 1149 Iceland)
- ISPSTC22 (CCSID=00875 Greek)
- ISPSTC23 (CCSID=01026 Turkish).

The source for the above modules is provided in the SYS1.SAMPLIB library in the MVS environment.

Example of User-Modifiable ISPF Translate Table

The following is the module for CCSID 00037 (ISPSTC1). The existing tables can be modified, or more pairs of direct translation tables can be added. To add direct translation tables, add a new ISPCCSID macro call for the new direct translate tables, and add the new tables. Rename the assembler program to ISPTTCx(x), where x(x) is the last 1- or 2-digit number of the ISPSTCx(x) name. For example, ISPSTC1 should be renamed ISPTTC1, and ISPSTC14 renamed ISPTTC14.

```
*  THE FOLLOWING MACROS WILL GENERATE THE CCSID 00037 MODULE.
*
*
         ISPCCSID CCSID=00037,TO=TTC1T5H,FROM=TTC1F5H
         ISPCCSID CCSID=08229,TO=TTC1TB1,FROM=TTC1FB2
         ISPCCSID CCSID=04371,TO=TTC1TB2,FROM=TTC1FB2
*
*    TTC1T5H - CCSID 00037 TO CCSID 00500 Table
*
TTC1T5H  DS   0XL191
         DC X'4041424344454647'          (X'40' TO X'47')
```

```
         DC X'4849B04B4C4D4EBB'          (X'48' TO X'4F')
         DC X'5051525354555657'          (X'50' TO X'57')
         DC X'58594F5B5C5D5EBA'          (X'58' TO X'5F')
         DC X'6061626364656667'          (X'60' TO X'67')
         DC X'68696A6B6C6D6E6F'          (X'68' TO X'6F')
         DC X'7071727374757677'          (X'70' TO X'77')
         DC X'78797A7B7C7D7E7F'          (X'78' TO X'7F')
         DC X'8081828384858687'          (X'80' TO X'87')
         DC X'88898A8B8C8D8E8F'          (X'88' TO X'8F')
         DC X'9091929394959697'          (X'90' TO X'97')
         DC X'98999A9B9C9D9E9F'          (X'98' TO X'9F')
         DC X'A0A1A2A3A4A5A6A7'          (X'A0' TO X'A7')
         DC X'A8A9AAABACADAEAF'          (X'A8' TO X'AF')
         DC X'5FB1B2B3B4B5B6B7'          (X'B0' TO X'B7')
         DC X'B8B94A5ABCBDBEBF'          (X'B8' TO X'BF')
         DC X'C0C1C2C3C4C5C6C7'          (X'C0' TO X'C7')
         DC X'C8C9CACBCCCDCECF'          (X'C8' TO X'CF')
         DC X'D0D1D2D3D4D5D6D7'          (X'D0' TO X'D7')
         DC X'D8D9DADBDCDDDEDF'          (X'D8' TO X'DF')
         DC X'E0E1E2E3E4E5E6E7'          (X'E0' TO X'E7')
         DC X'E8E9EAEBECEDEEEF'          (X'E8' TO X'EF')
         DC X'F0F1F2F3F4F5F6F7'          (X'F0' TO X'F7')
         DC X'F8F9FAFBFCFDFE'            (X'F8' TO X'FE')
*
*    TTC1F5H - CCSID 00037 FROM CCSID 00500 Table
*
TTC1F5H  DS   0XL191
         DC X'4041424344454647'          (X'40' TO X'47')
         DC X'4849BA4B4C4D4E5A'          (X'48' TO X'4F')
         DC X'5051525354555657'          (X'50' TO X'57')
         DC X'5859BB5B5C5D5EB0'          (X'58' TO X'5F')
         DC X'6061626364656667'          (X'60' TO X'67')
         DC X'68696A6B6C6D6E6F'          (X'68' TO X'6F')
         DC X'7071727374757677'          (X'70' TO X'77')
         DC X'78797A7B7C7D7E7F'          (X'78' TO X'7F')
         DC X'8081828384858687'          (X'80' TO X'87')
         DC X'88898A8B8C8D8E8F'          (X'88' TO X'8F')
         DC X'9091929394959697'          (X'90' TO X'97')
         DC X'98999A9B9C9D9E9F'          (X'98' TO X'9F')
         DC X'A0A1A2A3A4A5A6A7'          (X'A0' TO X'A7')
         DC X'A8A9AAABACADAEAF'          (X'A8' TO X'AF')
         DC X'4AB1B2B3B4B5B6B7'          (X'B0' TO X'B7')
         DC X'B8B95F4FBCBDBEBF'          (X'B8' TO X'BF')
         DC X'C0C1C2C3C4C5C6C7'          (X'C0' TO X'C7')
         DC X'C8C9CACBCCCDCECF'          (X'C8' TO X'CF')
         DC X'D0D1D2D3D4D5D6D7'          (X'D0' TO X'D7')
         DC X'D8D9DADBDCDDDEDF'          (X'D8' TO X'DF')
         DC X'E0E1E2E3E4E5E6E7'          (X'E0' TO X'E7')
         DC X'E8E9EAEBECEDEEEF'          (X'E8' TO X'EF')
         DC X'F0F1F2F3F4F5F6F7'          (X'F0' TO X'F7')
         DC X'F8F9FAFBFCFDFE'            (X'F8' TO X'FE')
*
*    TTC1TB1 - CCSID 00037 TO CCSID 08229 Table
*
TTC1TB1  DS   0XL191
         DC X'404B4B4B4B4B4B4B'          (X'40' TO X'47')
         DC X'4B4B4A4B4C4D4E4F'          (X'48' TO X'4F')
         DC X'504B4B4B4B4B4B4B'          (X'50' TO X'57')
         DC X'4B4B5A5B5C5D5E5F'          (X'58' TO X'5F')
         DC X'60614B4B4B4B4B4B'          (X'60' TO X'67')
         DC X'4B4B6A6B6C6D6E6F'          (X'68' TO X'6F')
         DC X'4B4B4B4B4B4B4B4B'          (X'70' TO X'77')
         DC X'4B797A7B7C7D7E7F'          (X'78' TO X'7F')
         DC X'4B81828384858687'          (X'80' TO X'87')
         DC X'88894B4B4B4B4B4B'          (X'88' TO X'8F')
         DC X'4B91929394959697'          (X'90' TO X'97')
         DC X'98994B4B4B4B4B4B'          (X'98' TO X'9F')
```

```
         DC X'4BA1A2A3A4A5A6A7'          (X'A0' TO X'A7')
         DC X'A8A94B4B4B4B4B4B'          (X'A8' TO X'AF')
         DC X'4B4B4B4B4B4B4B4B'          (X'B0' TO X'B7')
         DC X'4B4B4B4B4B4B4B4B'          (X'B8' TO X'BF')
         DC X'C0C1C2C3C4C5C6C7'          (X'C0' TO X'C7')
         DC X'C8C94B4B4B4B4B4B'          (X'C8' TO X'CF')
         DC X'D0D1D2D3D4D5D6D7'          (X'D0' TO X'D7')
         DC X'D8D94B4B4B4B4B4B'          (X'D8' TO X'DF')
         DC X'E04BE2E3E4E5E6E7'          (X'E0' TO X'E7')
         DC X'E8E94B4B4B4B4B4B'          (X'E8' TO X'EF')
         DC X'F0F1F2F3F4F5F6F7'          (X'F0' TO X'F7')
         DC X'F8F94B4B4B4B4B'            (X'F8' TO X'FE')
*
*    TTC1FB1 - CCSID 00037 FROM CCSID 08229 Table
*
TTC1FB1  DS   0XL191
         DC X'4041424344454647'          (X'40' TO X'47')
         DC X'48494A4B4C4D4E4F'          (X'48' TO X'4F')
         DC X'5051525354555657'          (X'50' TO X'57')
         DC X'58595A5B5C5D5E5F'          (X'58' TO X'5F')
         DC X'6061626364656667'          (X'60' TO X'67')
         DC X'68696A6B6C6D6E6F'          (X'68' TO X'6F')
         DC X'7071727374757677'          (X'70' TO X'77')
         DC X'78797A7B7C7D7E7F'          (X'78' TO X'7F')
         DC X'8081828384858687'          (X'80' TO X'87')
         DC X'88898A8B8C8D8E8F'          (X'88' TO X'8F')
         DC X'9091929394959697'          (X'90' TO X'97')
         DC X'98999A9B9C9D9E9F'          (X'98' TO X'9F')
         DC X'A0A1A2A3A4A5A6A7'          (X'A0' TO X'A7')
         DC X'A8A9AAABACADAEAF'          (X'A8' TO X'AF')
         DC X'B0B1B2B3B4B5B6B7'          (X'B0' TO X'B7')
         DC X'B8B9BABBBCBDBEBF'          (X'B8' TO X'BF')
         DC X'C0C1C2C3C4C5C6C7'          (X'C0' TO X'C7')
         DC X'C8C9CACBCCCDCECF'          (X'C8' TO X'CF')
         DC X'D0D1D2D3D4D5D6D7'          (X'D0' TO X'D7')
         DC X'D8D9DADBDCDDDEDF'          (X'D8' TO X'DF')
         DC X'E0E1E2E3E4E5E6E7'          (X'E0' TO X'E7')
         DC X'E8E9EAEBECEDEEEF'          (X'E8' TO X'EF')
         DC X'F0F1F2F3F4F5F6F7'          (X'F0' TO X'F7')
         DC X'F8F9FAFBFCFDFE'            (X'F8' TO X'FE')
*
*    TTC1TB2 - CCSID 00037 TO CCSID 04371 Table
*
TTC1TB2  DS   0XL191
         DC X'404B4B4B4B4B794B'          (X'40' TO X'47')
         DC X'4B4B4B4B4C4D4E4B'          (X'48' TO X'4F')
         DC X'50D04B4B4B4B4B4B'          (X'50' TO X'57')
         DC X'4B4B4F5A5C5D5E4B'          (X'58' TO X'5F')
         DC X'60614B4B4B4B7C4B'          (X'60' TO X'67')
         DC X'5B4B4B6B6C6D6E6F'          (X'68' TO X'6F')
         DC X'4B4A4B4B4B4B4B4B'          (X'70' TO X'77')
         DC X'4B4B7A4B4B7D7E7F'          (X'78' TO X'7F')
         DC X'4B81828384858687'          (X'80' TO X'87')
         DC X'88894B4B4B4B4B4B'          (X'88' TO X'8F')
         DC X'4B91929394959697'          (X'90' TO X'97')
         DC X'98994B4B4B4B4B4B'          (X'98' TO X'9F')
         DC X'4BA1A2A3A4A5A6A7'          (X'A0' TO X'A7')
         DC X'A8A94B4B4B4B4B4B'          (X'A8' TO X'AF')
         DC X'5F44B4BB4B4B4B4B'          (X'B0' TO X'B7')
         DC X'4B4B4B4B4B4B4B4B'          (X'B8' TO X'BF')
         DC X'4BC1C2C3C4C5C6C7'          (X'C0' TO X'C7')
         DC X'C8C94B4B4B4B4BC0'          (X'C8' TO X'CF')
         DC X'4BD1D2D3D4D5D6D7'          (X'D0' TO X'D7')
         DC X'D8D94B4B4B4B4B4B'          (X'D8' TO X'DF')
         DC X'E04BE2E3E4E5E6E7'          (X'E0' TO X'E7')
         DC X'E8E94B4B4B4B4B7B'          (X'E8' TO X'EF')
         DC X'F0F1F2F3F4F5F6F7'          (X'F0' TO X'F7')
```

```
         DC X'F8F94B4B4B4B4B'            (X'F8' TO X'FE')
*
*    TTC1FB2 - CCSID 00037 FROM CCSID 04371 Table
*
TTC1FB2  DS   0XL191
         DC X'4041424344454647'          (X'40' TO X'47')
         DC X'4849714B4C4D4E5A'          (X'48' TO X'4F')
         DC X'5051525354555657'          (X'50' TO X'57')
         DC X'58595B685C5D5EB0'          (X'58' TO X'5F')
         DC X'6061626364656667'          (X'60' TO X'67')
         DC X'6869486B6C6D6E6F'          (X'68' TO X'6F')
         DC X'7071727374757677'          (X'70' TO X'77')
         DC X'78467AEF667D7E7F'          (X'78' TO X'7F')
         DC X'8081828384858687'          (X'80' TO X'87')
         DC X'88898A8B8C8D8E8F'          (X'88' TO X'8F')
         DC X'9091929394959697'          (X'90' TO X'97')
         DC X'98999A9B9C9D9E9F'          (X'98' TO X'9F')
         DC X'A0A1A2A3A4A5A6A7'          (X'A0' TO X'A7')
         DC X'A8A9AAABACADAEAF'          (X'A8' TO X'AF')
         DC X'B0B1B2B3B4B5B6B7'          (X'B0' TO X'B7')
         DC X'B8B9BABBBCBDBEBF'          (X'B8' TO X'BF')
         DC X'CFC1C2C3C4C5C6C7'          (X'C0' TO X'C7')
         DC X'C8C9CACBCCCDCECF'          (X'C8' TO X'CF')
         DC X'51D1D2D3D4D5D6D7'          (X'D0' TO X'D7')
         DC X'D8D9DADBDCDDDEDF'          (X'D8' TO X'DF')
         DC X'E0E1E2E3E4E5E6E7'          (X'E0' TO X'E7')
         DC X'E8E9EAEBECEDEEEF'          (X'E8' TO X'EF')
         DC X'F0F1F2F3F4F5F6F7'          (X'F0' TO X'F7')
         DC X'F8F9FAFBFCFDFE'            (X'F8' TO X'FE')
         END
```

Appendix A. Character Translations for APL, TEXT, and Katakana

This appendix contains the character translation tables for APL, TEXT, and Katakana. This information does not include Extended Code Page Support. See "Chapter 10. Extended Code Page Support" on page 313.

ISPF permits use of all keyboards for all models of 3270 and 3290 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 88 on page 328 and Figure 89 on page 329. ISPF also permits use of 3277 and 3278 Japanese Katakana terminals. ISPF does not permit the use of 3277 and 3278 Katakana terminals and an APL terminal at the same time.

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. Input and output fields and model line fields are not affected by use of the KANA keyword. Rules for display of text fields are as follows:

- 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 nondisplayable 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.

See "How to Define a Message" on page 292 for a description of how the KANA keyword provides a similar function for messages containing lowercase characters that must be displayed on a Katakana terminal.

Note: The KANA keyword is not needed for panels and messages that specify a CCSID for Extended Code Page Support. See "Chapter 10. Extended Code Page Support" on page 313.

© Copyright IBM Corp. 1980, 1999

3278 only; invalid character on 3277.

National use character.
Graphics shown are for U.S. keyboards;
graphics differ in other countries.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00																
10																
20																
30																
40	sp	A	B	C	D	E	F	G	H	I	¢	.	<	(	+	|
50	&	J	K	L	M	N	O	P	Q	R	!	$	*	)	;	¬
60	_	/	S	T	U	V	W	X	Y	Z	¦	,	%	_	>	?
70		^	¨						v	`	:	#	@	'	=	"
80	~	a	b	c	d	e	f	g	h	i	↑	↓	≤	⌈	⌊	→
90	□	j	k	l	m	n	o	p	q	r	⊃	⊂		o		←
A0	¯	~	s	t	u	v	w	x	y	z	∩	∪	⊥	[	≥	∘
B0	α	∈	ι	ρ	ω		×	\	÷		∇	∆	⊤	]	≠	|
C0	{	A	B	C	D	E	F	G	H	I	⍨	⍩		⌽		⍉
D0	}	J	K	L	M	N	O	P	Q	R	⌶	!	⍱	⍋	⍞	⍈
E0	\		S	T	U	V	W	X	Y	Z	≠	⍲		⊖	⍇	⍑
F0	0	1	2	3	4	5	6	7	8	9		⍨	⍋	⍟	⍎	

Figure 88. Internal Character Representations for APL Keyboards

National use character.
Graphics shown are for U.S. keyboards;
graphics differ in other countries.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00																
10																
20																
30																
40	sp										¢	.	<	(	+	|
50	&	1	2	3						↓	!	$	*	)	;	¬
60	—	/									¦	,	%	_	>	?
70	n	o								`	:	#	@	'	=	"
80		a	b	c	d	e	f	g	h	i	↑	{	≤	(	+	+
90	□	j	k	l	m	n	o	p	q	r		}	¤	)	±	■
A0	¯	~	s	t	u	v	w	x	y	z	⊛	⌊	⌈	[	≥	•
B0	0	1	2	3	4	5	6	7	8	9	▽	⌋	⌉	]	≠	—
C0	{	A	B	C	D	E	F	G	H	I	△	⊥	⊤	∨	≠	⍲
D0	}	J	K	L	M	N	O	P	Q	R	⍞	⍙	§	¶	←	→
E0	\	\	S	T	U	V	W	X	Y	Z	⍨	⊢	⊣	⍋	⍒	∧
F0		1	2	3	4	5	6	7	8	9	|	⌊	⍕	⍎	⌈	

Figure 89. Internal Character Representations for Text Keyboards

Appendix B. ISPTTDEF Specify Translate Table Set

ISPF provides a program, ISPTTDEF, that can be used for specifying the set of terminal translate tables to be used. This lets you specify private sets of translate tables.

Note: This program is not used for Extended Code Page Support translate tables. See "Chapter 10. Extended Code Page Support" on page 313.

You can invoke ISPTTDEF from a selection panel, as a command, or from a dialog function. The format of the ISPTTDEF program call is:

```
SELECT PGM(ISPTTDEF) PARM(xxx)
```

where xxx is the terminal type or the name of the load module containing translate tables.

