Interactive System Productivity Facility (ISPF)

# Services Guide

*z/OS Version 1 Release 6.0*

SC34-4819-03

Interactive System Productivity Facility (ISPF)

# Services Guide

*z/OS Version 1 Release 6.0*

SC34-4819-03

**Note**
Before using this document, read the general information under “Notices” on page 305.

**Fourth Edition (September 2004)**

This edition applies to ISPF for Version 1 Release 6.0 of the licensed program z/OS (program number 5694-A01) and to all subsequent releases and modifications until otherwise indicated in new editions.

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:

IBM Corporation
Department J87/D325
555 Bailey Avenue
San Jose, CA 95141-1003
U.S.A.

Internet: comments@us.ibm.com

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

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

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

The ISPF development team maintains a site on the World Wide Web. The URL for the site is:
http://www.ibm.com/software/awdtools/ispf/

**© Copyright International Business Machines Corporation 1980, 2004. All rights reserved.**
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

# Contents


**Figures . . . . . ix**

**Preface . . . . . xi**
Who Should Use This Document? . . . . . xi
What Is In This Document? . . . . . xi
Using LookAt to look up message explanations . . xi

**Summary of Changes . . . . . xiii**
ISPF Product and Library changes . . . . . xiii
ISPF Dialog Manager component changes . . . . xiii
Dialog Tag Language (DTL) changes. . . . . xiv
ISPF PDF Component changes. . . . . xiv
ISPF SCLM Component changes . . . . . xiv
ISPF Client/Server Component changes . . . . . xv
ISPF Migration Considerations . . . . . xv

**What's in the z/OS V1R6.0 ISPF library? . . . . . xvii**
z/OS V1R6.0 ISPF. . . . . xvii

**The ISPF User Interface . . . . . xix**
Some Terms You Should Know . . . . . xix
How to Navigate in ISPF Using the Action Bar Interface . . . . . xx
Action Bars . . . . . xx
Command Nesting . . . . . xxii
Action Bar Choices . . . . . xxiii
Point-and-Shoot Text Fields . . . . . xxiv
Function Keys . . . . . xxv
Selection Fields . . . . . xxvi
How to Navigate in ISPF without Using Action Bars . . . . . xxvi

**Chapter 1. Introduction to ISPF Services 1**
Description of the Services. . . . . 1
Notation Conventions . . . . . 2
Using ISPQRY to Test Whether ISPF Is Active . . . 2
Invoking the ISPF Services. . . . . 2
Load Module Search Order . . . . . 3
Invoking Services from Command Procedures . . 3
Invoking ISPF Services with Program Functions . 5
Return Codes from Services . . . . . 12
Command Invocation Return Code Variable . . 13
Call Invocation Return Code Variables . . . . 13
Return Code of 12 or Higher . . . . . 13
System Variables Used to Format Error Messages 14
Return Codes from I/O and Command Routines 15
A Summary of the ISPF Services . . . . . 15
Display Services . . . . . 15
File Tailoring Services . . . . . 15
Library Access Services . . . . . 16
PDF Component Services. . . . . 17
Table Services . . . . . 17
Variable Services. . . . . 18
Miscellaneous Services. . . . . 19

**Chapter 2. Description of the ISPF Services . . . . . 21**
ADDPOP – Start Pop-Up Window Mode . . . . 21
Command Invocation Format . . . . . 21
Call Invocation Format . . . . . 21
Parameters . . . . . 21
Return Codes. . . . . 22
Example . . . . . 22
BRIF – Browse Interface . . . . . 23
Command Invocation Format . . . . . 23
Call Invocation Format . . . . . 24
Parameters . . . . . 24
Dialog-Supplied Routines. . . . . 25
Read Routine . . . . . 25
Command Routine . . . . . 26
Return Codes. . . . . 26
BROWSE – Browse a Data Set . . . . . 28
Command Invocation Format . . . . . 28
Call Invocation Format . . . . . 28
Parameters . . . . . 28
Return Codes. . . . . 30
Example . . . . . 30
CONTROL – Set Processing Modes . . . . . 31
Command Invocation Format . . . . . 31
Call Invocation Format . . . . . 31
ADDPOP/REMPOP Service Relation to Control Service . . . . . 32
Parameters . . . . . 32
Return Codes. . . . . 37
Example . . . . . 37
DISPLAY – Display Panels and Messages . . . . 38
Command Invocation Format . . . . . 38
Call Invocation Format . . . . . 38
Parameters . . . . . 39
Using the COMMAND Option . . . . . 41
Return Codes. . . . . 42
Example 1 . . . . . 42
Example 2 . . . . . 42
Example 3 . . . . . 43
Example 4 . . . . . 43
Example 5 . . . . . 44
DSINFO – Data Set Information Dialog Service . . 44
Command Invocation Format . . . . . 44
Call Invocation Format . . . . . 44
Parameters . . . . . 44
Return Codes. . . . . 46
Example . . . . . 46
EDIF – Edit Interface . . . . . 46
Command Invocation Format . . . . . 47
Call Invocation Format . . . . . 47
Parameters . . . . . 47
Dialog-Supplied Routines. . . . . 49
Read Routine . . . . . 50