Return codes from invoking ISPTTDEF are as follows:

0 Normal completion
4 Translate tables could not be loaded

Valid terminal types are those that can be specified using the ISPF Settings panel. If the name specified is not a valid terminal type, ISPF attempts to load a module having that name.

© Copyright IBM Corp. 1980, 1999

Appendix C. Diagnostic Tools and Information

This chapter contains debugging tools, diagnostic information, and common problems that can occur while using ISPF.

ISPF Debug Tools

The following tools ship with ISPF as samples.

ISRABEND
A CLIST that provides a step-by-step explanation of how to diagnose an abend interactively. It uses TSO TEST to gather the information that the IBM support organization normally requires.

ISRCSECT
A REXX exec used in conjunction with ISRTCB exec. It takes the entry point of a load module and begins searching for a specific CSECT. If it finds one, the exec displays the CSECT's eye-catcher.

ISRFIND
A REXX exec that issues a LISTA STATUS and searches for a specified member or load module. Also, the exec optionally calls AMBLIST to check the MODIFIED, FIXED, and PAGEABLE LPAs and checks LPALIST and LNKLST (pointed to by system control blocks) for the specified load module. If invoked under ISPF, the information is displayed via an ISPF table display (panel ISRFINDP) and allows the user to BROWSE or EDIT the specified member.

ISRPOINT
A REXX exec used in conjunction with the ISRTCB exec. This exec uses the entry point address obtained from ISRTCB and lists the CSECT eye-catchers associated with that load module.

ISRTCB
A REXX exec that emulates the TSO TEST command LISTMAP. It lists the TCBs and the load modules (with their entry points) associated with each TCB, without using TSO TEST.

ISRTEST
A CLIST that uses TSO TEST to load the job pack area (JPA) and set breakpoints on entry to a specific ISPF or PDF CSECT. This allows for the verification of the compile date associated with the CSECT with the most recent maintenance level for that version or release. Additionally, you can modify this sample to set specific breakpoints within the CSECT to identify the failing instruction.

Diagnostic Information

This section is intended to help you gather information in order to diagnose ISPF problems.

© Copyright IBM Corp. 1980, 1999

Using the ENVIRON System Command

ISPF provides the ENVIRON command to assist you in gathering data that can be helpful in diagnosing problems, thus reducing service time. The ISPF session does not have to be running in any ISPF TEST/TRACE mode when you use the ENVIRON command.

The ENVIRON command can help you:

- Produce system abend dumps when not running in ISPF TEST mode (ENBLDUMP parameter)
- Trace the TPUT, TGET, and PUTLINE buffers and obtain dump information for TPUT and TGET errors (TERMTRAC parameter)
- Gather terminal status information (TERMSTAT parameter)

You can display a panel (Figure 90) for selecting command options by entering the ENVIRON command with no parameters, or display the panel through the use of the Environ settings... choice from the Environ pull-down on the ISPF Settings panel. This panel includes the current values of the ENVIRON command parameters (ENBLDUMP and TERMTRAC) and the ddname, if any, allocated for a dump data set. The values can be changed by entering new values directly on the panel.

```
  Log/List  Function keys  Colors  Environ  Temporary  Help
- ------------------------------ ISPF Settings ------------------------------ --
                      ISPF ENVIRON Command Settings                            +
S   Enter "/" to select option
    _  Enable a dump for a subtask abend when not in ISPF TEST mode

    Terminal Tracing (TERMTRAC)
      Enable . . . _  1. Enable terminal tracing (ON)
                      2. Enable terminal tracing when a terminal error
                         is encountered (ERROR)
                      3. Disable terminal tracing (OFF)
      DDNAME . . . ISPSNAP   (DDNAME for TERMTRAC ON, ERROR, or DUMP.)

T   Terminal Status (TERMSTAT)
      Enable . . . _  1. Yes, invoke TERMSTAT immediately
                      2. Query terminal information
                      3. No
    Command ===> ______________________________________________________
     F1=Help       F2=Split      F3=Exit       F7=Backward   F8=Forward
     F9=Swap      F12=Cancel
C                                                                          __
 F1=Help      F2=Split     F3=Exit      F7=Backward  F8=Forward   F9=Swap
F10=Actions  F12=Cancel
```

Figure 90. ENVIRON Settings Panel (ISPENVA)

You can issue the ENVIRON command at any time during an ISPF session.

ENVIRON Command Syntax and Parameter Descriptions

The general syntax for the ENVIRON command is:

```
ENVIRON [ENBLDUMP [ON|OFF]]
        [TERMTRAC [ON|ERROR|DUMP|OFF]]
        [TERMSTAT [QUERY]]
```

The parameter descriptions for the ENVIRON command are as follows:

ENBLDUMP

Specifying the ENBLDUMP parameter enables ISPF to produce an abend dump if a subtask abnormally terminates when ISPF is not running in TEST mode (as required prior to ISPF Version 2.3, and documented in *ISPF Dialog Developer's Guide and Reference*). The ENBLDUMP parameter does not apply to attached commands. Prior to the time that a dump is taken you must allocate either the SYSUDUMP, SYSMDUMP, or SYSABEND ddname. For more information about these data sets, refer to *MVS Diagnostic Techniques*

The default value for the ENBLDUMP parameter is ON. ENVIRON ENBLDUMP ON specifies to ISPF that a dump for the abending subtask is to be generated.

Issuing ENVIRON ENBLDUMP OFF cancels the effect of the ON status.

The ENBLDUMP parameter value is preserved across ISPF sessions in the ISPSPROF profile.

With ENBLDUMP active, even when ISPF is not running in TEST mode, abnormal termination of a subtask results in a dump being taken and control being returned to TSO. ISPF execution is not resumed.

When running in ISPF TEST mode, issuing ENVIRON ENBLDUMP has no effect on dump processing.

TERMTRAC

Specifying the TERMTRAC parameter allows you to trace all terminal input and output data (TPUT, TGET, PUTLINE) during an ISPF session. The TERMTRAC parameter also allows you to turn on in-core tracing and cause ISPF to produce a SNAP dump if the TPUT or TGET service results in an error. ISPF does not have to be running in TSO TEST mode.

Note: The ENVIRON TERMTRAC buffer does not include:

- The TPUT/TGET instructions issued to query the terminal:
  - At ISPF initialization
  - By the ENVIRON TERMSTAT command
- The TPUT instruction issued to clear the screen at ISPF termination
- Under certain severe ISPF error conditions, the TPUT instruction issued to display a severe error line message

Before issuing the ENVIRON TERMTRAC DUMP command you must have first issued the ENVIRON TERMTRAC ON or ENVIRON TERMTRAC ERROR command.

Diagnostic Information

Prior to using the TERMTRAC option, you must define to ISPF the ddname for the data set to be used for the SNAP macro, which ISPF invokes to provide data stream dumps. The ddname can be defined by specifying it on the panel displayed as a result of either issuing the ENVIRON command with no parameters, or selecting the Environ settings... choice from the Environ pull-down on the ISPF Settings panel. You must follow the data set characteristics guidelines defined by MVS for the SNAP macro. See *MVS/XA Supervisor Services and Macro Instructions* for DCB information that can be specified for the SNAP ddname.

The terminal data stream buffer used for ENVIRON TERMTRAC data collection is not reset to zeroes.

Subparameters define terminal data tracing as follows:

- *ENVIRON TERMTRAC ON*

  Activates TPUT, TGET, and PUTLINE buffer tracing of the terminal data stream. All data is retained in a 24K buffer provided by ISPF. No buffer entry is fragmented. If an entry will not fit into the remaining buffer space, ISPF issues a SNAP to capture the buffer data. The next trace entry is stored at the top of the buffer, regardless of the status of the SNAP execution.

  Messages are displayed to the user only for errors during SNAP execution. No messages are displayed during dumps taken as a result of the data buffer filling.

  Because ENVIRON TERMTRAC ON causes a SNAP dump to be taken each time the buffer fills, the ddname that you allocate for the SNAP macro should have a disposition of MOD. This assures that no trace data is lost.

  The layout of the terminal data buffer for all SNAP dumps is:

```
1 TPUT/TGET/PUTLINE BUFFER TRACE
  2 Header of 8 bytes initialized to
    TERMTRAC
  2 4-byte pointer to where the next entry
    is to be placed
  2 Reserved (20 bytes, for 32-byte boundary
    alignment)
  2 TPUT/TGET/PUTLINE DATA (*)
    3 8-byte TPUT/TGET/PUTLINE identifier
    3 4-byte pointer to previous entry
    3 Information specific to the terminal
      type identifier.
```

  The TPUT/TGET identifiers and specific information for each is as follows. Each buffer entry is aligned on a 32-byte boundary.

  TGET ‘TGET ’ –prior to issuing TGET SVC. 4-byte pointer to previous entry. General purpose registers 0, 1, and 15:

```
R0  = input data area size
R1  = input data area pointer
R15 = TGET option byte
```

  TGETR

  ‘TGETR ’ –return from TGET SVC. 4-byte pointer to previous entry. General purpose registers 1 and 15:

```
R1  = input data length
R15 = TGET return code
```

  Four-byte length of data stream. Data stream.

TPUT 'TPUT ' –prior to issuing edit TPUT macro. 4-byte pointer to previous entry. General purpose registers 0, 1, and 15:

```
R0  = output data area
R1  = output data area pointer
R15 = TPUT option byte
```

4-byte length of data stream. Data stream.

TPUTR
'TPUTR ' –return from edit TPUT macro. 4-byte pointer to previous entry. General purpose register 15:

```
R15 = TPUT return code
```

TPUTNE
'TPUTNE ' –prior to issuing the noedit TPUT macro. 4-byte pointer to previous entry. General purpose registers 0, 1, and 15:

```
R1  = address of plist
R15 = TPUT option byte
```

16-byte noedit plist:

```
Reserved (2 bytes)
2-byte length of data stream
Code (1 byte)
3-byte addr of data stream
Reserved (8 bytes)
```

Data stream.

TPUTNER
'TPUTNER ' –return from noedit TPUT macro. 4-byte pointer to previous entry. General purpose register 15:

```
R15 = TPUT return code
```

PUTLINE
'PUTLINE ' –prior to issuing the PUTLINE macro. 4-byte pointer to previous entry 12-byte PUTLINE parameter block:

```
Control flags (2 bytes)
2-byte TPUT options field
4-byte address of message
4-byte address of format-only line
```

125-byte message description:

```
2-byte message length
2-byte message offset
121-byte message
```

Actions that occur as a result of issuing the ENVIRON TERMTRAC command when ENVIRON TERMTRAC ON is already in effect are listed by command subparameter below:

ON ENVIRON TERMTRAC ON continues to function normally.

OFF Tracing is turned off and ISPF issues a SNAP macro. If ENVIRON TERMTRAC tracing is requested again, the next entry is written at the top of the buffer, regardless of whether the prior SNAP was successful.

ERROR
Changes the setting of the command to ENVIRON TERMTRAC

ERROR. Tracing continues, with the next buffer entry being written after the last entry written by the ENVIRON TERMTRAC ON setting.

DUMP
The ENVIRON TERMTRAC ON condition continues. In addition, ISPF issues a SNAP macro and, if the SNAP is successful, the next trace entry is written at the top of the buffer. If the SNAP fails, the next entry is written after the last entry prior to the SNAP.

- *ENVIRON TERMTRAC ERROR*

  Initiates tracing of the TPUT, TGET, and PUTLINE buffers. In addition, it causes ISPF to initiate an MVS SNAP dump if a TPUT or TGET error occurs. The dump includes the storage trace buffer, the current TCB, all system control program information, and all problem program information. The MVS SNAP macro definition provides more specific information about the areas dumped when all system control program and problem program information is requested.

  ISPF issues the SNAP macro on the first occurrence of a TPUT failure. ISPF makes three consecutive attempts to correct a TPUT error.

  Before using this option, you must have defined the ddname for the SNAP macro as described earlier in this topic under TERMTRAC.

  Actions that occur as a result of issuing the ENVIRON TERMTRAC command when ENVIRON TERMTRAC ERROR is already in effect are listed by command subparameter below:

  ON Changes the setting of the command to ENVIRON TERMTRAC ON. Tracing continues, with the next buffer entry being written after the last entry written by the ENVIRON TERMTRAC ON setting.

  ERROR
  ENVIRON TERMTRAC ERROR continues to function normally, with the next trace entry written after the last ERROR trace entry.

  OFF The setting for ENVIRON TERMTRAC is set to OFF. If ENVIRON TERMTRAC tracing is requested again, the next entry is written at the top of the buffer, regardless of whether the prior SNAP was successful.

  DUMP
  The ENVIRON TERMTRAC ERROR condition continues. In addition, ISPF issues a SNAP macro and, if the SNAP is successful, the next trace entry is written at the top of the buffer. If the SNAP fails, the next entry is written after the last entry prior to the SNAP.

- *ENVIRON TERMTRAC DUMP*

  Causes ISPF to immediately issue a SNAP macro, but only if ENVIRON TERMTRAC ON or ENVIRON TERMTRAC ERROR is active. The resulting dump includes the storage trace buffer, the current TCB, all system control program information, and all problem program information. The MVS SNAP macro definition provides more specific information about the areas dumped when all system control program and problem program information is requested.

  Notes:

  1. This command execution does not turn off terminal data stream tracing if it is active at the time.

2. The next entry is written to the top of the terminal data buffer if the SNAP was successful; otherwise, tracing continues immediately after the last trace buffer entry.

- *ENVIRON TERMTRAC OFF*

  Resets active ENVIRON TERMTRAC ON and ENVIRON TERMTRAC ERROR commands. If ENVIRON TERMTRAC is active, ISPF issues a SNAP macro.

The TERMTRAC parameter value is preserved across ISPF sessions in the ISPSPROF profile. The ddname specified for TERMTRAC on the ENVIRON option panel is also saved across sessions.

TERMSTAT

Specifying the TERMSTAT option of the ENVIRON command allows you to collect information about the characteristics of the terminal you are using and the line to which it is attached. The information is returned to your terminal by using line mode, and is written to the ISPF log data set.

The description below of the information returned from an ENVIRON TERMSTAT request is divided into three parts:

- A list of terminal characteristics as defined in ISPF variables. In other words, this list defines what ISPF thinks your terminal characteristics are.
- A list of terminal characteristics as defined within TSO.
- A list of structured fields that apply only to terminals with extended data stream (EDS) capability.

If you issue ENVIRON TERMSTAT (without the QUERY parameter) ISPF unconditionally returns information from lists A and B (below). In addition, if your terminal is connected to a port that supports extended data streams, ISPF returns information from list C (below).

If your terminal is one that supports extended data streams, such as an IBM 3279, but is connected to a non-EDS port, you can issue ENVIRON TERMSTAT QUERY to force ISPF to return information from list C. Be aware that if you issue ENVIRON TERMSTAT QUERY, and your terminal is not a type that supports extended data streams, such as the IBM 3277, you will receive an ORDER STREAM CHECK error.

Information returned as a result of issuing the ENVIRON TERMSTAT command is as follows:

List A – Terminal Characteristics as Defined Within ISPF

```
14-bit terminal addressing mode (ON or OFF)
16-bit terminal addressing mode (ON or OFF)
Color mode (ON or OFF)
Highlighting mode (ON or OFF)
DBCS mode (ON or OFF)
Primary screen size (length, width, total bytes)
Alternate screen size (length, width, total bytes)
Partition screen size (length, width, total bytes)
ISPF terminal buffer data (TSB ptr., TSB size,
 TPP addr.)
```

List B – Terminal Characteristics as Defined Within TSO

```
Return code from GTTERM
Primary screen information (rows, columns)
Alternate screen information (rows, columns)
```

```
Screen attribute value
Character set (ASCII or EBCDIC)
Extended data streams or non-EDS support
Return code from GTSIZE
GTSIZE information (rows, columns)
Access method being used (VTAM* or TCAM)
```

List C – Terminals Supporting EDS (structured fields)

```
Usable areas
Partitions
Character sets
Color
Highlighting
Reply modes
PC 3270
Implicit partition
Input control
Field rule
```

- *ENVIRON TERMSTAT QUERY*

  The QUERY parameter allows you to request terminal data related to extended data stream capability, even though your terminal is connected to a port that does not support extended data streams.

Abend Panels Provide Diagnostic Information

When ISPF processing ends abnormally, diagnostic panels are available for displaying:
- Task abend code
- Reason code
- Module name
- Entry point address
- Program-Status Word (PSW)
- Register content at the time of the abend

This information is used in logged abend messages. A tutorial panel displays a list of the common abend codes.

On abnormal ISPF termination, the Error Recovery panel shown in Figure 91 on page 341 indicates the abend code and reason code.

```
                          Error Recovery

      * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
      * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
      * *              ISPF processor ended abnormally                 * *
      * *                                                              * *
      * *                                                              * *
      * *                        Reason code                           * *
      * *                                                              * *
      * *                                                              * *
      * *                                                              * *
      * *  NOTE: The ABEND and REASON codes displayed above are        * *
      * *        HEXADECIMAL values for "SYSTEM" abends and DECIMAL    * *
      * *        values for "USER" abends.                             * *
      * *                                                              * *
      * *  Enter HELP command for list of common ABEND codes.          * *
      * *  Press ENTER key for additional DIAGNOSTIC information.      * *
      * *  Enter END command to display primary option menu.           * *
      * *                                                              * *
      * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
      * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Command ===> ____________________________________________________________
 F1=Help    F2=Split   F3=Exit    F9=Swap   F12=Cancel
```

Figure 91. Error Recovery Panel (ISPPRS1)

If you are not running in an MVS/XA environment, the reason code will be blank. If you are running in an MVS/XA environment and the SDWA (System Diagnostic Work Area) Reason Code is not supplied, that is, the SDWA reason code flag bit is OFF, the Reason Code panel field will be blank. If the abend code documentation indicates that the reason code is in a particular register, see the contents of that register, which can be displayed on the Additional Diagnostic Information panel as shown in Figure 93 on page 342.