© Copyright IBM Corp. 1980, 2004

Write Routine. . . . . 50
Command Routine . . . . . 51
Return Codes. . . . . 52
Read Routine. . . . . 52
Write Routine Return Codes. . . . . 52
Command Routine Return Codes . . . . . 52
EDIF Service Return Codes . . . . . 52
Example . . . . . 52
EDIREC - Initialize Edit Recovery . . . . . 53
Command Invocation Format . . . . . 53
Call Invocation Format . . . . . 53
Parameters . . . . . 53
Return Codes. . . . . 54
Example . . . . . 54
EDIT – Edit a Data Set . . . . . 54
Command Invocation Format . . . . . 56
Call Invocation Format . . . . . 57
Parameters . . . . . 57
Return Codes. . . . . 60
Example 1. . . . . 60
Example 2. . . . . 60
EDREC – Specify Edit Recovery Handling . . . . 61
Command Invocation Format . . . . . 61
Call Invocation Format . . . . . 62
Parameters . . . . . 62
Return Codes. . . . . 64
Example . . . . . 64
FILESTAT – Statistics for a file . . . . . 64
Command Invocation Format . . . . . 65
Call Invocation Format . . . . . 65
Parameters . . . . . 65
Return Codes. . . . . 65
Example . . . . . 65
FILEXFER – Upload or Download File . . . . . 65
Command Invocation Format . . . . . 66
Call Invocation Format . . . . . 66
Parameters . . . . . 66
Return Codes. . . . . 68
Example . . . . . 69
FTCLOSE – End File Tailoring . . . . . 69
Command Invocation Format . . . . . 69
Call Invocation Format . . . . . 69
Parameters . . . . . 70
Return Codes. . . . . 70
Example . . . . . 70
FTERASE – Erase File Tailoring Output . . . . . 70
Command Invocation Format . . . . . 71
Call Invocation Format . . . . . 71
Parameters . . . . . 71
Return Codes. . . . . 71
Example . . . . . 71
FTINCL – Include a Skeleton . . . . . 72
Command Invocation Format . . . . . 72
Call Invocation Format . . . . . 72
Parameters . . . . . 72
Return Codes. . . . . 72
Example . . . . . 72
FTOPEN – Begin File Tailoring . . . . . 73
Command Invocation Format . . . . . 73
Call Invocation Format . . . . . 73
Parameters . . . . . 73
Return Codes. . . . . 74
Example . . . . . 74
GETMSG – Get a Message . . . . . 74
Command Invocation Format . . . . . 74
Call Invocation Format . . . . . 75
Parameters . . . . . 75
Return Codes. . . . . 76
Example . . . . . 76
GRERROR – Graphics Error Block Service . . . . 76
Command Invocation Format . . . . . 76
Call Invocation Format . . . . . 76
Parameters . . . . . 77
Return Codes. . . . . 77
GRINIT – Graphics Initialization . . . . . 77
Command Invocation Format . . . . . 78
Call Invocation Format . . . . . 78
Parameters . . . . . 78
Return Codes. . . . . 78
Example . . . . . 79
GRTERM – Graphics Termination Service . . . . 79
Command Invocation Format . . . . . 79
Call Invocation Format . . . . . 79
Return Codes. . . . . 79
LIBDEF – Allocate Application Libraries. . . . . 79
Application Data Element Search Order . . . . 82
LIBDEF Display Utility . . . . . 83
User Link Libraries. . . . . 86
Message Libraries . . . . . 86
Command Invocation Format . . . . . 87
Call Invocation Format . . . . . 87
Parameters . . . . . 87
Return Codes. . . . . 88
Example 1 - The DATASET keyword . . . . . 89
Example 2 - The EXCLDATA keyword . . . . 90
Example 3 - The LIBRARY keyword . . . . . 90
Example 4 - The EXCLLIBR keyword. . . . . 91
Example 5. . . . . 91
Example 6 — STKADD . . . . . 92
LIST – Write Lines to the List Data Set . . . . . 93
Command Invocation Format . . . . . 93
Call Invocation Format . . . . . 93
Parameters . . . . . 94
Return Codes. . . . . 94
Formatting Data to be Written to the List Data Set . . . . . 95
List Data Set Characteristics Affect the LIST Service . . . . . 95
Controlling Line Spacing, Page Eject, and Highlighting . . . . . 95
How Carriage Control Characters Affect Truncation. . . . . 96
Example 1. . . . . 97
Example 2. . . . . 97
Example 3. . . . . 97
Example 4. . . . . 98
LMCLOSE – Close a Data Set . . . . . 98
Command Invocation Format . . . . . 98
Call Invocation Format . . . . . 98
Parameters . . . . . 99
Return Codes. . . . . 99
Example . . . . . 99

LMCOMP – Compresses a Partitioned Data Set . . 99
Command Invocation Format . . . . . 99
Call Invocation Format . . . . . 99
Parameters . . . . . 100
Return Codes . . . . . 100
Example . . . . . 100
LMCOPY – Copy Members of a Data Set . . . . 101
Command Invocation Format . . . . . 101
Call Invocation Format . . . . . 102
Parameters . . . . . 102
Return Codes . . . . . 103
Example . . . . . 104
LMDDISP – Data Set List Service. . . . . 104
Command Invocation Format . . . . . 105
Call Invocation Format . . . . . 105
Parameters . . . . . 105
Return Codes . . . . . 106
Example . . . . . 106
LMDFREE – Free a Data Set List ID . . . . . 106
Command Invocation Format . . . . . 106
Call Invocation Format . . . . . 106
Parameters . . . . . 107
Return Codes . . . . . 107
Example . . . . . 107
LMDINIT – Initialize a Data Set List. . . . . 107
Command Invocation Format . . . . . 108
Call Invocation Format . . . . . 108
Parameters . . . . . 108
Return Codes . . . . . 108
Example . . . . . 109
LMDLIST – List Data Sets . . . . . 109
Command Invocation Format . . . . . 110
Call Invocation Format . . . . . 110
Parameters . . . . . 110
Return Codes . . . . . 111
Example . . . . . 112
LMERASE – Erase a Data Set . . . . . 112
Command Invocation Format . . . . . 112
Call Invocation Format . . . . . 113
Parameters . . . . . 113
Return Codes . . . . . 114
Example . . . . . 114
LMFREE – Free Data Set from its Association with Data ID . . . . . 115
Command Invocation Format . . . . . 115
Call Invocation Format . . . . . 115
Parameters . . . . . 115
Return Codes . . . . . 115
Example . . . . . 116
LMGET – Read a Logical Record from a Data Set 116
Command Invocation Format . . . . . 116
Call Invocation Format . . . . . 117
Parameters . . . . . 117
Return Codes . . . . . 118
Example . . . . . 118
Example (MULTX). . . . . 118
LMINIT – Generate a Data ID for a Data Set . . . 119
Command Invocation Format . . . . . 120
Call Invocation Format . . . . . 120
Parameters . . . . . 120
Return Codes . . . . . 123
Examples. . . . . 123
LMMADD – Add a Member to a Data Set. . . . . 124
Command Invocation Format . . . . . 124
Call Invocation Format . . . . . 125
Parameters . . . . . 125
Return Codes . . . . . 126
Example . . . . . 127
LMMDEL – Delete Members from a Data Set. . . . 127
Command Invocation Format . . . . . 127
Call Invocation Format . . . . . 127
Parameters . . . . . 128
Return Codes . . . . . 128
Example . . . . . 128
LMMDISP – Member List Service . . . . . 129
Dialog Variables . . . . . 129
DISPLAY Option . . . . . 130
GET Option . . . . . 133
PUT Option . . . . . 135
ADD Option . . . . . 137
DELETE Option . . . . . 139
FREE Option . . . . . 140
LMMFIND – Find a Library Member . . . . . 141
Command Invocation Format . . . . . 141
Call Invocation Format . . . . . 142
Parameters . . . . . 142
Return Codes . . . . . 144
Example . . . . . 145
LMMLIST – List a Library's Members . . . . . 145
Command Invocation Format . . . . . 146
Call Invocation Format . . . . . 146
Parameters . . . . . 146
Return Codes . . . . . 148
Example . . . . . 148
LMMOVE – Move Members of a Data Set. . . . . 149
Command Invocation Format . . . . . 150
Call Invocation Format . . . . . 150
Parameters . . . . . 150
Return Codes . . . . . 151
Example . . . . . 152
LMMREN – Rename a Data Set Member . . . . . 152
Command Invocation Format . . . . . 152
Call Invocation Format . . . . . 153
Parameters . . . . . 153
Return Codes . . . . . 153
Example . . . . . 154
LMMREP – Replace a Member of a Data Set . . . 154
Command Invocation Format . . . . . 154
Call Invocation Format . . . . . 154
Parameters . . . . . 155
Return Codes . . . . . 156
Example . . . . . 156
LMMSTATS – Set and Store, or Delete ISPF Statistics . . . . . 156
Command Invocation Format . . . . . 157
Call Invocation Format . . . . . 157
Parameters . . . . . 158
Return Codes . . . . . 159
Example . . . . . 159
LMOPEN – Open a Data Set . . . . . 160
Command Invocation Format . . . . . 160
Call Invocation Format . . . . . 160