If you enter HELP, the panel shown in Figure 92 on page 342 displays a list of the common abend codes.

```
TUTORIAL -------------------- COMMON ABEND CODES --------------------- TUTORIAL

                              COMMON ABEND CODES

   The following list contains some common ABEND codes.  For more information
   about these codes and for information about other abend codes, see the
   appropriate MVS completion code manual.  For abends resulting from
   other products, refer to the appropriate product's message library.


001 - I/O ERROR                        706 - NON-EXECUTABLE PROGRAM
002 - I/O INVALID RECORD               804 - INSUFFICIENT VIRTUAL STORAGE
004 - OPEN ERROR                       806 - UNABLE TO LOAD (LINK ETC) PROGRAM
008 - I/O SYNAD ERROR                  80A - INSUFFICIENT VIRTUAL STORAGE
013 - OPEN ERROR                       878 - INSUFFICIENT VIRTUAL STORAGE
028 - PAGING I/O ERROR                 737 - I/O ERROR
0CX - PROGRAM CHECK EXCEPTIONS:        A14 - I/O ERROR
  0C1 - OPERATION,                     B37 - INSUFFICIENT DASD SPACE
  0C4 - PROTECTION / ADDRESSING,       D37 - INSUFFICIENT DASD SPACE
  0C5 - ADDRESSING,                    E37 - INSUFFICIENT DASD SPACE
COMMAND ===> ______________________________________________________________
 F1=Help      F2=Split     F3=Exit      F7=Backward  F8=Forward   F9=Swap
F10=Left     F11=Right    F12=Cancel
```

Figure 92. Common Abend Codes (ISP93010)

To return to the Error Recovery panel, enter `end` from the Common ABEND panel.

If you press Enter from the Error Recovery panel, the panel shown in Figure 93 is displayed:

```
                         Additional Diagnostic Information

                      System Abend code = 806
                            Reason code =

           ISPF Release Level   4.1

           Module name          *Not specified*
           Entry point address  00000000

           PSW                  FF850008 4000DB8E

           Register content:

           R0  00806000   R1  80806000   R2  000526E0   R3  0009DDA0
           R4  00000058   R5  008FD568   R6  008C7850   R7  000526E0
           R8  0009DD30   R9  008C7850   R10 81005342   R11 00000C00
           R12 00000000   R13 008C7850   R14 810056C6   R15 0000001C


   Enter HELP command for list of common ABEND Codes.
   Enter END command to display primary option menu.
Command ===> ______________________________________________________________
 F1=Help    F2=Split   F3=Exit    F9=Swap   F12=Cancel
```

Figure 93. Additional Diagnostic Information (ISPPRS2)

Entry point, PSW, and register values are in hexadecimal. Abend code and reason code are in hexadecimal for system abends and in decimal for user abends. Meanings for the entries on the Additional Diagnostic Information panel are:

Abend code
Abend completion code, identified on the panel as “user” or “system”.

Reason code
Component reason code or return code associated with the abend.

ISPF Release Level
ISPF version/release/modification level.

Module Name
Name of abending program or *NOT SPECIFIED* if no name is available.

Entry Point Address
Entry point address of abending program.

PSW Program-Status Word at time of error.

Register content
General Purpose register content at time of error.

If the Recovery Termination Manager (RTM) could not get storage for the System Diagnostic Work Area (SDWA) or an error occurred within the error routine, all fields on this panel will contain 0’s, with the exception of the abend code and ISPF release level. Those fields will contain the correct data.

You can enter the HELP command from this panel as well to display the list of common abend codes. Information associated with an abend is available from the ISPF log file.

Pressing the END function key returns you to the primary option menu.

ISPF Statistics Entry in a PDS Directory

The following is the format of the information that ISPF writes to the PDS directory to maintain statistics for a member. If you suspect the statistics data has been corrupted, you can compare the existing entry against these formats to help in problem determination.

Byte **Description and Format**

1 Version number, in hexadecimal format. Value is between X'01' and X'99'.

2 Modification level, in hexadecimal format. Value is between X'00' and X'99'.

3 Flags:

Bit 1 SLCM indicator. SCLM uses this to determine whether the member and any related SCLM information are still in sync.

- ON means the member was last edited by SCLM, the PDF Software Configuration and Library Manager.
- OFF means the member was somehow processed outside SCLM.

Bit 2–8
Reserved for future ISPF use.

4 The seconds portion of the time last modified, in packed decimal format.

5–8 Creation date:

Byte 5 Century indicator.

Byte 6–8
Julian date, in packed decimal format

9–12 Date last modified:

Byte 9 Century indicator.

Byte 10–12
Julian date, in packed decimal format

13–14 Time last modified, in packed format:

Byte 13
Hours, in packed decimal format

Byte 14
Minutes, in packed decimal format

15–16 Current number of lines, in hexadecimal format

17–18 Initial number of lines, in hexadecimal format

19–20 Number of modified lines, in hexadecimal format

21–27 Userid, in character format

28–30 Reserved for future ISPF use

Common Problems Using ISPF

This section contains some common error messages that may be encountered while using ISPF. Error resolutions and explanations are also included.

Messages

- IKJ56500I COMMAND NOT FOUND

  If a command processor exists only in LPA, there must be an entry in the ISPTCM for the command processor. Please refer to *ISPF Planning and Customizing* for more details on customizing the ISPF TSO command table.

- IKJ56861I FILE ddname NOT FREED, DATA SET IS OPEN

  If the LIBRARY parameter is used with a table service, the user is not able to free the ddname for the table library pointed to by the LIBRARY parameter. ISPF keeps this library open until a new ddname is used in the LIBRARY parameter with another table service. ISPF functions in this manner for performance reasons.

  Issuing a table service with a LIBRARY parameter containing a ddname that does not exist causes the previous library to be closed and therefore allows the user to free the previous ddname. Use of CONTROL ERRORS RETURN may be used to guard against a severe error as a result of a ddname not existing.

  For example:

```
ALLOC FILE(DD1) DATASET('USERID.YOUR.TABLES') SHR
ISPEXEC TBOPEN MYLIB LIBRARY(DD1)
.
.                                  /*ISPF services against your table*/
.
ISPEXEC TBCLOSE MYLIB LIBRARY(DD1)
ISPEXEC CONTROL ERRORS RETURN
ISPEXEC TBOPEN JUNK LIBRARY(DDJUNK) /*non-existent table in a  */
                                    /*non-existent library     */
ISPEXEC CONTROL ERRORS CANCEL
FREE F(DD1)
```

- ISPP150

Panel 'name' error–At least one of the CLEAR names listed is not a panel field name. or ISPP121

Panel 'name' error–Panel definition too large, greater that screen size.

when entering KEYLIST, when requesting field-level help in ISPF panels, or when displaying panels created using DTL.

These messages are often caused by having a GML library in the ISPPLIB concatenation or by having GML source code in the panel library. Check your ISPPLIB concatenation to make sure that the ISPF-supplied GML library is not concatenated first. The ISPF-supplied GML library should not be in any of the ISPF library concatenations. Make sure that the libraries in your ISPPLIB concatenation do not contain GML source code.

- ISPT036 'Table in use–'table service' issued for table 'table name' that is in use, ENQUEUE failed.

  This message frequently occurs when batch jobs that use ISPF services run concurrently. This occurs because most batch jobs allocate a new profile each time they run. ISPF issues a TBOPEN against ISPPROF DD card for member ISPSPROF. The TBOPEN fails since ISPPROF does not contain this member. ISPF then issues a TBOPEN against ISPTLIB to copy the default ISPSPROF from ISPTLIB to ISPPROF.

  If the first data set in the ISPTLIB concatenation sequence is the same for two batch jobs running concurrently, message ISPT036 is issued. To make sure that this condition does not occur, the first data set in the ISPTLIB concatenation should be user unique. For example, *'sysuid..ISPPROF'* would be a user unique data set, which could be used as the first data set concatenated to the ISPTLIB DD.

  For the same reasons, this problem can also occur when two users logon to ISPF for the first time if they have the same data set concatenated first in the ISPTLIB concatenation.

- ISPT016, ISPT017, and other I/O Errors

  ISPF has various messages that reference I/O errors on either GET or PUT (READ and WRITE macros) such as message ISPT017. These errors are typically caused by concatenation problems on one of the ISPF libraries.

  Allocating data sets that do not have consistent DCB parameters in ISPF library concatenations often causes these messages. Also, ISPTABL, ISPFILE, and ISPPROF are used for output and therefore must have only a single data set allocated to their ddnames.

  – For I/O errors during panel services, check your ISPPLIB concatenation for inconsistent DCBs.
  – For I/O errors during file tailoring services, check your ISPSLIB concatenation for inconsistent DCBs and make sure that only one data set is allocated to ddname ISPFILE.
  – For I/O errors during table services, check your ISPTLIB concatenation for inconsistent DCBs and make sure that only one data set is allocated to ddname ISPTABL.

  I/O error messages cannot be issued when there is a problem with the ISPMLIB concatenation since messages cannot be located due to the I/O error. Message CMG999 occurs when there is an I/O error due to an ISPMLIB concatenation problem.

- CMG999

Diagnostic Information

CMG999 is issued with an appropriate description of the error condition for any problem with accessing a message. Refer to *ISPF Dialog Developer's Guide and Reference* for further information on how to define a message.

Unexpected Output

- ISPF services do not pick up updated copies of messages or panels.

  When not in TEST mode, the most recently accessed panel and message definitions are retained in virtual storage for performance reasons. If you have modified a panel or message file, using TEST mode ensures that the latest copy of each message or panel is accessed. Refer to *ISPF Services Guide* for more information on executing ISPF in TEST mode.
- ISPF commands such as WINDOW, COLOR, CUAATTR, EXIT, CANCEL, ACTIONS, KEYSHELP, KEYLIST, EXHELP, FKA, and ISPDTLC are not recognized as valid commands, or function keys defined as these commands do not function properly.

  The user issuing these commands or pressing the function keys defined as these commands has a private copy of ISPCMDS in the ISPTLIB concatenation. The user's private copy of ISPCMDS is missing some or all of the new commands supplied in the new command table, ISPCMDS.

  Users experiencing this problem should either replace their private copy of ISPCMDS with the OS/390 V2R8.0 ISPF supplied copy, or update their private ISPCMDS with the missing commands.

Abend Codes and Information

ISPF controller and processor task abends are controlled by STAE and STAI exit routines and by ISPF execution modes set using the ISPSTART TEST parameters.

Under normal conditions (that is, when processor and controller dumps have not been requested by specifying the ISPSTART TEST command):

- When a processor task abends:
  - No dump is taken.
  - The controller reattaches the processor main drive (ISPPMD).
  - The primary option menu is redisplayed for that logical screen.
- When the controller task abends:
  - ISPF terminates with `*** ISPF MAIN TASK ABEND ***` message.
  - Control returns to TSO.
  - Pressing Enter causes a dump to be taken if a dump data set has been allocated.

The controller and processor tasks issue the ABEND system service and allow dumps under certain situations. The ISPF modules that issue ABENDs and their associated codes and reasons are listed below:

ABEND0C1 in various common ISPF subroutines
: In several ISPF modules, an invalid operation code of (X'00') is executed to force an abend at the point that an unexpected condition occurs. Contact IBM support if this condition occurs within an ISPF module.

ABEND0C4 in ISPDVCGT, ISPDVCPT, or ISPDVCFD
: These abends are often caused by mismatched VDEFINE and VDELETE services in a user's program. The VDEFINE service gives ISPF addressability to user storage. This storage is used by variable services any time the variable that has been established by the VDEFINE service is

referenced. If this storage is released back to the system, an ABEND0C4 may occur depending on whether the storage is still accessible. Following are two common scenarios that often show these abends:

- A program establishes a variable in a called subroutine using the VDEFINE service and subsequently uses an ISPF service that references this variable in another routine. If the called subroutine was dynamically loaded and therefore released its storage, an ABEND0C4 could occur when the subroutine references a VDEFINEd variable.
- A program establishes a variable in a called subroutine using the VDEFINE service and then calls another program without using the SELECT service. Then the called program VDEFINEs a variable with the same name, but does not VDELETE it on exit. If the calling program references that variable after the called program returns control to it, an ABEND0C4 can occur. Since a VDELETE has not been done, ISPF services still reference the variable VDEFINEd by the called program.

If the program intent is to use the same variable in the main and called routines, the variable should be VDEFINEd only in the main routine. If the program intent to isolate a variable to be used only in the routine in which it is VDEFINEd, then the program should also VDELETE the variable before it ends. To diagnose whether the user application has this problem, a function trace on VDEFINE, VDELETE, and the SELECT services (Option 7.7.1) is very helpful.

Abend codes 111 or 222
To produce these abends, the user must be in test mode and request processor dumps by entering one of the following commands on the ISPF command line. With exception of the user completion code, both commands function in the same manner.

ABEND
Terminates ISPF with user completion code 111.

CRASH
Terminates ISPF with user completion code 222.

Abend code 908
ZISPFRC value was not valid

Abend code 920
ISPSTART command syntax was not valid

Abend code 985
An attempt was made to start a GUI in batch mode, but no workstation connection was made.

Abend code 987
An attempt was made to start GUI with GUISCRW or GUISCRD and the GUI intitialization failed.

Abend code 988
Invalid TSO environment. Refer to *ISPF Planning and Customizing* for the proper TSO version.

Abend code 989
The ISPF C/S component window was closed while still running ISPF in GUI mode

Diagnostic Information

Abend code 990
An error occurred running in batch mode. If ZISPFRC has not been set peviously, and ISPF encounters a severe error that terminates the product, then 990 is set.

Abend code 997 (or X'3E5')
A TPUT returned a return code other than 0 or 8. A message is displayed and an attempt is made to redisplay the full screen. If the redisplay fails twice, this abend is issued.

Abend code 998 (or X'3E6')
An ISPF severe error that occurs while not in CONTROL ERRORS RETURN mode and before ISPF is fully initialized. ISPF is considered to be fully initialized when the Enter key on the primary option menu has been processed without a severe error occurring.

Abend code 999 (or X'3E7')
This abend is issued for the following reasons:

- No function pool is established for a command processor.

  For example, a command processor that uses ISPF services is invoked using option 6 or SELECT CMD, but the command processor does not have a function pool. The user needs to have an entry for the command processor in the ISPTCM with the X'40' flag set on. The X'40' flag indicates that the command requires a function pool. Refer to *ISPF Planning and Customizing* for more information on customizing the ISPTCM.
- An error occurs while another error is already being processed.

  ISPF issues the abend code 999 in this case to protect against an infinite loop.
- An error occurred during ISPF initialization.

  For example:
  - An I/O error occurred due to ISPF library allocations such as ISPSLIB, ISPPLIB, ISPMLIB, and so forth, containing inconsistent or incorrect DCB attributes.
  - An ISPF library allocation does not contain the required ISPF libraries in its concatenation. For example, the ISPMLIB contains user product libraries but not ISPF libraries.

Terminal I/O Error Codes

Below is a list of terminal I/O error codes that you may see while using ISPF.

- ISPF screen output error code

  41 TPUT return code not equal to 0 or 8
- ISPF screen input error code

  21 TGET return code other than 0, 4, or 8.

  22 Input stream size greater than input buffer size or 0.

  23 Unknown attention identifier (AID).

  24 Invalid input AID.

  25 Input stream size invalid for input AID.

  26 Input cursor location not within physical screen.

28 First byte of input buffer field not an SBA (invalid input data).

31 Byte preceding the physical screen field is past the end of the physical screen (input data from invalid screen position).

32 Byte preceding the physical screen field is not an input attribute (input data from invalid screen position).

33 Physical screen field not defined on panel (input data from invalid screen position).

51 Physical screen field attribute not found in logical screen.

52 Byte preceding logical screen field is not an input attribute.

55 Physical screen size is greater than corresponding logical screen size.

Notes:

1. The physical screen size is determined by ISPF during initialization.
2. The input buffer size is a variable based on the physical screen size.
3. The logical screen is the same size as the physical screen, and is the size that the processor task uses for screen I/O. When the 3290 is running in 62 X 160 partition mode, the SPLITV command makes the logical screen width equal to 80. When a 3278 mod 5 is running in standard mode, the logical screen size is 24 X 80.
4. Only part of the logical screen appears on the physical screen when ISPF is running in split-screen mode. When the 3290 is running in 62 X 160 partition mode, the entire logical screen may be visible, depending on the position of the horizontal split line.
5. An input buffer field extends from an SBA to either the next SBA or the end of the input buffer.
6. A physical screen field extends from the location indicated in the input buffer SBA to the location of the next attribute byte in the physical screen.

Register Linkage Conventions

ISPF uses standard linkage conventions:

- **SELECT PGM(program-name)**

  REGISTER
  CONTENTS

  1 Points to the address of the parameter data (from the PARM keyword) field (half-word length) followed by the data

  2 - 12 Not used

  13 72-byte save area

  14 Return address

  15 Entry address / Return code on exit

- **ISPF EXITS / Call to ISPLINK**

  REGISTER
  CONTENTS

  1 On entry, points to a parameter list; each address in the list in turn points to a parameter. On return to the caller of ISPLINK, the user's parameter list starts at the second parameter. ISPF has inserted a parameter in front of the user's parameters for ISPF use.