Parameters . . . . . 161
Return Codes . . . . . 161
Example . . . . . 162
LMPRINT – Print a Partitioned or Sequential Data Set . . . . . 162
Command Invocation Format . . . . . 162
Call Invocation Format . . . . . 162
Parameters . . . . . 163
Return Codes . . . . . 163
Example . . . . . 163
LMPUT – Write a Logical Record to a Data Set . . 164
Command Invocation Format . . . . . 164
Call Invocation Format . . . . . 165
Parameters . . . . . 165
Return Codes . . . . . 166
Example . . . . . 166
Example (MULTX) . . . . . 166
LMQUERY – Give a Dialog Information about a Data Set . . . . . 167
Command Invocation Format . . . . . 167
Call Invocation Format . . . . . 168
Parameters . . . . . 168
Return Codes . . . . . 170
Example . . . . . 170
LMRENAME – Rename an ISPF Library . . . . 171
Command Invocation Format . . . . . 171
Call Invocation Format . . . . . 171
Parameters . . . . . 171
Return Codes . . . . . 172
Example . . . . . 172
LOG – Write a Message to the Log Data Set . . . 173
Command Invocation Format . . . . . 173
Call Invocation Format . . . . . 173
Parameters . . . . . 173
Return Codes . . . . . 173
Example 1 . . . . . 174
Example 2 . . . . . 174
Example 3 . . . . . 174
MEMLIST – Member List Dialog Service . . . . 174
Command Invocation Format . . . . . 175
Call Invocation Format . . . . . 175
Parameters . . . . . 175
Return Codes . . . . . 176
Example . . . . . 176
PQUERY – Obtain Panel Information . . . . . 176
Command Invocation Format . . . . . 176
Call Invocation Format . . . . . 177
Parameters . . . . . 177
Return Codes . . . . . 178
Example . . . . . 178
QBASELIB – Query Base Library Information . . 178
Command Invocation Format . . . . . 178
Call Invocation Format . . . . . 178
Parameters . . . . . 179
Return Codes . . . . . 179
Example . . . . . 179
QLIBDEF – Query LIBDEF Definition Information 180
Command Invocation Format . . . . . 180
Call Invocation Format . . . . . 180
Parameters . . . . . 180
Return Codes . . . . . 181
Example . . . . . 181
| QTABOPEN – Query Open ISPF Tables . . . . 181
| Command Invocation Format . . . . . 181
| Call Invocation Format . . . . . 181
| Parameters . . . . . 182
| Return codes . . . . . 182
| Example . . . . . 182
QUERYENQ – Query System ENQ Data . . . . 182
Command Invocation Format . . . . . 182
Call Invocation Format . . . . . 182
Parameters . . . . . 182
Variables Returned in Each Row of the Table 183
Return codes . . . . . 184
REMPOP – Remove a Pop-Up Window . . . . 184
Command Invocation Format . . . . . 184
Call Invocation Format . . . . . 184
Parameters . . . . . 185
Return codes . . . . . 185
SELECT – Select a Panel or Function . . . . . 185
Command Invocation Format . . . . . 185
Call Invocation Format . . . . . 186
Parameters . . . . . 186
Return Codes . . . . . 192
Example 1 . . . . . 192
Example 2 . . . . . 192
Example 3 . . . . . 193
Example 4 . . . . . 193
Example 5 . . . . . 193
Example 6 . . . . . 193
SETMSG – Set Next Message . . . . . 193
Command Invocation Format . . . . . 194
Call Invocation Format . . . . . 194
Parameters . . . . . 194
Return Codes . . . . . 195
Example 1 . . . . . 195
Example 2 . . . . . 195
Return Codes . . . . . 196
TBADD – Add a Row to a Table . . . . . 196
Command Invocation Format . . . . . 197
Call Invocation Format . . . . . 197
Parameters . . . . . 197
Return Codes . . . . . 198
Example 1 . . . . . 198
Example 2 . . . . . 199
TBBOTTOM – Set the Row Pointer to Bottom . . 199
Command Invocation Format . . . . . 199
Call Invocation Format . . . . . 199
Parameters . . . . . 199
Return Codes . . . . . 200
Example . . . . . 200
TBCLOSE – Close and Save a Table . . . . . 201
Command Invocation Format . . . . . 201
Call Invocation Format . . . . . 201
Parameters . . . . . 202
Return Codes . . . . . 202
Example . . . . . 203
TBCREATE – Create a New Table . . . . . 203
Command Invocation Format . . . . . 203
Call Invocation Format . . . . . 203
Parameters . . . . . 204
Return Codes . . . . . 205

Example 1 . . . . . 205
Example 2 . . . . . 205
Example 3 . . . . . 205
TBDELETE – Delete a Row from a Table . . . . 206
Command Invocation Format . . . . . 206
Call Invocation Format . . . . . 206
Parameters . . . . . 206
Return Codes . . . . . 206
Example . . . . . 206
TBDISPL – Display Table Information . . . . . 207
TBDISPL Operation . . . . . 208
Operational Results From User Actions . . . . 208
Command Invocation Format . . . . . 209
Call Invocation Format . . . . . 210
Parameters . . . . . 210
Parameter Processing . . . . . 212
Return Codes . . . . . 213
Example . . . . . 214
System Variables Related to TBDISPL . . . . 214
Panel Control Variables Related to TBDISPL . . 216
Parameter Variables Related to TBDISPL . . . 216
Using TBDISPL with Other Services . . . . . 216
Techniques for Using the TBDISPL Service . . 218
Rules Applying to Variable Model Lines . . . 220
An Example of Using the TBDISPL and TBPUT Services . . . . . 223
Command Procedure Function . . . . . 224
Description of Function Steps . . . . . 225
TBDISPL Summary . . . . . 229
TBEND – Close a Table without Saving . . . . 231
Command Invocation Format . . . . . 231
Call Invocation Format . . . . . 232
Parameters . . . . . 232
Return Codes . . . . . 232
Example . . . . . 232
TBERASE – Erase a Table . . . . . 232
Command Invocation Format . . . . . 232
Call Invocation Format . . . . . 233
Parameters . . . . . 233
Return Codes . . . . . 233
Example . . . . . 233
TBEXIST – Determine Whether a Row Exists in a Table . . . . . 233
Command Invocation Format . . . . . 234
Call Invocation Format . . . . . 234
Parameters . . . . . 234
Return Codes . . . . . 234
Example . . . . . 234
TBGET – Retrieve a Row from a Table . . . . . 235
Command Invocation Format . . . . . 235
Call Invocation Format . . . . . 235
Parameters . . . . . 235
Return Codes . . . . . 236
Example . . . . . 236
TBMOD – Modify a Row in a Table . . . . . 237
Command Invocation Format . . . . . 237
Call Invocation Format . . . . . 237
Parameters . . . . . 237
Return Codes . . . . . 238
Example . . . . . 238
TBOPEN – Open a Table . . . . . 238
Command Invocation Format . . . . . 239
Call Invocation Format . . . . . 239
Parameters . . . . . 239
Return Codes . . . . . 240
Example . . . . . 240
TBPUT – Update a Row in a Table . . . . . 240
Command Invocation Format . . . . . 240
Call Invocation Format . . . . . 241
Parameters . . . . . 241
Return Codes . . . . . 241
Example . . . . . 241
TBQUERY – Obtain Table Information . . . . . 242
Command Invocation Format . . . . . 242
Call Invocation Format . . . . . 242
Parameters . . . . . 243
Return Codes . . . . . 244
Example . . . . . 244
TBSARG – Define a Search Argument . . . . . 244
Command Invocation Format . . . . . 246
Call Invocation Format . . . . . 246
Parameters . . . . . 246
Return Codes . . . . . 247
Examples . . . . . 247
TBSAVE – Save a Table . . . . . 248
Command Invocation Format . . . . . 248
Call Invocation Format . . . . . 249
Parameters . . . . . 249
Return Codes . . . . . 250
Example . . . . . 250
TBSCAN – Search a Table . . . . . 250
Command Invocation Format . . . . . 251
Call Invocation Format . . . . . 251
Parameters . . . . . 252
Return Codes . . . . . 253
Example 1 . . . . . 254
Example 2 . . . . . 254
Example 3 . . . . . 254
TBSKIP – Move the Row Pointer . . . . . 255
Command Invocation Format . . . . . 255
Call Invocation Format . . . . . 255
Parameters . . . . . 255
Return Codes . . . . . 256
Example . . . . . 256
TBSORT – Sort a Table . . . . . 257
Command Invocation Format . . . . . 257
Call Invocation Format . . . . . 257
Parameters . . . . . 257
Return Codes . . . . . 259
Example 1 . . . . . 259
Example 2 . . . . . 259
TBSTATS – Retrieve Table Statistics . . . . . 260
Command Invocation Format . . . . . 260
Call Invocation Format . . . . . 261
Parameters . . . . . 261
Return Codes . . . . . 263
Example . . . . . 263
TBTOP – Set the Row Pointer to the Top . . . . 264
Command Invocation Format . . . . . 264
Call Invocation Format . . . . . 264
Parameters . . . . . 264
Return Codes . . . . . 264