2 - 12 Not used

13 72-byte save area

14 Return address

15 Entry address / Return code on exit

- **SELECT CMD(cmdname)** where *cmdname* is a program that will be attached as a command processor by ISPF:

  REGISTER
  CONTENTS

  1 Points to a CPPL (Command Processor Parameter List) which is a list of four addresses that point respectively to: Command buffer, UPT, PSCB, ECT. See the TSO programming services manual for descriptions of these parameters.

  2 - 12 Not used

  13 72-byte save area

  14 Not applicable

  15 Return code on exit

Usually when an abend occurs within ISPF code, register 12 points to the entry point of the abending CSECT.

Obtaining Message IDs

In order to obtain the message ID associated with an error message in ISPF, you need to be in ISPF TEST mode.

ISPF is in TEST mode if:

- ISPF is invoked with the TEST, TESTX, TRACE, or TRACEX parameter specified on the ISPSTART, PDF, or ISPF command, or
- Restore TEST/TRACE option is not selected in option 0 and you go into option 7, Dialog Test, at some point in your current ISPF session.

If you are not in TEST mode, split the screen, enter option 7, Dialog Test, and swap back to the screen containing the error.

You can use the either of the following methods to get the message ID:

- Enter `print` on the panel displaying the error message. The message ID, along with the displayed message text and screen output, appears in the LIST data set. The LIST data set can be printed using the LIST command.
- With the short message displayed:
  1. Press the function key assigned to Help (default is F1) or type `help` on the command line. This displays the long message text for the error.
  2. Press the function key assigned to Help or type `help` on the command line once more to display the Tutorial panel associated with the error. The bottom lines of the Tutorial panel contain fields that list the current panel name, the previous panel name, and the message ID. The value following `LAST MSG=` is the message ID associated with the error.

Installation, Maintenance, and Migration Diagnostics

The information following is representative of common situations a user may encounter during installation, maintenance, and migration.

Common Installation and Maintenance Problems

Problem:
Unpredictable results such as ABEND0C4s, ABEND0C1s, and so forth, upon invocation of ISPF.

Cause: This problem can occur when internal LINKs and LOADs are used in conjunction with the ISPF ISPLLIB DCB.

Solution:
When using STEPLIB to test new maintenance, releases, or versions of products and an ISPLLIB is allocated, data sets allocated to STEPLIB should also be allocated to ISPLLIB.

Problem:
New levels of code residing in STEPLIB do not appear to be executed. This includes changes to customer applications, new levels of code, and changes made by PTFs, for example.

Cause: This problem can occur when internal LINKS and LOADs are used in conjunction with the ISPF ISPLLIB DCB.

Solution:
When using STEPLIB to test new maintenance, releases, or versions of products and an ISPLLIB is allocated, data sets allocated to STEPLIB should also be allocated to ISPLLIB.

Problem:
Abends occur when invoking or exiting HELP, KEYS, or ISPPREP.

Cause: An older level of ISPF exists in LPA or LINKLIB; an ISPLLIB or STEPLIB is used to allocate libraries for the new release of ISPF; and a LIBDEF for ISPLLIB that does not include the new level of ISPF libraries has been issued.

Not all of ISPF is loaded at initialization, for example, ISPTUTOR is linked when HELP is requested. The LIBDEF of ISPLLIB results in the older level of any modules that are not loaded on ISPF entry to be picked up from LPA or LINKLIB due to internal search orders. For further information, refer to the LIBDEF command in *ISPF Services Guide*

Solution:
Include the new level of ISPF in any LIBDEFs issued for ISPLLIB.

Problem:
ABEND806 for ISPLINK or ABEND0C1 in user application.

Cause: During installation, the SMP/E install logic for ISPF deletes the existing ISPLINK load module.

Solution:
Relink-edit ISPLINK to user application.

Migration from Version 2 and Version 3 to Version 4.2

Problem:
ABEND806 received for program ISRYXDR.

Cause: The Dialog Test Facility is now part of ISPF and is invoked by PGM(ISPYXDR). The Dialog Test Facility was part of ISPF/PDF and was previously invoked using PGM(ISRYXDR) from the ISR@PRIM panel.

Solution:
Use the ISP@PRIM or ISR@PRIM panel supplied with this release of ISPF or modify your own customized ISP@PRIM or ISR@PRIM panels. Change the dialog test selection to invoke ISPYXDR rather than ISRYXDR as shown below:

```
'PGM(ISPYXDR) PARM(ISR) NOCHECK' /* ISR@PRIM */
              or
'PGM(ISPYXDR) PARM(ISP) NOCHECK' /* ISP@PRIM */
```

Problem:
Abends in load modules IGC0009C and IGC0009D.

Cause: Failure to install ISPF in the same zone as TSO/E will result in the supplied ISPF SVC 93 and SVC 94 exits (ISPSC93, ISPSC94), creating SVC 93 and SVC 94 load modules without the proper link edit information.

Failure to use the correct version of either IGC0009C or IGC0009D or both.

Solution:
TSO/E, ISPF, and ISPF/PDF must all be installed in the same zone. Verify that the correct versions are copied from a test system to the production system.

Problem:
ISPP150 PANEL 'name' ERROR NO ")END" FOUND BEFORE REACHING END OF FILE

or

SPP121 PANEL 'name' ERROR PANEL DEFINITION TOO LARGE, GREATER THAN SCREEN SIZE when entering KEYLIST, requesting field level help in ISPF panels, or when displaying panels created using DTL.

Cause: These messages are often caused by having a GML library in the ISPPLIB concatenation, or by having GML source code in the panel library.

Solution:
Check your ISPPLIB concatenation to make sure that the ISPF-supplied GML library is not concatenated first. The ISPF-supplied GML library should not be in any of the ISPF library concatenations. Make sure that the libraries in your ISPPLIB concatenation do not contain GML source code.

Problem:
ISPF Version 4 specific commands such as RESIZE, SETTINGS, TSOCMD, START, WS, TUTOR, ISPFVAR, ZKEYS, DTEST, STATUS, and so on, are not recognized as valid commands or function keys defined as these commands do not function properly.

Cause: Users issuing these commands or pressing the function keys defined as these commands have a private copy of ISPCMDS in their ISPTLIB concatenation. Users' private copy is missing some or all of these supplied commands, or they may be using a command table from a previous version or release of ISPF.

Solution:
Users experiencing this problem should put their non-ISPF commands that are currently in their private copy of the ISPCMDS table into a user or site command table, and delete their ISPCMDS table.

Problem:
Receiving error messages such as: 'PROFILE TABLE NOT FOUND, UNABLE TO ACCESS ISRPROF TABLE FOR VAR 'ZERRMSG'' or ABEND3E7 or ABENDU999.

Cause: The profile data set may run out of directory space as Version 3 adds more members to the profile, or the logon allocation of ISPTLIB may be pointing to an old Version 2 or Version 3 table library. Another common cause of this symptom is that an application that link-edits the ISPF load module ISPTASK into it has not been relink-edited after installing a new version of ISPTASK.

Solution:
Add more directory blocks to the profile data set and ensure that the Version 4.2 table library containing ISPKEYS is in the ISPTLIB concatenation. Verify that the application that link-edits ISPTASK is relink-edited after a new release of ISPF is installed or after maintenance that updates ISPTASK is installed.

Problem:
Attempting to start an ISPF session with a workstation connection using ISPF C/S causes error messages like: 'CSV003I REQUESTED MODULE LSCSIO NOT FOUND' +LSCX012 Unable to load runtime I/O routines, execution cannot continue.

Cause: Library SISPSASC is not in the usual MVS load module search order. This library is *not* searched using the ISPLLIB allocation. It must be in STEPLIB or LNKLST.

Solution:
Put the SISPSASC library in STEPLIB or LNKLST. For more information about SISPSASC, refer to *ISPF Planning and Customizing*

Appendix D. Dialog Variables

This appendix describes the ISPF dialog variables.

The following table lists the dialog function pool variables that are both read from and written to by several of the PDF library access services. The variables are listed in alphabetical order.

The first column lists the variable name. The second column indicates the variable's type, which corresponds to the format parameter of the ISPF VDEFINE service. The third column specifies the variable's length, which corresponds to the length parameter of the VDEFINE service.

The fourth column lists the PDF services that either read from or write to the variable. An *R* in parentheses after a service name indicates that the service, when called, reads from the given variable. A *W* in parentheses after a service name indicates that the service, when called, writes to the given variable. All variables are available to a dialog unless otherwise indicated.

The last column contains a brief description of the contents of the variable and any restrictions on the value of the variable.

Variable Name	Format	Length	Service (Access)	Description
ZCMD	Char	256	LMMDISP(W)	Primary Command field from member list panel if the command is not a valid ISPF or PDF primary command.
ZDLBLKSZ	Char	5	LMDLIST(W)	Block size.
ZDLCDATE	Char	10	LMDLIST(W)	Creation date.
ZDLDEV	Char	8	LMDLIST(W)	Device type.
ZDLDSNTP	Char	8	LMDLIST(W)	DS name type ('PDS', 'LIBRARY', or ' ').
ZDLDSORG	Char	4	LMDLIST(W)	Data set organization.
ZDLEDATE	Char	10	LMDLIST(W)	Expiration date.
ZDLEXT	Char	3	LMDLIST(W)	Number of extents used.
ZDLLRECL	Char	5	LMDLIST(W)	Logical record length.
ZDLMIGR	Char	3	LMDLIST(W)	Whether the data set is migrated ('YES' or 'NO').
ZDLRDATE	Char	10	LMDLIST(W)	Date last referenced.
ZDLRECFM	Char	5	LMDLIST(W)	Record format.
ZDLSIZE	Char	6	LMDLIST(W)	Data set size in tracks.
ZDLSPACU	Char	10	LMDLIST(W)	Space units, one of the following: CYLINDERS, MEGABYTES, KILOBYTES, BYTES, BLOCKS or TRACKS.
ZDLUSED	Char	3	LMDLIST(W)	Percentage of used tracks or pages (PDSE).
ZDLVOL	Char	6	LMDLIST(W)	Volume serial.

© Copyright IBM Corp. 1980, 1999

Dialog Variables

Variable Name	Format	Length	Service (Access)	Description
ZDSN	Char	44	LMMDISP(W)	Name of the first or only data set in the concatenation of the member list being displayed. This variable is only available for member list panels.
ZDST	Char	54	BRIF (W) EDIF (W)	Title line data name for EDIF and BRIF.
ZEDBDSN	Char	44	EDIT (R) EDREC(W)	Backup data set name for standard edit recovery.
ZEDITCMD	Char	8	Any EDIT macro	The last primary command entered in Edit.
ZEDROW	Fixed	4	EDIT (R) EDREC(W)	Row number of entry in standard edit recovery table.
ZEDSAVE	Char	8	Data_changed EDIT macro command	END command will save data (SAVE or NOSAVE).
ZEDTDSN	Char	44	EDIT (R) EDREC(W)	Target data set name for standard edit recovery.
ZEDTMCMD	Char	8	Any Edit macro	The edit command entered that caused an edit macro to run. Can be the macro name or other name is the edit DEFINE command was used to define an alias.
ZEDTMEM	Char	8	EDIT (R) EDREC(W)	Target member name (if applicable) for standard edit recovery.
ZEDTRD	Char	6	EDIT (R) EDREC(W)	Volume serial of target data set for standard edit recovery.
ZEDUSER	Char	[2]	EDIT (R) EDREC(W)	User data table extension for standard edit recovery.
ZEIBSDN	Char	54	EDIF (R) EDIREC(W)	Backup data name for EDIF edit recovery.
ZEIROW	Fixed	4	EDIF (R) EDIREC(W)	Row number of entry in EDIF edit recovery table.
ZEITDSN	Char	54	EDIF (R) EDIREC(W)	Target data name for EDIF edit recovery.
ZEIUSER	Char	[2]	EDIF (R) EDIREC(W)	User data table extension variable for EDIF edit recovery.
ZERRALRM	Char	3	ALL(W)	The value YES if an alarm was specified in the message definition; otherwise, the value NO. Set when ISPF services issue a return code of 8 or greater.
ZERRHM	Char	8	ALL(W)	The name of a Help panel, if one was specified in the message definition. Set when ISPF services issue a return code of 8 or greater.

Variable Name	Format	Length	Service (Access)	Description
ZERRLM	Char	512	ALL(W)	Long-message text in which variables have been resolved. Set when ISPF services issue a return code of 8 or greater.
ZERRMSG	Char	8	ALL(W)	Message ID. Set when ISPF services issue a return code of 8 or greater.
ZERRSM	Char	24	ALL(W)	Short-message text in which variables have been resolved. Set when ISPF services issue a return code of 8 or greater.
ZGRPLVL	Char	8	LMHIER (W)	ISPF table variable that contains the level of this ISPF library in the controlled hierarchy.
ZGRPNME	Char	8	LMHIER (W)	ISPF table variable that contains the ISPF library group name.
ZLAC	Char	2	LMMDISP(W) LMMFIND(W) LMMLIST(W)	Authorization code of the member.
ZLALIAS	Char	8	LMMDISP(W) LMMFIND(W) LMMLIST(W)	Name of the real member of which this member is an alias.
ZLAMODE	Char	3	LMMDISP(W) LMMFIND(W) LMMLIST(W)	AMODE of the member.
ZLATTR	Char	20	LMMDISP(W) LMMFIND(W) LMMLIST(W)	Load module attributes. Refer to the *ISPF Services Guide*.
ZLCDATE	Char	8	LMMADD(R) LMMDISP(W) LMMFIND(W) LMMLIST(W) LMMREP(R)	Date on which the specified member was created. A character string in the national format. For example, yy/mm/dd or mm/dd/yy. If no value exists for this variable, the PDF component will set the value to blanks.
ZLC4DATE	Char	10	LMMADD(R) LMMDISP(W) LMMFIND(W) LMMLIST(W) LMMREP(W)	Date on which the specified member was created, in 4-character year format. A character string in the national format. For example, yyyy/mm/dd or mm/dd/yyyy. If no value exists for this variable, the PDF component will set the value to blanks.
ZLCNORC	Fixed	4	LMMADD(R) LMMDISP(W) LMMFIND(W) LMMLIST(W) LMMREP(R)	Current number of records in the specified member. A number from 0 to 65 535. If no value exists for this variable, the PDF component will set the value to blanks.

Dialog Variables

Variable Name	Format	Length	Service (Access)	Description
ZLINORC	Fixed	4	LMMADD(R) LMMDISP(W) LMMFIND(W) LMMLIST(W) LMMREP(R)	Number of records in the specified member when it was first created. A number from 0 to 65 535.
ZLLIB	Fixed	4	LMMDISP(W) LMMFIND(W) LMMLIST(W)	Position of the specified member in the concatenated data sets. A number from 1 to 4.
ZLMDATE	Char	8	LMMADD(R) LMMDISP(W) LMMFIND(W) LMMLIST(W) LMMREP(R)	Date on which the specified member was last modified. A character string in the national format. (For example, *yy/mm/dd* or *mm/dd/yy*.) If no value exists for this variable, the PDF component will set the value to blanks.
ZLM4DATE	Char	10	LMMADD(R) LMMDISP(W) LMMFIND(W) LMMLIST(W) LMMREP(W)	Date on which the specified member was last modified, in 4-character year format. A character string in the national format. (For example, *yyyy/mm/dd* or *mm/dd/yyyy*.) If no value exists for this variable, the PDF component will set the value to blanks.
ZLMEMBER	Char	8	LMMDISP(W)	Name of the current selected member.
ZLMNORC	Fixed	4	LMMADD(R) LMMDISP(W) LMMFIND(W) LMMLIST(W) LMMREP(R)	The number of records that have been modified in the specified member. A number from 0 to 65 535.
ZLMOD	Fixed	4	LMMADD(R) LMMDISP(W) LMMFIND(W) LMMLIST(W) LMMREP(R)	Modification level of the specified member. A number from 0 to 99.
ZLMTIME	Char	5	LMMADD(R) LMMDISP(W) LMMFIND(W) LMMLIST(W) LMMREP(R)	Time when the specified member was last modified. A character string in the form hh:mm.
ZLMSEC	Char	2	LMMADD(R) LMMDISP(W) LMMFIND(W) LMMLIST(W) LMMREP(R)	Seconds value of last modified time.
ZLPDSUDA	Char	62	LMMDISP(W)	A character string containing the contents of the user data area in the PDS directory entry of the specified member if the member's statistics are not in PDF format.