Example . . . . . 264
TBVCLEAR – Clear Table Variables . . . . . 265
Command Invocation Format . . . . . 265
Call Invocation Format . . . . . 265
Parameters . . . . . 265
Return Codes . . . . . 265
Example . . . . . 265
TRANS – Translate CCSID Data . . . . . 266
Command Invocation Format . . . . . 266
Call Invocation Format . . . . . 266
Parameters . . . . . 266
Return codes . . . . . 267
VCOPY – Create a Copy of a Variable . . . . . 267
Command Invocation Format . . . . . 267
Call Invocation Format . . . . . 267
Parameters . . . . . 268
Return Codes . . . . . 268
Example . . . . . 268
VDEFINE – Define Function Variables . . . . . 268
Command Invocation Format . . . . . 269
Call Invocation Format . . . . . 269
Parameters . . . . . 270
Return Codes . . . . . 274
Example 1 . . . . . 274
Example 2 . . . . . 274
Example 3 . . . . . 274
Example 4 . . . . . 275
VDEFINE Exit Routine . . . . . 275
VDELETE – Remove a Definition of Function Variables . . . . . 278
Command Invocation Format . . . . . 279
Call Invocation Format . . . . . 279
Parameters . . . . . 279
Return Codes . . . . . 279
Example . . . . . 279
VERASE – Remove Variables from Shared or Profile Pool . . . . . 279
Command Invocation Format . . . . . 279
Call Invocation Format . . . . . 279
Parameters . . . . . 280
Return Codes . . . . . 280
Example . . . . . 280
VGET – Retrieve Variables from a Pool or Profile 281
Command Invocation Format . . . . . 281
Call Invocation Format . . . . . 281
Parameters . . . . . 281
Return Codes . . . . . 281
Example . . . . . 282
VIEW – View a Data Set. . . . . 282
Command Invocation Format . . . . . 283
Call Invocation Format . . . . . 284
Parameters . . . . . 285
Return Codes . . . . . 287
Example 1 . . . . . 287
Example 2 . . . . . 288
Command Invocation . . . . . 288
VIIF – View Interface . . . . . 288
Command Invocation Format . . . . . 289
Call Invocation Format . . . . . 289
Parameters . . . . . 290
Dialog-Supplied Routines . . . . . 291
Read Routine . . . . . 292
Command Routine . . . . . 292
Return Codes . . . . . 293
Read Routine . . . . . 293
Command Routine Return Codes. . . . . 293
VIIF Service Return Codes . . . . . 293
Example . . . . . 293
VMASK – Mask and Edit Processing . . . . . 294
VMASK Call Invocation . . . . . 294
Parameters . . . . . 294
Return Codes . . . . . 297
Example . . . . . 297
The VEDIT Statement . . . . . 297
VPUT – Update Variables in the Shared or Profile Pool . . . . . 297
Command Invocation Format . . . . . 297
Call Invocation Format . . . . . 297
Parameters . . . . . 297
Return Codes . . . . . 298
Example . . . . . 298
VREPLACE – Replace a Variable . . . . . 298
Command Invocation Format . . . . . 299
Call Invocation Format . . . . . 299
Parameters . . . . . 299
Return Codes . . . . . 299
Example . . . . . 299
VRESET – Reset Function Variables . . . . . 299
Command Invocation Format . . . . . 299
Call Invocation Format . . . . . 300
Return Codes . . . . . 300
Example . . . . . 300
WSCON — Connect to a Workstation . . . . . 300
Command Invocation Format . . . . . 300
Call Invocation Format . . . . . 300
Parameters . . . . . 301
Return Codes . . . . . 302
Example . . . . . 302
WSDISCON — Disconnect from a Workstation . . 303
Command Invocation Format . . . . . 303
Call Invocation Format . . . . . 303
Parameters . . . . . 303
Return Codes . . . . . 303
Usage Notes. . . . . 304

**Notices . . . . . 305**
Programming Interface Information . . . . . 306
Trademarks . . . . . 307

**Index . . . . . 309**

# Figures


1. Panel with an Action Bar Pull-Down Menu xxi
2. Pop-Up Selected from an Action Bar Pull-Down . . . . . xxii
3. Panel with an Action Bar and Point-and-Shoot Fields . . . . . xxii
4. An Unavailable Choice on a Pull-Down xxiv
5. Multiple Pop-up Windows . . . . . 23
6. ISPLIBD - all LIBDEF definitions . . . . . 84
7. ISPLIBD ISPPLIB - ISPPLIB LIBDEF definition 85
8. ISPLIBD ISPPLIB - ISPPLIB LIBDEF stacked definition . . . . . 85
9. Variable Model Lines – Panel Definition 221
10. Variable Model Lines – Display 1 . . . . . 222
11. (Part 1 of 2). Variable Model Lines – Display 1. . . . . 222
12. (Part 2 of 2). Variable Model Lines – Display 2. . . . . 223
13. Five Rows in Table TAB1 . . . . . 225
14. Table TAB1 as Displayed Using Panel PAN1 225
15. Table Display Panel Definition PAN1 226


© Copyright IBM Corp. 1980, 2004

# Preface

This document describes how to use ISPF dialog management component (DM) services and Program Development Facility component (PDF) services. Programmers who develop applications with ISPF can use the services described in this publication to develop dialogs from programs or command procedures.

## Who Should Use This Document?

This document is for application programmers who develop dialogs using ISPF. Users should be familiar with coding in CLIST, REXX, or any of the other programming or command procedure languages supported by ISPF in the MVS environment.

## What Is In This Document?

This document contains two chapters.

Chapter 1, "Introduction to ISPF Services," on page 1 describes how to invoke ISPF services, provides an explanation of various service return codes, and lists and summarizes all of the services described in this document.

Chapter 2, "Description of the ISPF Services," on page 21 contains the following information about each of the ISPF services:

- A description of the function and operation of the service. This description also refers to other services that can be used with this service.
- The syntax used to code the service, showing both the command procedure format and the call format.
- A description of any required or optional keywords or parameters.
- A description of the error codes returned by the service.
- Examples of the how the service is used to develop dialogs.

The services are listed in alphabetical order.

## Using LookAt to look up message explanations

LookAt is an online facility that lets you look up explanations for most of the IBM® messages you encounter, as well as for some system abends and codes. Using LookAt to find information is faster than a conventional search because in most cases LookAt goes directly to the message explanation.

You can use LookAt from the following locations to find IBM message explanations for z/OS® elements and features, z/VM®, VSE/ESA™, and Clusters for AIX® and Linux:

- The Internet. You can access IBM message explanations directly from the LookAt Web site at http://www.ibm.com/eserver/zseries/zos/bkserv/lookat/.
- Your z/OS TSO/E host system. You can install code on your z/OS or z/OS.e systems to access IBM message explanations, using LookAt from a TSO/E command line (for example, TSO/E prompt, ISPF, or z/OS UNIX® System Services running OMVS).

© Copyright IBM Corp. 1980, 2004

- Your Microsoft® Windows® workstation. You can install code to access IBM message explanations on the *z/OS Collection* (SK3T-4269), using LookAt from a Microsoft Windows command prompt (also known as the DOS command line).
- Your wireless handheld device. You can use the LookAt Mobile Edition with a handheld device that has wireless access and an Internet browser (for example, Internet Explorer for Pocket PCs, Blazer, or Eudora for Palm OS, or Opera for Linux handheld devices). Link to the LookAt Mobile Edition from the LookAt Web site.