Variable Name	Format	Length	Service (Access)	Description
ZLRMODE	Char	3	LMMDISP(W) LMMFIND(W) LMMLIST(W)	RMODE of the member.
ZLSIZE	Char	8	LMMDISP(W) LMMFIND(W) LMMLIST(W)	Load module size (in Hex).
ZLTTR	Char	6	LMMDISP(W) LMMFIND(W) LMMLIST(W)	TTR of the member.
ZLUSER	Char	7	LMMADD(R) LMMDISP(W) LMMFIND(W) LMMLIST(W) LMMREP(R)	User ID of user who last modified the specified member.
ZLVERS	Fixed	4	LMMADD(R) LMMDISP(W) LMMFIND(W) LMMLIST(W) LMMREP(R)	Version number of the specified member. A number from 1 to 99. If no value exists for this variable, the PDF component will set the value to blanks.
ZMEMCNT	Char	8	LMMLIST(W)	Number of members in the member list.
ZMLCOLS	Char	80	LMMDISP(W)	A character string that contains the member statistics column headings that appear on the member list panel display. This variable is only available for member list panels.
ZMLCR	Fixed	4	LMMDISP(W)	The relative number in the member list of the member that appears at the top of the member list display. Its range is from 1–99 999. This variable is only available for member list panels.
ZMLTR	Fixed	4	LMMDISP(W)	Number of members in the member list. Its range is from 1–99 999. This variable is only available for member list panels.
ZSCALIAS	Char	1	LMINIT(W)	Data set name is an alias (’Y’ or ’N’).
ZSCLM	Char	1	LMMDISP(W) LMMFIND(W) LMMLIST(W)	Last updater of member. ’Y’ indicates SCLM was last updater. ’N’ indicates PDF.
ZSCMVOL	Char	1	LMINIT(W)	Data set name is multivolume (’Y’ or ’N’).
ZUSERMAC	Char	0	EDIT(R) EDIF(R)	Application-wide edit macro.

PDF Non-Modifiable Variables

The following read-only variables are available to PDF component dialogs:

Variable Name	Format	Length	Service (Access)	Description
ZCUNIT	Char	8	none	Unit name to be used for temporary allocations. This variable comes from ISPF configuration table keyword PDF_DEFAULT_UNIT.
ZCUSIZE	Fixed	4	none	Number of kilobytes available for use by the edit UNDO command when running in SETUNDO STORAGE mode. This variable comes from ISPF configuration table Keyword UNDO_STORAGE_SIZE. See *ISPF Edit and Edit Macros* for further information.
ZICFPRT	Char	3	none	ICF indicator. 'YES' - All foreground print requests will be processed using ICF. 'NO' - ICF will not be used. This variable comes from ISPF configuration table keyword PRINT_USING_ICF.
ZPDFREL	Char	8	none	PDF version number in the form "PDF x.y ". The x.y is a sequence number. If x.y: • <= 4.2 means the x.y version.release of PDF • = 4.3 means ISPF for OS/390 Release 2 • = 4.4 means PDF 4.2.1 and ISPF OS/390 Release 3
ZSESS	Char	8	none	Session manager indicator. 'Y' - Use session manager panels in options 4 and 6. 'N' - Use standard panels in options 4 and 6. This variable comes from ISPF configuration table keyword USE_SESSION_MANAGER.
ZSWIND	Char	4	none	Sliding window value used by PDF for determining the century of 2–character years. This variable comes from ISPF configuration table keyword YEAR_2000_SLIDING_RULE. Dates less than or equal to this value are 20xx. Dates greater than this value are 19xx.

2. Length limited only by ISPF restrictions on the length of table extension variables.

Appendix E. System Variables

The system variables are described with type and pool information in the following tables. The variables are also discussed with the ISPF service to which they apply.

Commonly used system variables that a dialog can access are listed below. They are grouped by topic.

The first column gives the name of the variable. The second column indicates in which pool the variable resides. The following abbreviations are used:
func Function pool
shr Shared pool
prof Profile pool
any Any pool.

The third column indicates the variable's type. The following abbreviations are used:
in Input variable, set by a dialog to provide information to ISPF
out Output variable, set by ISPF to provide information to dialogs
non Non-modifiable output variable
i/o Both an input and an output variable.

The fourth column gives the length of the variable.

The fifth column gives a brief description of the variable.

Numeric system variables set by ISPF are right-justified and padded with zeros on the left, if necessary. If a program function uses the VCOPY service to access the variable, the value will be in character string format rather than in fixed binary format.

Time and Date

Name	Pool	Type	Len	Description
ZDATE	shr	non	8	Current date. The format of ZDATE depends on the current national language (see ZDATEF and ZDATEFD).
ZDATEF	shr	non	8	Current national language date format using the characters DD for day, MM for month, and YY for year. ZDATEF contains the national language delimiter. For example, DD/MM/YY, YY/MM/DD, MM.DD.YY. For countries that use a delimiter other than a slash (/), that delimiter replaces the slash in the date representation.
ZDATEFD	shr	non	8	The date format as described under ZDATEF but with the national language convention instead of DD, MM, and YY.
ZDATESTD	shr	non	8	Current date with a 4–digit year (YYYY/MM/DD). The format of ZDATESTD depends on the current national language (see ZDATEF and ZDATEFD).
ZDAY	shr	non	2	Day of month (2 characters)
ZJDATE	shr	non	6	Day-of-year date (format yy.ddd)
ZJ4DATE	shr	non	8	Day-of-year date (format yyyy.ddd)

© Copyright IBM Corp. 1980, 1999

System Variables

Name	Pool	Type	Len	Description
ZMONTH	shr	non	2	Month of year (2 characters)
ZSTDYEAR	shr	non	4	All 4 digits of the current year (4 characters).
ZTIME	shr	non	5	Time of day (format hh:mm)
ZTIMEL	shr	non		Time of day (format hh:mm:ss:TQ —where *T* is tenths of a second, and *Q* is hundredths)
ZYEAR	shr	non	2	Year (2 characters)

The current date is displayed in the appropriate format for the session language, where DD=DAY, MM=MONTH, and YY=YEAR. For countries that use a delimiter other than a slash (/), that delimiter replaces the slash in the date representation.

General

Name	Pool	Type	Len	Description
Z	shr	non	0	Null Variable
ZACCTNUM	shr	non	40 [3]	The MVS account number specified at logon time.
ZAPLCNT	shr	non	4	Number of times APL invoked for a logical screen
ZAPPLID	shr	non	8	Application identifier
ZAPPTTL	any	in	N/A	When running in GUI mode, the title to be displayed in the window frame. **Note:** If the panel is to be displayed in a pop-up window, the value specified in ZWINTTL will be used instead of ZAPPTTL.
ZCS	shr	non	5	NLS currency symbol
ZCSDLL	shr	non	8	Filename of the DLL required for this level of code for the Client/Server
ZDECS	shr	non	1	NLS decimal separator character
ZDEL	shr	i/o	1	The delimiter is used to separate stacked commands. The default delimiter is a semicolon (;).
ZENTKTXT	any	in	12	When you are running in GUI mode, the name that appears on the Enter key push button. If this variable is not found, *Enter* appears on the push button.

3. 40 is the maximum length.

Name	Pool	Type	Len	Description
ZENVIR	shr	non	32	Environment description: • 1 to 8 contain the product name and sequence number, ISPF x.y. The sequence number x.y indicates the following. If the sequence number: – <= 4.2 means the x.y version.release of ISPF – = 4.3 means ISPF for OS/390 Release 2 – = 4.4 means ISPF 4.2.1 and ISPF OS/390 Release 3 System variable ZISPFOS will contain the release of ISPF for OS/390 on your system. System variable ZOS390RL will contain the release of OS/390 on your system. • 9 to 16 contain the generic operating system name (MVS) • 17 to 24 contain the operating system environment (TSO or BATCH) • 25 to 32 contain blanks and are reserved.
ZEURO	shr	non	1	The EURO currency symbol.
ZGUI	shr	non	68	Workstation address or name (in character format) if ISPSTART is issued with the GUI parameter or if specified on the Settings GUI invocation panel. ZGUI will be set to blank if ISPSTART is issued without the GUI parameter or if GUI is not invoked from the Settings panel.
ZISPFOS	shr	non	30	The level of ISPF code that is running as part of OS/390 on your system. This level might or might not match the OS/390 level forund in ZOS390RL.
ZISPFRC	shr	in	8	Return code from ISPSTART-selected dialog to invoking application.
ZKEYHELP	any	in	8	Keys help panel identifier. If a keys help panel is not specified on the referenced keylist, the application can provide the keys help panel name in this variable. If the help panel name is present as part of the referenced keylist definition, it takes precedence over the ZKEYHELP value. This system variable must be redefined each time the keys help panel is to change.
ZLANG	prof	non	8	Session language
ZLOGO	shr	non	3	Indicates whether the user has requested bypass of LOGO panel. NO indicates that the user has specified the NOLOGO keyword at the time ISPF was called, thus, requesting that the LOGO panel be bypassed. Otherwise, the value of the variable will be YES.
ZLOGON	shr	non	8	Stepname of TSO logon procedure
ZOS390RL	shr	non	16	Indicates the OS/390 release running on your system.
ZPFKEY	shr	non	4	The name of the PFkey (PFxx) in effect when the user exits the panel.
ZPLACE	prof	i/o	7	Command line placement (ASIS or BOTTOM)
ZPREFIX	shr	non	8	TSO user prefix
ZPROFAPP	prof	in	8	Name of application profile pool extension table
ZSCRCUR	shr	non	4	Displays the number of logical screens currently in use.
ZSCRNAME	shr	in	8	Screen name set by dialog. The screen name is in effect only for the select level in which it was defined. Option 7.3 can alter ZSCRNAME, but this will have no impact. See “ZSCRNAME Examples” on page 365 for examples of its use.

System Variables

Name	Pool	Type	Len	Description
ZSCRMAX	shr	non	4	Displays the number of logical screens allowed by the installation.
ZSCTPREF	shr	non	4	Site command table prefix
ZSCTSRCH	shr	non	1	Search order for site command table relative to system command table. Set to either B (Before ISP) or A (After ISP).
ZSYSICON	shr	non	8	The 8-character variable that contains the command to be executed when the system icon is double-clicked or **close** is selected.
ZSYSID	shr	non	8	The 8-character SYSNAME obtained from the SYS1.PARMLIB member IEASYSxx which is read at IPL time. NONAME is the default value of SYSNAME. The operator can change this value at IPL time. See the *MVS/ESA System Programming Library: Initialization and Tuning* (GS28-1828-2 for more information.
ZSYSNODE	shr	non	12	The network node name of your installation's JES. This name identifies the local JES in a network of systems or system complexes being used for network job entry (NJE) tasks. The node name returned in ZSYSNODE derives from the NODE initialization statement of JES. If the system finds that the subsystem is not active, the ZSYSNODE variable contains the string ***—INACTIVE—*** (note the string delimiters). If the system finds that the subsystem is neither JES2 4.3 or later, nor JES3 5.1.1 or later, the ZSYSNODE variable contains the string ***' —DOWNLEVEL—'*** (note the string delimiters). The value in ZSYSNODE remains the same throughout the ISPF session. **Note:** If, for instance, the JES subsystem is taken down during an ISPF session and the node name is changed, the value in ZSYSNODE will still contain the value as determined at ISPF initialization.
ZSYSPLEX	shr	non	8	The MVS sysplex name as found in the COUPLExx or LOADxx member of **SYS1.PARMLIB**. If no sysplex name is specified in SYS1.PARMLIB, ZSYSPLEX contains blanks.
ZTEMPF	shr	non	44	Name of temporary data set for file tailoring output
ZTEMPN	shr	non	8	DDNAME of temporary data set for file tailoring output
ZTERMCID	shr	non	5	CCSID coded character set identifier of the terminal. Set by ISPF based on the code page and character set of the terminal. If the terminal code page and character set cannot be queried or if they are not supported by ISPF, this variable will be blank.
ZTERMCP	shr	non	4	CECP support 4-digit code page
ZTERMCS	shr	non	4	CECP support 4-digit character set
ZTHS	shr	non	1	NLS thousands separator character
ZTS	shr	non	1	NLS time separator character
ZTSICMD	shr	non	32767 [4]	The entire initial invocation command string which invoked the ISPF environment. If storage cannot be obtained at startup, only the first 50 characters will be saved.
ZTSSCMD	shr	non	32767 [4]	SELECT portion of the initial invocation command.
ZUCTPREF	shr	non	4	User command table name

Name	Pool	Type	Len	Description
ZUSER	shr	non	8	User ID
ZVERB	shr	out	8	Command verb after a SETVERB command table action
ZWINTTL	any	in	N/A	Title to be displayed in pop-up window frame
ZWSCDPG	shr	non	4	When running in GUI mode, the code page of the workstation. When not running in GUI mode, value will be blank.
ZWSCON	shr	non	68	TCP/IP or APPC address when ISPF session is connected to a workstation.
ZWSOPSYS	shr	non	16	Operating system of workstation to which the session is connected. The first 10 characters are the operating system name, followed by a blank, followed by two 2-digit numbers separated by a blank. These numbers are returned to ISPF from the operating system and change by version and release.

ZSCRNAME Examples

Example 1

On the ISPF primary option panel the user issues the command SCRNAME POP. The primary option panel's screen name is now POP. The user then invokes CLIST1.

```
CLIST1

   PROC 0
   ISPEXEC DISPLAY PANEL(PANELA)
   SET &ZSCRNAME = EDIT1
   ISPEXEC VPUT (ZSCRNAME) SHARED
   ISPEXEC EDIT DATASET ('PROJECT.GROUP.TYPE(BBBBBB)')
   SET &ZSCRNAME = EDIT2
   ISPEXEC VPUT (ZSCRNAME) SHARED
   ISPEXEC EDIT DATASET ('PROJECT.GROUP.TYPE(CCCCCC)')
   SET &ZSCRNAME = BROWSE1
   ISPEXEC VPUT (ZSCRNAME) SHARED
   ISPEXEC BROWSE DATASET ('PROJECT.GROUP.TYPE(DDDDDD)')
   SET &ZSCRNAME = LASTPAN
   ISPEXEC VPUT (ZSCRNAME) SHARED
   ISPEXEC DISPLAY PANEL(PANELA)
```

After the CLIST processes, the following results occur:

1. PANELA displays with screen name POP.
2. The EDIT session displays with the screen name EDIT1.
3. The next EDIT session displays with the screen name EDIT2.
4. The BROWSE session displays with the screen name BROWSE1.
5. PANELA displays with the screen name LASTPAN.
6. End from PANELA and the primary option panel displays with screen name POP.

4. 32767 is the maximum length.

Example 2

On the ISPF primary option panel the user issues the command SCRNAME POP. The primary option panel's screen name is now POP. The user then invokes CLIST1 with the following results:

1. PANELA displays with screen name POP.
2. The EDIT session displays with the screen name EDIT1.
3. The user enters SCRNAME MYEDIT, so the screen name becomes MYEDIT.
4. After the EDIT session ends, the CLIST sets ZSCRNAME to EDIT2.
5. The EDIT session displays with the screen name EDIT2.
6. After this EDIT session ends, the CLIST sets ZSCRNAME to BROWSE1.
7. The BROWSE session displays with the screen name BROWSE1.
8. The user enters SCRNAME MYBROWSE PERM, so the screen name becomes MYBROWSE.
9. After the BROWSE session ends, the CLIST sets ZSCRNAME to LASTPAN.
10. PANELA displays with the screen name MYBROWSE. The CLIST command ZSCRNAME=LASTPAN is ignored because the user issued the SCRNAME MYBROWSE command with the PERM parameter.
11. The CLIST completes and the primary option panel displays with the screen name MYBROWSE (again because the user issued the SCRNAME MYBROWSE command with the PERM parameter).

Example 3

On the ISPF primary option panel the user issues the command SCRNAME POP. The primary option panel's screen name is now POP. The user then invokes CLIST2.

```
CLIST2

   PROC 0
   SET &ZSCRNAME = STATE
   ISPEXEC VPUT (ZSCRNAME) SHARED
   ISPEXEC SELECT PANEL(MENUA) SCRNAME(NATION)
   ISPEXEC DISPLAY PANEL(PANELA)
```

After the CLIST processes, the following results occur:

1. MENUA displays with screen name NATION.
2. PANELA displays with the screen name STATE.
3. End from PANELA and the primary option panel displays with screen name POP.

Terminal and Function Keys

Name	Pool	Type	Len	Description
ZCOLORS	shr	non	4	Number of colors supported by the terminal type (either 1 or 7)
ZDBCS	shr	non	3	DBCS terminal capability (YES or NO)

5. 255 is the maximum length.

Name	Pool	Type	Len	Description
ZFKA	prof	non	8	Current state of the function key area form (LONG, SHORT, OFF (no display))
ZGE	shr	non	3	Terminal support for graphic escape order: • YES — graphic escape is supported • NO — graphic escape is not supported. **Note:** If you are running in GUI mode, ZGE will be set to Off.
ZHILITE	shr	non	3	Extended highlighting availability (YES or NO)
ZKEYS	prof	out	4	Number of Function keys
ZKLAPPL	shr	non	4	If KEYLIST is ON and it is a panel with the)PANEL statement, this contains the application id where the current keylist came from.
ZKLNAME	shr	non	8	If KEYLIST is ON and it is a panel with the)PANEL statement, this contains the name of the current keylist.
ZKLTYPE	shr	non	1	If KEYLIST is ON and it is a panel with the)PANEL statement, this contains either P (for Private) or S (for Shared) for the current keylist.
ZKLUSE	prof	i/o	1	If KEYLIST is ON this contains Y, if it is OFF, it contains an N.
ZPFCTL	prof	i/o	5	User authorization to use PFSHOW command • USER—User controls function key display with PFSHOW command • ON—Display function key defitions on all panels • OFF—Do not display function key definitions
ZPFFMT	prof	i/o	4	Number of Function key definitions displayed per line • SIX—Always display six keys per line • MAX—Display as many keys as will fit on each line
ZPFSET	prof	i/o	4	Function key definition set displayed • PRI—Primary set (1–12) • ALT—Alternate set (13–24) • ALL—All keys (1–24)
ZPFSHOW	prof	out	4	PFSHOW command status
ZPFxx	prof	i/o	255 [5]	Setting for Function keys: ZPF13-ZPF24 contain settings for the primary keys (for 12-key terminals: physical keys 1-12; for 24-key terminals: physical keys 13-24) ZPF01-ZPF12 contain settings for the alternate keys (for 24-key terminals only: physical keys 1-12)
ZPFLxx	prof	i/o	8	Setting for Function key labels: ZPF13-ZPF24 contain labels for the primary keys ZPF01-ZPF12 contain settings for the alternate keys
ZPRIKEYS	prof	i/o	4	Indicates the set of Function keys that will be the primary keys • LOW—1 to 12 are primary keys • UPP—13 to 24 are primary keys

System Variables

Name	Pool	Type	Len	Description
ZSCREEN	shr	non	1	Logical screen number up to 32 screens (1–9, A–W)
ZSCREEND	shr	non	4	Screen depth available for dialog use. In batch mode, this variable is set by the value specified for BATSCRD on the ISPSTART call.
ZSCREENW	shr	non	4	Screen width available for dialog use. In batch mode this variable is set by the value specified for BATSCRW on the ISPSTART call. ZSCREEND and ZSCREENW are generally the dimensions of the physical display screen. There are two exceptions: 1. On a 3290, if a dialog is executing on a display with a width of 160 characters and the user does a vertical split, then ZSCREENW is 80. 2. On a 3278 model 5, if a user has specified SCREEN FORMAT IS STD, then ZSCREENW is 80 and ZSCREEND is 24, rather than the maximum physical size of 132 by 27.
ZSCRMAXD	shr	non	4	Maximum screen depth available for dialog use. In batch mode, this variable is set by the value specified for BATSCRD on the ISPSTART call.
ZSCRMAXW	shr	non	4	Maximum screen width available for dialog use. In batch mode, this variable is set by the value specified for BATSCRW on the ISPSTART call. ZSCRMAXD and ZSCRMAXW are identical to ZSCREEND and ZSCREENW, except for terminals on which an alternate size is available. In that case, ZSCRMAXD and ZSCRMAXW contain the screen configuration size that produces the largest screen. For the 3290, these variables contain sizes of the hardware partition on which ISPF is operating.
ZSPLIT	shr	non	3	Split-screen mode in effect (YES or NO)
ZTERM	prof	out	8	Terminal type as defined by option 0

Scrolling

Name	Pool	Type	Len	Description
ZSCBR	prof	i/o	4	Scroll amount for the BROWSE service
ZSCED	prof	i/o	4	Scroll amount for the EDIT service
ZSCML	prof	i/o	4	Scroll amount for member lists
ZSCROLLA	shr	out	4	Value from scroll amount field (PAGE, MAX, number)
ZSCROLLD	any	in	4	Value to be used as default scroll value for scrollable dynamic areas and table display
ZSCROLLN	shr	out	4	Scroll number as computed from the value in the scroll amount field

PRINTG Command

Name	Pool	Type	Len	Description
ZASPECT	func	in	4	Aspect ratio of printed output from PRINTG
ZDEVNAM	func	in	8	Device name for PRINTG

Name	Pool	Type	Len	Description
ZFAMPRT	func	non	4	Family printer type for PRINTG

Table Display Service

Name	Pool	Type	Len	Description
ZTDADD	func	out	3	More rows needed to satisfy scroll request (YES|NO)
ZTDAMT	func	out	4	Number of rows that the dialog should add to satisfy scroll
ZTDLROWS	func	in	6	Number of rows in the logical table (dynamic table expansion)
ZTDLTOP	func	in	6	Maps current top row in physical table to its position in logical table.
ZTDMARK	any	in	note [6]	User-defined text for table display Bottom-of-Data marker
ZTDMSG	any	in	8	User-defined message ID for table display top-row-displayed indicator
ZTDRET	func	in	8	Defines whether dialog wants to use scroll return feature.
ZTDROWS	func	out	6	Number of table rows upon return from table display
ZTDSCRP	func	in/out	6	CRP of top row to be displayed after the scroll
ZTDSELS	func	out	4	Number of selected table rows upon return from each table display
ZTDSIZE	func	out	4	Size (number of model sets) of the table display scrollable section
ZTDSRID	func	out	6	Rowid of the row pointed to by ZTDSCRP
ZTDTOP	func	out	6	Row number (CRP) of top row displayed during most recent table display

LIST Service

Name	Pool	Type	Len	Description
ZLSTLPP	shr	non	4	Number of lines per page in list data set
ZLSTNUML	shr	non	4	Number of lines written to current list data set page
ZLSTTRUN	shr	non	4	List data set record length truncation value

LOG and LIST Data Sets

Name	Pool	Type	Len	Description
ZLOGNAME	shr	non	44	Contains the fully qualified data set name of the log data set.
ZLSTNAME	shr	non	44	Contains the fully qualified data set name of the list data set.

6. Any length not more than the screen width.

Dialog Error

Name	Pool	Type	Len	Description
ZERRALRM	func	out	3	Message alarm indicator (YES or NO)
ZERRHM	func	out	8	Name of help panel associated with error message
ZERRLM	func	out	512	Long error message text
ZERRMSG	func	out	8	Error message-id
ZERRSM	func	out	24	Short error message text
ZERRTYPE	func	out	8	Error message type
ZERRWIND	func	out	6	Error message window type

Tutorial Panels

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

Selection Panels

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

DTL Panels or Panels Containing a)PANEL Section

Name	Pool	Type	Len	Description
ZCURFLD	func	out	8	Name of field (or list column) containing the cursor when the user exits the panel.
ZCURINX	func	out	8	For table display panels, the current row number of the table row containing the cursor. The value ZCURINX is in character format. If the cursor is not within a table row, this value will be 0.
ZCURPOS	func	out	4	Position of the cursor within the field specified by ZCURFLD when the user exits the panel. The value in ZCURPOS is in character format. If the cursor is not within a field, ZCURPOS will contain a 1.

Note: These variables will contain the values that would result if they were set to .CURSOR, .CSRPOS, and .CSRROW, as the first statements in the panel's)PROC section.

Index

Special Characters

'' (quotation marks), enclosing literals 113
= (equal sign) operator on the IF statement 244
% sign
  beginning a command procedure name with 12
  default attribute character 177
_ (underscore) character, default attribute 177
+ sign
  continuation character for literals 113
  default attribute character 177
)ABC section, defining pull-down choice 164
)ABC section of panel definition 161
)ABCINIT section of panel definition 167
)ABCPROC section of panel definition 168
.ALARM control variable 268
)AREA section of panel definition 169
.ATTR control variable
  considerations 271
  description 269
  override conditions 210
  using with table display panels 270
)ATTR section of panel definition 176
.ATTRCHAR control variable
  description of 270
  dynamic area override 150
  override conditions 210
.AUTOSEL control variable 272
)BLANK file-tailoring control statement 303, 308
)BODY section of panel definition 212
)BODY statement, WINDOW keyword 111
.CCSID section of message definition 297
)CCSID section of panel definition 218
)CM file-tailoring control statement 310
.CSRPOS control variable 272
.CSRROW control variable 273
.CURSOR control variable
  description 274
  example 268
  when not initialized or set to blank 274
÷= operator on the IF statement 244
÷> operator on the IF statement 244
÷< operator on the IF statement 244
)DEFAULT skeleton control statement 306
)DOT file-tailoring control statement 309
)END section of panel definition 219
)END statement, required on panel definition 112
)ENDDOT file-tailoring control statement 309
)ENDSEL file-tailoring control statement 309
> (greater than) operator on the IF statement 244
>= operator on the IF statement 244
<= operator on the IF statement 244
< operator on the IF statement 244
.HELP control variable
  description 275, 285, 293
  example 268
)HELP section of panel definition 219
.HHELP control variable
  description 275
)IM file-tailoring control statement 305
)INIT section of panel definition 220
.KANA control variable in messages 293
)LIST section of panel definition 221
)MODEL section of panel definition 222
.MSG control variable
  description 275
  in batch mode 38
  panel user exit messages 251
.NRET control variable 276
)PANEL statement KEYLIST parameter 222
.PFKEY control variable 277
)PNTS statement 226
)PROC section of panel definition 230
)REINIT section of panel definition 230
.RESP control variable
  description 277
  in batch mode 37
)SEL file-tailoring control statement 309
)SET file-tailoring control statement 310
)TB file-tailoring control statement 307
)TBA file-tailoring control statement 307
.TRAIL control variable
  description 278
  example 118, 235
.TYPE keyword, message definition 294
.WINDOW keyword, message definition 294
.ZVARS control variable
  description 278, 279
  example 279
.ZVARS control variable, associating a PDC with a variable name in)ABCINIT 166

Numerics

3278 Mod 5
  batch mode 38
  graphics interface mode 154
3290
  batch mode 38
  graphics interface mode 154
908 error return code 23
920 error return code 23
985 error return code 23
987 error return code 23
988 error return code 23
989 error return code 23
990 error return code 23
997 error return code 23
998 error return code 23
999 error return code 24

A

A, used to specify alternate tabbing 307
ABCINIT section of panel definition 167
ABCPROC section of panel definition 168
abend
  diagnostic panels 340
ABEND
  codes 341
  description 26
accessing table data 73, 371
action bar choice initialization panel definition section
  definition 167
action bar choice processing section of panel definition
  definition 168
action bar choice section of panel definition
  definition 161
action bars and pull-down choices 94
ADDPOP parameter on ISPSTART command 10
ADDPOP service 93, 94
ADDSOSI built-in function on assignment statement 238
alarm indicator message 370
ALARM keyword, message definition 294
ALPHA parameter on VER statement 256
ALPHAB parameter on VER Statement 256
alternate tabbing 307
APL keyboard character translations 327
APL2
  multiple calls of 32, 371
  number of times invoked, system variable containing 362
  using 30, 371
  workspace used as the function pool 33, 371
application-id parameter on ISPSTART 10, 16
application identifier, system variable 362
application keylist 93
application profile pool 63, 68, 371
application profile pool extension name, system variable 363
AREA(DYNAMIC) parameter in)ATTR section 179
AREA(SCRL) parameter in)ATTR section 183
area section of panel definition
  definition 169

© Copyright IBM Corp. 1980, 1999

array of variable lengths on panel user exit parameter 251
array of variable names on panel user exit parameter 251
ASIS parameter
  in)BODY header statement 214
  on VGET panel statement 264
  on VPUT panel statement 266
  on VPUT statement 264
  with JUST keyword 192
aspect ratio system variable for PRINTG 368
assignment statement in panel definition 234
attention exits (CLIST) 28
ATTN keyword in)ATTR section 184
ATTN statement 28
attribute characters
  default 177
  restriction 178
attribute section of panel definition
  basic attribute types 204
  CUA attribute types 206
  default characters 177
  definition 176
  other attribute types 208
  requirements for table display panel 139
authorized programs, invoking 27
authorized TSO commands, invoking 27
AUTOSEL (.AUTOSEL) control variable 272
AUTOSEL (auto-selection) 136
autoskip
  description 202
  graphic area 154

B

BACK tutorial command 285
background display execution 37
background panel processing 37
BARRIER keyword 117
batch display facility, using 37
batch environment
  avoiding loops in batch 39
  display error processing 39
  log and list data sets 39
  processing commands 39
  terminal characteristics 38
  TSO 35
batch execution
  description 35
  TSO error processing 36
  TSO sample job 35
BATSCRD keyword on ISPSTART command 10, 38
BATSCRW keyword on ISPSTART command 10, 38
BDBCS keyword on ISPSTART command 10, 38
BDISPMAX keyword on ISPSTART command 10, 39
BIT parameter on VER statement 256
BKGRND keyword on ISPSTART command 10
BKGRND parameter on ISPSTART 16
BLANK file-tailoring control statement 308
blinking, specifying for HILITE keyword 192
body section of panel definition
  controlling width of panel 212
  defining 212
  definition 212
  formatting message field 214
  requirements 140
  requirements for table display panel 140
  sample 217
Boolean operators on the IF statement 246
bottom-of-data marker
  definition 136
  system variable containing for table display, user defined 369
BREDIMAX keyword on ISPSTART command 10, 39
BRIF service 89, 371
BROWSE service 89, 371
browse service scroll amount, system variable 368
browse services panel definition, scroll field location 106

C

call of ISPF 9, 10
CAPS keyword in panel)ATTR section 140, 178, 184
CCSID parameter of the GETMSG service 314
CCSID section of message definition
  messages tagged 297
CCSID section of panel definition
  definition 218
  extended code page support 314
chain mode, explicit 120
char parameter
  with PAD keyword 196
  with PADC keyword 197
  with PAS keyword 198
character compare on IF statement 244
character level attribute 150
character translations for APL, TEXT and Katakana keyboards 327
CHINESES keyword on the ISPSTART command 10, 18
CHINESET keyword on the ISPSTART command 10, 18
CKBOX keyword in panel)ATTR section 184
CLEAR keyword on)MODEL statement in table display panel 141
CLIST
  attention exits 28
  example of invoking procedure by using ISPSTART command 18
  variables used in procedure 8
CM file-tailoring control statement 310
CMD
  keyword
    in)PROC section 117
    in panel)BODY section 214
CMD *(continued)*
  parameter on ISPSTART command 10
COBOL compiler used with ISPF 2
code page parameter for ISPSTART 15
coded character set identifier, system variable 364
CODEPAGE 15
COLOR keyword in panel)ATTR section 185
COMBO keyword in panel)ATTR section 186
command field
  naming of 106
  naming with the CMD keyword 214
  panel)BODY section 212
  position in panel definition 106
command field of a table display panel 136
command line placement, system variable 363
COMMAND parameter, in panel)PROC section 117
command procedure 65, 371
command tables
  and application IDs 16
  definition of 3
  ISPCMDS system command table 3
command verb after a SETVERB command table action, system variable 365
commands
  ISPF, in batch environment 37
  processing in batch environment 39
comment statements 113
comments, optional display 192
Common User Access (CUA)
  description of ISPF support 93
  dot leaders 108
  keyword values 207
compare character vs. numeric 244
compilers used with ISPF 2
COMPOUND variables 8
concatenation of variables 114
conditional padding of panel field 178
conditional substitution string 304
CONT system variable on tutorial panels 286
continuation character for literals 113
continuation panel 287
CONTROL NONDISPL in batch mode 37
CONTROL service 91, 371
control statements in skeleton definition 303
control variables
  example 268
  in panels 266
  initialization 267
  list of 266
  when reset 267
conversion utility 93
CRASH 26
creating action bars 166
creating panel display dialog elements 5
CRP of top row displayed in most recent table display, system variable 369

CSRGRP(x) keyword in panel)ATTR section 187
CSRPOS (.CSRPOS) control variable 272
CUA guidelines, dot leaders 108
CUADYN 206
CUADYN keyword in panel)ATTR section 187
cursor placement, default 274

D

DANISH keyword on ISPSTART command 10, 18
data records in skeleton definition 303
DATAMOD keyword in)ATTR section of dynamic panels 180
date and time information (system variables) 361
DBCS
  batch mode 38
  command and message fields 213
  data validation 115
  parameter on VER statement 257
  replacement characters 201
  specifying format 191
  specifying search argument format for table services 85, 371
  system variable containing terminal capability 366
  variables
    in messages and file skeletons 155
    on panel definitions 302, 311
  verifying string length (VER LEN) 260
DDL filename
  system variable 362
DDLIST keyword in panel)ATTR section 187
ddname of file tailoring temporary file, system variable 364
debug tools
  ISRABEND 333, 371
  ISRCSECT 333, 371
  ISRFIND 333, 371
  ISRPOINT 333, 371
  ISRTCB 333, 371
  ISRTEST 333, 371
DEFAULT
  attribute or body section statement 177
  skeleton control statement 306
default attribute characters 177
default keylist for DTL Help Panels 284
defining messages 291
delimiter
  system variable 362
delimiters in verified variable 257
DELSOSI built-in function on assignment statement 238
DEPTH keyword in panel)ATTR section 190
determining table size 76, 371
device name system variable for PRINTG 368
diagnosing ISPF abends 340
dialog
  beginning with menu or function 6, 10
  call by using application master menu 19
  control 5
  definition 1
  development of 4
  elements 1
  example 76, 371
  function, languages used for coding 2
  initiation 20
  organization 5
  return codes 23
  running of 10
  scope 22
  termination 22
  variables 7
  writing
    using display services 43, 371
    using file-tailoring services 86, 371
    using miscellaneous services 91, 371
    using PDF services 89, 371
    using table services 72, 371
    using variable services 62, 371
dialog elements
  description 4, 5
  test of 4
dialog function 1
  creation of 4
  description of 2
  dialog, languages used for coding 2
  example 76, 371
  function pools 64, 371
  naming 12
  scope 22
Dialog Tag Language (DTL) 93
dialog variables, format 70, 371
dialog variables, list of 355, 371
directive lines, optional display 192
display error processing in the batch environment 39
display message variations 296
display services 371
  DBCS-related variables 155, 302, 311
  in batch mode 37
displaying a pop-up window 94
DOT file-tailoring control statement 309
DSNAME parameter on VER statement 257
DSNAMEQ parameter on VER statement 257
DUMP keyword on ENVIRON command 338
dynamic area
  character level attribute support 150
  formatting panels 147
dynamic table expansion 47, 136, 371

E

EBCDIC
  parameter on VER statement 257
  specifying format 191
EDIF service 89, 371
EDIREC service 89, 371
EDIT service 89, 371
edit service panel definition, specifying location of scroll field 106
edit service scroll amount, system variable 368
EDREC service 89, 371
elements of a dialog 1
ELSE statement in panel sections 243
ENBLDUMP parameter on ENVIRON command 335
end of displayed data specification 136
END section of panel definition
  definition 219
ENDDOT file-tailoring control statement 309
ENDSEL file-tailoring control statement 309
ENGLISH keyword on ISPSTART command 10, 18
entry point address on diagnostic panel 340
ENUM parameter on VER statement 257
ENVIRON system command 334
environment 1
environment description, system variable 363
EQ operator on the IF statement 244
error conditions for panel user exit 251
ERROR keyword on ENVIRON command 338
error message-id, system variable 370
error panel 39, 371
error processing
  SYSPRT file 22
  TSO batch execution 36
  when put into effect 22
error recovery panel at abend 340
error return codes from dialog to invoking application 23
ESTAE restrictions 34
executable section of a dialog 220, 230
executing APL2 functions 32, 371
EXHELP 97, 281
exit data on panel user exit parameter 250
EXIT keyword in)PROC section 117, 119, 121
EXIT statement in panel section 241
EXIT statements 241
exits, CLIST attention 28
EXPAND keyword in panel)BODY section 212
expected-length operand (on VER LEN) 260
explicit chain mode 120
EXTEND parameter
  in)ATTR section 179, 183
  in graphic areas 181
Extended Code Page Support
  base code pages 320
  CCSIDs supported 318
  description 313
  ISPF-provided translate tables 323
  messages tagged 314
  panels tagged 314

Extended Code Page Support *(continued)*
translate load modules 315
Z variables 313
Extended Code Page Translate Tables Provided by ISPF 323
extended help 97, 281
extended highlighting availability, system variable 367

F

field-level help 97, 219, 281, 371
field-type specification in panel)ATTR section 202
file-tailoring services
example 87, 371
skeleton files 86, 371
writing dialogs 86, 371
file-tailoring skeleton
control statement considerations 305
data record considerations 86, 303, 371
DBCS considerations 311
defining 303
definition 3
sample 311
file tailoring temporary file name, system variable 364
FILEID parameter on VER statement 259
fixed portion of a TBDISPL display 136
FORMAT keyword
in panel)ATTR section 178, 191
in panel)BODY section 212
formatting guidelines for panels 222
FORTRAN compiler used with ISPF 2
FRAME parameter on ISPSTART 16
FRENCH keyword on ISPSTART command 10
FRENCH keyword on the ISPSTART command 18
function 2
function, definition 1
function commands, definition 140, 371
Function key set displayed, system variable 367
Function key settings, system variables 367
Function keys, system variable containing number of 367
function pool 63, 64, 65, 371

G

GDDM
in batch environment 37
interface to 153
GDDM service 92, 371
GE keyword
in panel)ATTR section 191
GE operator on the IF statement 244
GERMAN keyword on ISPSTART command 10, 18
GETMSG service 92, 371
GIF 222
GOTO statement in panel section 241, 243
graphic area, panel definition 181
graphical user interface
batch mode 40
GRPBOX 208
GT operator on the IF statement 244
GUI in batch mode 40
GUI parameter on ISPSTART 10, 14
GUISCRD 15
GUISCRD parameter on ISPSTART 10
GUISCRW 15
GUISCRW parameter on ISPSTART 10
GUIWEB parameter on ISPSTART 15

H

help
extended 97, 281
field-level 97, 281
help for help 281
keys 98, 281
message 281, 282
panel 281, 282
reference phrase 98, 282
TUTOR command 282
tutorial 282
help for help command 281
help panel 115
name associated with error message 370
system variable containing name associated with error message 370
with scrollable areas 171
help section of panel definition
definition 219
HELP system command
entry to tutorial 285
on ABEND panels 341
HEX parameter on VER statement 259
HIGH parameter with INTENS keyword 192
HILITE keyword in panel)ATTR section 192

I

IF statement
basic IF 244
with Boolean operators 246
with VER constructs 245
IM file-tailoring control statement 305
IMAGE keyword 223
IN parameter used with CAPS keyword 184
INCLUDE parameter on VER Statement 259
index page, specifying for tutorials 287
INDEX tutorial command 285
initialization of control variables 267
initialization section of panel definition
definition 220
requirements for table display 142
initiating dialog execution 20
INPUT parameter used with TYPE keyword 203
INTENS keyword in panel)ATTR section 178
invoking
authorized commands 27
invoking *(continued)*
authorized programs 27
authorized TSO commands 27
TSO commands 27
invoking a dialog
from a selection panel 19
from the ISPF master application menu 19
the ISPSTART command 18
ISP@MSTR, ISPF Master Application Menu 121
ISP@PRIM on the ISPF Primary Option Menu 127
ISPCMDS system command table 3
ISPF
command 27, 371
Common User Access support 93
default keylist 284
EDIF service 34, 371
help panels 281
interface with APL2 34, 371
overview 4
tutorial panels 281
variables 70, 371
ISPF conversion utility 93
ISPF Services in Batch Mode 34
ISPPREP preprocessed panel routine
batch environment 39
error conditions 159
examples 159
restrictions 157
return codes 159
using 155
ISPSTART command
description 9, 10
example 10
syntax 10
TSO 27, 371
ISPTTDEF, using to specify translate tables 331
ISPTUTOR 285
ISRABEND debug tool 333, 371
ISRCSECT debug tool 333, 371
ISRFIND debug tool 333, 371
ISRPOINT debug tool 333, 371
ISRROUTE command 166
ISRTCB debug tool 333, 371
ISRTEST debug tool
description 333, 371
ITALIAN keyword on ISPSTART command 10
ITALIAN keyword on the ISPSTART command 18

J

JAPANESE keyword on ISPSTART command 10, 18
JUST keyword in panel)ATTR section 140, 178, 192
justifying a panel field 192

K

KANA keyword
extended code page support 317
on panel)BODY section 212, 327

Katakana
keyboard character translations 327
terminal displaying messages 293
key assignment 93
keylist
application 93
system 93
keylist defaults for DTL Help Panels 284
KEYLIST parameter on)PANEL statement 222
keylist utility 110
keys 284
keys help 98, 281
KEYS system command, batch environment 39
KEYSHELP 98, 281
KOREAN keyword on ISPSTART command 10
KOREAN keyword on the ISPSTART command 18

L

LANG(APL) parameter
in panel)PROC section 117
on ISPSTART command 10
languages used for coding functions 2
last visible line function (LVLINE) 238
LE operator on the IF statement 244
leading blanks in verified variable 257
LEFT parameter used with JUST keyword 192
LEN keyword on VER statement 260
LIBDEF service 92, 371
library access services 90, 371
light pen, using to select a field 184
line display mode, automatic and nonautomatic entry into line mode 11
list data set in a batch environment 39
LIST parameter on VER statement 261
list section of panel definition
definition 221
LIST service 92, 371
LISTBOX keyword in panel)ATTR section 193
LISTV parameter on VER Statement 261
LISTVX parameter on VER Statement 262
LISTX parameter on VER Statement 262
LMSG parameter on panel)BODY section 214
loading a panel user exit routine 249
log data set
batch messages 38
in batch environment 39
LOG service 92, 371
logical screens
system variable 363
logical screens, maximum
system variable 364
LOGO parameter on ISPSTART command 17
LOGOFF command 27, 371
LOGON command 27, 371
long error message text, system variable 370
loops, avoiding in batch 39
LOW parameter used with INTENS keyword 192
LT operator on the IF statement 244
LVLINE built-in function on assignment statement 238

M

master application menu
example of definition 121
example of display 19
member lists scroll amount, system variable 368
menu
definition of primary option 119
entry to tutorial 285
example of a master application menu 121
example of primary option 134
special definition requirements 115, 116
use of ZPARENT to set next display 120
message alarm indicator 370
message definition
DBCS considerations 302, 311
description of 3
example of short and long 292
Katakana considerations 293
message ID 292
processing 291
syntax 292, 301
message field location 105
message fields in panel)BODY section 212
message help 281, 282, 293
message-id, system variable containing error 370
message ID on panel user exit parameter 251
message library
description of 291
example 292
message text
long error 370
short error 370
system variable containing 370
messages
display variations 296
in batch environment 38
miscellaneous services, used in writing dialogs 91, 371
MIX parameter on VER statement 262
mixed characters, specifying format 191
MODE keyword 117, 119
model lines
definition of 137
specified in a variable 142
model section of panel definition
definition 222
requirements for table display panel 141
model sets
description of 137
example 45
modeless message pop-ups 298
module name on diagnostic panel 340
moveable pop-ups
manual movement 97
moveable pop-ups *(continued)*
WINDOW command 96
MSG=value parameter on assignment statement 235
msgid keyword 292

N

NAME parameter on VER statement 262
naming defined and implicit variables 67, 371
naming restrictions for dialog functions 12
NB parameter on VER statement 256
NE operator on the IF statement 244
negative number indicators 257
NEST keyword 117
nested CLISTS, attention exits 30
NEWAPPL, (application-id) parameter 10, 117
NEWPOOL parameter in)PROC section 117
NG operator on the IF statement 244
NL operator on the IF statement 244
NLS
common characters 313, 371
GETMSG service 314, 371
messages tagged with CCSID 297, 371
TRANS service 315, 371
NOCHECK parameter
example 118
in)PROC section 117
NOKANA keyword in message definition 293
NOLOGO parameter on ISPSTART command 17
NON parameter used with INTENS keyword 192
NONBLANK parameter on VER statement 256
null system variable 362
NULLS parameter used with PAD keyword 196
NUM parameter on VER statement 262
number of colors supported by the terminal type, system variable 366
number of Function keys, system variable 367
number of variables on panel user exit parameter 251
numeric (extended) verification 257
numeric compare on IF statement 244
NUMERIC keyword in panel)ATTR section 195
Numeric Lock feature (with NUMERIC attribute keyword) 195

O

OFF parameter
with ATTN keyword 184
with CAPS keyword 184
with NUMERIC keyword 196
with SKIP keyword 202
ON parameter
with ATTN keyword 184

ON parameter *(continued)*
with CAPS keyword 184
with NUMERIC keyword 196
with SKIP keyword 202
ONEBYTE built-in function on assignment statement 239
online tutorial 285
OPT(option) parameter on ISPSTART command 10
OPT system variable 116
OUT parameter used with CAPS keyword 184
OUTLINE keyword
in panel)ATTR section 177, 178, 196
in panel)BODY section 212, 216
OUTPUT parameter used with TYPE keyword 203

P

PAD keyword in panel)ATTR section 178, 196
PADC keyword in panel)ATTR section 197
panel definition 104
)PNTS statement 226
attribute section
default characters 177
blanks 112
body section
sample 217
command field
description 105
specifying 212
comment statement 112
creation of 5
description 104, 105
design suggestions 108
dynamic areas 205
graphic areas 181
GUI considerations 141, 154
help and tutorial panels 285
initialization section
statement formats 233
line 1 content 106
line 2 content 106
line 3 content 106
location 105
menus 115
model section 141
panel title, location 105
reinitialization section
statement formats 233
restrictions 112
sections 104
short message for TBDISPL operations 106
size 111
special requirements 115
specifying a message field 214
split-screen consideration 108
syntax rules 112
table display 134
tutorial and help panels 285
using)PANEL 222
panel help 281, 282
panel name on panel user exit parameter 250
PANEL parameter
in)PROC section 117
on ISPSTART command 10
panel redisplay 231
panel section of panel definition
formatting panel 222
panel section on panel user exit parameter 250
panel user exit routine
description 247
how to invoke 249
how to load 249
parameters passed 250
return codes 251
panels
preprocessed 155
vertically scrollable 111
PANEXIT statement 247
PARM
keyword
in)PROC section 117
on preprocessed panels 156
parameter on ISPSTART command 10
parts of a dialog 1
PAS keyword in panel)ATTR section 198
passing control from program-coded to command-coded function 6
PDF command 27, 371
PDF service
library access 90, 371
where to find examples and listings 90, 371
writing dialogs 89, 371
pending END request 137
pending scroll request 137
pending selected rows 137
percent (%) sign, beginning a command procedure name with 11
PFK built-in function on assignment statement 237
PFkey, system variable 363
PGM keyword in)PROC section 117
PGM parameter on ISPSTART command 10
PICT parameter on VER statement 262
PL/I compiler used with ISPF 2
pools, variable
application profile 63, 371
function 63, 371
shared 63, 371
pop-up window
ADDPOP service 94
moveable 95
processing considerations 154
size 213
PORTUGUESE keyword on ISPSTART command 10, 18
POSITION, TBDISPL parameter 138, 371
PQUERY
in batch environment 38
used with dynamic area 150
PQUERY service 92, 371
prefix system variable 363
preprocessed panels
creating (ISPPREP) 155
preprocessed panels *(continued)*
definition 155
ISPPREP call 157
PARM keyword 156
SELECT service 156
Primary Option Menu 119
printer family type for PRINTG 369
processing section of panel definition
definition 230
requirements for table display 143
PROFILE parameter
on VGET panel statement 265
on VPUT panel statement 264, 266
program-name parameter
in panel)PROC section 117
on ISPSTART command 10
program status word on diagnostic panel 340
protecting table resources 75, 371
PSW on diagnostic panel 340
pull-down choice, defining within the)ABC section 164
pushbuttons 226
pushbuttons, large 226

Q

QUERY parameter on the ENVIRON command 340
quotation mark, enclosing literals 113
quote mark, enclosing literals 113

R

RADIO keyword in panel)ATTR section 199
RANGE parameter on VER statement 262
read-only profile pool extension variables 68, 371
reason code on diagnostic panel 340
recovery termination manager at abend 343
redisplay of a panel 231
reference phrase help 98, 282
REFRESH statement in panel sections 252
register content at abend on diagnostic panel 340
reinitialization section of panel definition
definition 230
requirements for table display 143
relational operators (on VER LEN) 260
removing a pop-up window 94
removing variables from the shared or profile pool 68, 371
REMPOP service 93, 94
REP keyword in panel)ATTR section 178, 201
replacement characters 201
reset of control variables 267
return codes
for panel user exit routine 251
from terminating dialog 23
return to function when scrolling 47, 371

REVERSE parameter used with HILITE keyword 192
reverse video, specifying 192
RIGHT parameter used with JUST keyword 192
ROWID, TBDISPL parameter 371
ROWS keyword on)MODEL statement in table display panel 141
rows of a table, adding dynamically 47, 51, 371
running a dialog 10

S

scope of a function 22
screen
  logical number of 368
  system variable containing 368
screen depth and width available for use by a dialog
  system variable containing 368
screen depth and width available for use by a dialog, system variable 368
screen depth on ISPSTART command for batch 10
screen depth parameter for ISPSTART 15
screen name
  system variable 363
screen width for batch mode on ISPSTART command 10
screen width parameter for ISPSTART 15
scroll amount
  field of a TBDISPL display, definition of 137
  for browse service, system variable containing 368
  for edit service, system variable containing 368
  for member lists, system variable containing 368
  location 105
  system variable containing 368
  system variable containing field value 368
  system variable containing number of lines or columns to 368
  value default for dynamic areas and table display 368
SCROLL parameter in)ATTR section 179
scrollable areas
  definition, section of panel 169
  in the)BODY section 183
  vertically scrollable panels 111
  with help panel 171
scrollable portion of a TBDISPL display 137
scrolling, expanding displayed table 48, 371
SDWA reason code at abend 341
searching variable pools 63, 371
SEL
  file-tailoring control statement 309
  system variable 116, 286
select field of a TBDISPL display 138
SELECT service 63
SELECT service 63 *(continued)*
  call 22
  controlling access to dialog variable pools 371
  description 20
  panel (VGET) 265
  panel processing 117
  passing control in a dialog 63, 371
  preprocessed panels 156
Selected Choice (SC) attribute 210
selected row, defined 138
selection panel, system variables 370
separator
  system variable 362
  system variable containing 364
services
  to dialogs 1
  to interactive applications 1
services description, SELECT 20
SET file-tailoring control statement 310
SGERMAN keyword on the ISPSTART command 10, 18
shadow variable 150
SHARED parameter
  on VGET panel statement 264
  on VPUT panel statement 264, 266
shared pool 63, 371
sharing variables among dialogs 67, 371
shift-in character (DBCS) 191, 238
shift-out character (DBCS) 191, 238
short error message text, system variable 370
short message syntax 293
site command table prefix, system variable 364
skeleton
  description of 3
skeleton definition
  assigning a value to a variable 310
  comment statement 310
  control statements 305
  defining 303
  description of 303
  example 311
  format of 303
  imbedding 305
  imbedding blank lines 308
  specifying table processing 309
  tab stop 307
  types of records in 303
SKIP
  keyword in panel)ATTR section 178, 202
  tutorial command 285
SMSG parameter on panel)BODY section 214
SPANISH keyword on ISPSTART command 10
SPANISH keyword on the ISPSTART command 18
specifying DBCS search argument format 85, 371
SPF command 27, 371
SPLIT command, disabled in batch environment 39
split-screen in effect, system variable 368
SPLITV system command, disabled in batch environment 39
stacked commands, graphics interface mode restriction 153
standard tabbing 307
START service 100, 371
starting a dialog
  methods 10
  using the ISPSTART command 18
  using the SELECT service 20
starting ISPF 9, 10
STEM variables 8
stepname of TSO logon, system variable 363
storing variables from a panel in shared and profile pools (VPUT) 266
string of variable values on panel user exit parameter 251
substitution string, conditional 304
subtasking support 34
syntax rules
  message definition 292, 301
  panel definition 112
  skeleton definitions 303
System keylist 93
system variable containing number of lines or columns to scroll 368
system variables 371
  list of 361
  used for communication between dialogs and ISPF 370
  Z 362
  ZACCTNUM 362
  ZAPLCNT 362
  ZAPPLID 362
  ZAPPTTL 362
  ZASPECT 368
  ZCMD 370
  ZCOLORS 366
  ZCONT 370
  ZCS 362
  ZCSDLL 362
  ZCURFLD 370
  ZCURINX 370
  ZCURPOS 370
  ZDATE 361
  ZDATEF 361
  ZDATEFD 361
  ZDATESTD 361
  ZDAY 361
  ZDBCS 366
  ZDECS 362
  ZDEL 362
  ZDEVNAM 368
  ZENTKTXT 362
  ZENVIR 363
  ZERRALRM 370
  ZERRHM 370
  ZERRLM 370
  ZERRMSG 370
  ZERRSM 370
  ZERRTYPE 370
  ZERRWIND 370
  ZEURO 363
  ZFAMPRT 369
  ZFKA 367
  ZGE 367

system variables 371 *(continued)*
ZGUI 363
ZHILITE 367
ZHINDEX 370
ZHTOP 370
ZIND 370
ZISPFOS 363
ZISPFRC 363
ZJ4DATE 361
ZJDATE 361
ZKEYHELP 363
ZKEYS 367
ZKLAPPL 367
ZKLNAME 367
ZKLTYPE 367
ZKLUSE 367
ZLANG 363
ZLOGNAME 369
ZLOGO 363
ZLOGON 363
ZLSTLPP 369
ZLSTNAME 369
ZLSTNUML 369
ZLSTTRUN 369
ZMONTH 362
ZOS390RL 363
ZPARENT 370
ZPF01-24 367
ZPFCTL 367
ZPFFMT 367
ZPFKEY 363
ZPFLxx 367
ZPFSET 367
ZPFSHOW 367
ZPLACE 363
ZPREFIX 363
ZPRIKEYS 367
ZPRIM 370
ZPROFAPP 363
ZSCBR 368
ZSCED 368
ZSCML 368
ZSCRCUR 363
ZSCREEN 368
ZSCREEND 368
ZSCREENW 368
ZSCRMAX 364
ZSCRMAXD 368
ZSCRMAXW 368
ZSCROLLA 368
ZSCROLLD 368
ZSCROLLN 368
ZSCTPREF 364
ZSEL 370
ZSPLIT 368
ZSTDYEAR 362
ZSYSICON 364
ZSYSID 364
ZSYSNODE 364
ZSYSPLEX 364
ZTDADD 369
ZTDAMT 369
ZTDLROWS 369
ZTDLTOP 369
ZTDMARK 369
ZTDMSG 369
ZTDRET 369

system variables 371 *(continued)*
ZTDROWS 369
ZTDSCRP 369
ZTDSELS 369
ZTDSIZE 369
ZTDSRID 369
ZTDTOP 369
ZTEMPF 364
ZTEMPN 364
ZTERM 368
ZTERMCID 364
ZTERMCP 364
ZTERMCS 364
ZTHS 364
ZTIME 362
ZTIMEL 362
ZTS 364
ZTSICMD 364
ZTSSCMD 364
ZUCTPREF 364
ZUCTSRCH 364
ZUP 370
ZUSER 365
ZVERB 365
ZWINTTL 365
ZWSCDPG 365
ZWSCON 365
ZWSOPSYS 365
ZYEAR 362
SYSTSPRT file for error messages 37

T

tab stop in skeleton definition 307
tabbing
alternate 307
standard 307
table
accessing data 73, 371
adding rows dynamically 47, 371
definition 3
dynamic expansion 136
temporary or permanent 73, 371
when created or updated 3
table display (TBDISPL), terms related to 135
table display output example 146, 147
table display panel definition
attribute section 140
body section 140
example 144
example of multiple model lines 146
initialization section 142
message location 106
model line 45, 135
model section 141
scroll field location 106
short message area content 106
using the TBDISPL service 134
table rows
number of selected upon return from table display 369
number of system variable containing upon return from table display 369
system variable containing 369
table services
determining table size 76, 371
example 75, 76, 371

table services *(continued)*
protecting resources 75, 371
row operation 74, 371
using 72, 74, 371
tags, creating dialog elements 93
task abend code on diagnostic panel 340
TB file-tailoring control statement 307
TBA file-tailoring control statement 307
TBDISPL series 138
TBDISPL service
description 144, 371
dynamically building the table 48, 371
terms related to 135
writing dialogs 43, 371
terminal data in batch mode 38
terminal type
specifying ISPTTDEF 331
system variable containing 368
terminating
a dialog 22
ISPF 9, 10
TERMSTAT parameter on ENVIRON command 339
TERMTRAC parameter on ENVIRON command 335
TEST 371
difference from TESTX 26
mode 25
parameter on ISPSTART command 10
testing dialog elements 4
TESTX
difference from TEST 26
mode 26
parameter on ISPSTART command 17
TEXT keyboard character translations 327
TEXT parameter used with TYPE keyword 203
time and date information (system variables) 361
title displayed in window frame 362
TOC tutorial command 285
TOG statement 253
top-row-displayed indicator 50, 138, 369, 371
TRACE
difference from TEST and TRACEX 27
mode 27
parameter on ISPSTART command 10
TRACEX
difference from TEST and TRACE 27
mode 27
parameter on ISPSTART command 17
trailing blanks in verified variable 257
TRANS built-in function on assignment statement
description 235, 236
example 118, 237, 274
example, nested 235, 236
translate tables, specifying 331

translation
common characters 313, 371
GETMSG service 314, 371
messages tagged with CCSID 297, 371
TRANS service 315, 371
TRUNC built-in function on assignment statement
description 234
example 118, 234, 237
example, nested 235, 236
truncation, system variable containing list data set 369
TSO
batch environment 35
batch execution 35
command restrictions 27
invoking authorized commands 27
invoking commands 27
TSO command 27, 371
TUTOR command 282
tutorial 115
call of 285
commands 285
defining panels 285
description 282
ending of 286
entry to 285
sample hierarchy of panels 287
sample panel 288
specifying an index page 287
use 285
tutorial panels, system variables that contain information about 370
TWOBYTE built-in function on assignment statement 239
TYPE keyword in panel)ATTR section 178

U

unavail specification in panel)ATTR section 203
underscore, specifying 192
UP tutorial scroll command 285
UPPERENG keyword on the ISPSTART command 10, 18
USCORE parameter used with HILITE keyword 192
used for communication between dialogs and ISPF 70, 371
user exit for panel processing 247
USER parameter used with PAD keyword 196
user-selection 139
userid, system variable 365
USERMOD parameter in)ATTR section 180

V

validation of DBCS data 115
value from scroll amount field, system variable 368
variable model lines 142
variable services
creating or deleting defined variables 66, 371
summary 72, 371
writing dialogs 62, 371
variables
assignment statement 234
COMPOUND 8
creating implicit 66, 371
description of 7
dialog 63, 371
dialog, format 70, 371
in IF or ELSE statements 243
in message definition 301
in VER statements 255
maximum size 7
names too long for panel definition 279
naming 7
naming defined and implicit 67, 371
on panels, restricted size 112
owned by ISPF 70, 371
processing using panel user exit 248
read-only extension 68, 371
removing from the shared or profile pool 68, 371
saving across ISPF sessions 67, 371
sharing among dialogs 67, 371
STEM 8
storing from a panel to shared and profile pools (VPUT) 266
system variable charts 361
testing the value of 243
to function pool from shared or profile pools (VGET) 264
value test during panel processing 245
ZERRCSID 313
ZKEYHELP 98
ZTERMCID 313
ZTERMCP 313
ZTERMCS 313
Variables for ISPSTART parameters 11
VARS variable in table display panel 142
VCOPY service 71, 371
VDEFINE service
in panel user exit routine 248
writing dialogs 71, 371
VDELETE service 71, 371
VEDIT statement 254
VER statement in panel section
description 255
syntax 256
VERASE service 72, 371
verifying variable content 256
VGET statement
in panel)INIT,)REINIT, or)PROC section 264
on DISPLAY panel 264
on SELECT panel 265
syntax 264
using 72, 371
VMASK service 71, 371
VPUT statement
example 266
VPUT statement *(continued)*
in panel)INIT,)REINIT, or)PROC section 266
syntax 266
using 72, 371
VREPLACE service 71, 371
VRESET service 71, 371

W

WIDTH keyword in panel)ATTR section 203
WIDTH keyword in panel)BODY section 212
WINDOW command 96
WINDOW keyword
defining pop-up windows 111
in panel)BODY section 213
window title variable 93, 94
workstation command 13
workstation command var 14
writing dialogs
display services 43, 371
file-tailoring services 86, 371
miscellaneous services 91, 371
PDF services 89, 371
table services 72, 371
variable services 62, 371
WSCMD 13
WSCMD parameter on ISPSTART command 10
WSCMDV 14
WSCMDV parameter on ISPSTART command 10

Z

Z system variable 362
Z variables used for field name place-holders 279
ZACCTNUM system variable 362
ZAPLCNT system variable 362
ZAPPLID system variable 362
ZAPPTTL system variable 362
ZASPECT system variable 368
ZC system variable 302, 312
ZCMD 355, 371
ZCMD system variable 370
example 118
on tutorial panels 286
processing
blank 119
invalid option 119
truncation 117
versus other names for command field 106
ZCOLORS system variable 366
in batch mode 38
ZCONT system variable 287, 289, 370
ZCS system variable 362
ZCSDLL system variable 362
ZCUNIT 360, 371
ZCURFLD
general description 226
ZCURFLD system variable 370
ZCURINX
general description 226

ZCURINX system variable 370
ZCURPOS
general description 226
ZCURPOS system variable 370
ZCUSIZE 360, 371
ZDATE system variable 361
ZDATEF system variable 361
ZDATEFD system variable 361
ZDATESTD system variable 361
ZDAY system variable 361
ZDBCS system variable 366
in batch mode 38
ZDECS system variable 362
ZDEL system variable 362
ZDEVNAM system variable 368
ZDLBLKSZ 355, 371
ZDLCDATE 355, 371
ZDLDEV 355, 371
ZDLDSNTP 355, 371
ZDLDSORG 355, 371
ZDLEDATE 355, 371
ZDLEXT 355, 371
ZDLLRECL 355, 371
ZDLMIGR 355, 371
ZDLRDATE 355, 371
ZDLRECFM 355, 371
ZDLSIZE 355, 371
ZDLSPACU 355
ZDLUSED 355, 371
ZDLVOL 355, 371
ZDSN 356, 371
ZDST 356
ZE system variable 302, 312
ZEDBDSN 356, 371
ZEDROW 356, 371
ZEDSAVE 356
ZEDTDSN 356, 371
ZEDTMCMD 356
ZEDTMEM 356, 371
ZEDTRD 356, 371
ZEDUSER 356, 371
ZEIBSDN 356, 371
ZEIROW 356, 371
ZEITDSN 356, 371
ZEIUSER 356, 371
ZENTKTXT 371
ZENVIR system variable 35, 363
ZERRALRM 356, 371
ZERRALRM system variable 370
ZERRHM 356, 371
ZERRHM system variable 370
ZERRLM 357, 371
ZERRLM system variable 370
ZERRMSG 357, 371
ZERRMSG system variable 370
for panel user exit messages 252
ZERRSM 357, 371
ZERRSM system variable 370
ZERRTYPE system variable 370
ZERRWIND system variable 370
ZEURO system variable 363
ZFAMPRT system variable 369
ZFKA system variable 367
ZGE system variable 151, 367
ZGRPLVL 357, 371
ZGRPNME 357, 371
ZGUI system variable 363
ZHILITE system variable 367
in batch mode 38
ZHINDEX system variable 370
example 127
specifying top indexed panel 286
ZHTOP system variable 370
example 127
specifying top tutorial panel 286
ZICFPRT 360, 371
ZIND system variable 370
using on tutorial panels 287
ZISPFOS system variable 363
ZISPFRC system variable
description 23
example of using 24
return codes 363
ZJ4DATE system variable 361
ZJDATE system variable 361
ZKEYHELP system variable 98, 363
ZKEYS system variable 367
ZKLAPPL system variable 367
ZKLNAME system variable 367
ZKLTYPE system variable 367
ZKLUSE system variable 367
ZLAC 357
ZLALIAS 357
ZLAMODE 357
ZLANG system variable 363
ZLATTR 357
ZLC4DATE 357
ZLCDATE 357, 371
ZLCNORC 357, 371
ZLINORC 358, 371
ZLLIB 358, 371
ZLM4DATE 358
ZLMDATE 358, 371
ZLMEMBER 358, 371
ZLMNORC 356, 358, 371
ZLMOD 358, 359, 371
ZLMSEC 358, 371
ZLMTIME 358, 359, 371
ZLOGNAME system variable 369
ZLOGO system variable 363
ZLOGON system variable 363
ZLPDSUDA 358, 371
ZLRMODE 359
ZLSIZE 359
ZLSTLPP system variable 369
ZLSTNAME system variable 369
ZLSTNUML system variable 369
ZLSTTRUN system variable 369
ZLTTR 359
ZLUSER 359, 371
ZLVERS 359, 371
ZMLCOLS 359, 371
ZMLCR 359, 371
ZMLTR 359, 371
ZMONTH system variable 362
ZPARENT system variable 120, 370
ZPDFREL 360, 371
ZPF01-24 system variables 367
ZPFCTL system variable 367
ZPFFMT system variable 367
ZPFKEY system variable 363
ZPFSET system variable 367
ZPFSHOW system variable 367
ZPLACE system variable 363
ZPREFIX system variable 363
ZPRIKEYS system variable 367
ZPRIM system variable 370
example 119, 127
ignored in explicit chain mode 121
using 120
ZPROFAPP system variable 363
ZSCBR system variable 368
ZSCED system variable 368
ZSCLM 359
ZSCML system variable 368
ZSCRCUR system variable 363
ZSCREEN system variable 368
ZSCREEND system variable 368
in batch environment 38
ZSCREENW system variable 368
in batch environment 38
ZSCRMAX system variable 364
ZSCRMAXD system variable 368
in batch environment 38
panel definition 111
ZSCRMAXW system variable 368
in batch environment 38
panel definition 111
ZSCROLLA system variable 140, 368
ZSCROLLD system variable 140, 368
ZSCROLLN system variable 140, 368
ZSCTPREF system variable 364
ZSEL system variable 370
contains result of truncating
ZCMD 116
example 118
on menus 116
on tutorial panels 286
parameters and keywords used
with 117
restriction for 286
ZSESS 360, 371
ZSPLIT system variable 368
ZSTDYEAR system variable 362
ZSWIND 360
ZSYSICON system variable 364
ZSYSID system variable 364
ZSYSNODE system variable 364
ZSYSPLEX system variable 364
ZTDADD function variable
definition of 47, 371
using 49, 371
ZTDADD system variable 369
ZTDAMT function variable
definition of 47, 371
using 49, 371
ZTDAMT system variable 369
ZTDLROWS function variable
definition of 48, 371
using 50, 371
ZTDLROWS system variable 369
ZTDLTOP function variable
definition of 48, 371
using 48, 50, 371
ZTDLTOP system variable 369
ZTDMARK system variable 136, 369, 371
ZTDMSG system variable 369, 371
ZTDRET function variable
definition of 47, 371
using 47, 371

ZTDRET system variable 369
ZTDROWS system variable 369, 371
ZTDSCRP function variable
  definition of 47, 371
  using 49, 371
ZTDSCRP system variable 369
ZTDSELS system variable 143, 369
  description 46, 371
  example 46, 371
ZTDSIZE function variable
  definition of 48, 371
  using 50, 371
ZTDSIZE system variable 369
ZTDSRID function variable
  definition of 47, 371
  using 49, 371
ZTDSRID system variable 369
ZTDTOP system variable 369, 371
ZTEMPF system variable 364
ZTEMPN system variable 364
ZTERM, mapped to APL2 terminals 31, 371
ZTERM system variable 368
ZTERMCID system variable 364
ZTERMCP system variable 364
ZTERMCS system variable 364
ZTHS system variable 364
ZTIME system variable 362
ZTIMEL system variable 362
ZTS system variable 364
ZTSICMD system variable 364
ZTSSCMD system variable 364
ZUCTPREF system variable 364
ZUCTSRCH system variable 364
ZUP system variable 370
  on tutorial panels 286
ZUSER system variable 365
ZUSERMAC 359
ZVERB system variable 140, 365
ZWINTTL 94
ZWINTTL system variable 365
ZWSCDPG system variable 365
ZWSCON system variable 365
ZWSOPSYS system variable 365
ZYEAR system variable 362

Readers' Comments — We'd Like to Hear from You

Interactive System Productivity Facility (ISPF)
Dialog Developer's Guide and Reference
OS/390 Version 2 Release 8.0

Publication No. SC28-1273-03

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	☐	☐	☐	☐	☐

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	☐	☐	☐	☐	☐
Complete	☐	☐	☐	☐	☐
Easy to find	☐	☐	☐	☐	☐
Easy to understand	☐	☐	☐	☐	☐
Well organized	☐	☐	☐	☐	☐
Applicable to your tasks	☐	☐	☐	☐	☐

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? ☐ Yes ☐ No

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Name _____ Address _____

Company or Organization _____

Phone No. _____

Readers' Comments — We'd Like to Hear from You
SC28-1273-03

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 MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
Information Development
Department CGMD / Bldg 062
P.O. Box 12195
Research Triangle Park, NC
27709-2195

Fold and Tape | **Please do not staple** | Fold and Tape

SC28-1273-03

Cut or Fold
Along Line

File Number: S370/4300-39
Program Number: 5647-A01

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SC28-1273-03