You can obtain code to install LookAt on your host system or Microsoft Windows workstation from a disk on your *z/OS Collection* (SK3T-4269), or from the LookAt Web site (click **Download**, and select the platform, release, collection, and location that suit your needs). More information is available in the LOOKAT.ME files available during the download process.

# Summary of Changes

z/OS V1R6.0 ISPF contains the following changes and enhancements:
- ISPF Product and Library changes
- ISPF Dialog Manager component changes (including DTL changes)
- ISPF PDF Component changes
- ISPF SCLM Component changes
- ISPF Client/Server Component changes

## ISPF Product and Library 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 the version number and release. Note that the *x.y* value is not necessarily the same as the operating system version. For example, a value of ″ISPF 5.6″ represents ISPF for z/OS Version 1 Release 6.0.

The ZOS390RL variable contains the level of the z/OS release running on your system.

The ZISPFOS system variable contains the level of ISPF that is running as part of the operating system release on your system. This might or might not match ZOS390RL. For this release of ISPF, the variable contains **ISPF for z/OS 01.06.00**.

## ISPF Dialog Manager component changes

The DM component of ISPF includes the following new functions and enhancements:

- The new *REXX panel definition statement allows the inclusion of Rexx code within a panel's )INIT, )REINIT, or )PROC sections to perform arithmetic, verification, transformation, translation, and formatting of dialog variables.
- Enhancements to ISPF File Tailoring:
  - Added support for continuation of control statements.
  - Increased maximum imbed levels from 3 to 15.
  - Increased maximum select levels from 8 to 32.
  - Increased the maximum number of parameters on a control statement from 31 to 63.
  - Eliminated exclusive SPFEDIT enqueue on a skeleton member when processing a skeleton.
  - Improved skeleton processing by reading skeleton records into storage.
  - Added support for the )DO ... )ENDDO control statements.
  - Added support for the )IF ... THEN ... )ELSE control statements.
  - Added )ITERATE and )LEAVE control words for use within )DO ... )ENDDO loops.
  - Added the )NOP control word, which can be used with a null )IF or )ELSE statement.
  - Added TBSCAN support to the )DOT control word.
- The restriction that Language Environment-enabled programming languages can not be used for ISPF exits has been partially removed. Exits can now be written

© Copyright IBM Corp. 1980, 2004

in languages that use the LE run-time environment, as long as all the dialogs and service routines are LE-conforming. A mixture of LE-conforming main dialog code and service routine code is not supported.

ISPF Configuration Utility changes:

- Added support for zero block size for dynamic allocation of the ISPLIST, ISPLOG, ISPCTL*x*, ISPLST*x*, and ISPWRK*x* data sets.
- Added support for specifying primary and secondary space for the ISPCTL0 and ISPLST*x* data sets.
- New keywords control what happens when an explicit member list request is made for an empty PDS/E:

```
DISPLAY_EMPTY_MEMBER_LIST
DISPLAY_EMPTY_MEMBER_LIST_PATTERN
DISPLAY_EMPTY_MEMBER_LIST_FUNCTION
RESET_EMPTY_MEMBER_LIST_OPTIONS
```

### Dialog Tag Language (DTL) changes

There are no changes to Dialog Tag Language (DTL) for this release.

## ISPF PDF Component changes

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

- The TBQUERY service has been enhanced to return information about TBSORT and the last TBSARG issued against a table.
- The new QTABOPEN service allows an ISPF dialog to obtain a list of currently open ISPF tables. The TBSTATS or TBQUERY service can then be used to obtain more detailed information about each table.
- A new Edit primary command, COLS, displays a non-scrolling columns line in Edit or View.
- A new Edit primary command and edit macro command, HIDE, removes the excluded lines messages from the display. RESET HIDE restores the excluded lines messages.
- The CUT Edit primary command and edit macro commands now support two new flags: X specifies to cut only excluded lines to the clipboard. NX specifies to cut only nonexcluded lines.
- The MODEL edit macro command now issues RC=4 if the model exceeds the right hand margin of the data being edited.
- The VOLUME edit macro command is changed to permit return of the volume serial where the original data set is stored.
- The CURSOR edit assignment statement has been changed as follows: if the cursor is in the command area, the cursor value is column 0 of the first data line on the panel.
- When saving a member list to a dataset using the SAVE primary command, the new keyword LONG formats all dates in yyyy/mm/dd format for the member.
- Documentation has been added for the OVOLUME parameter on the LMQUERY service.

## ISPF SCLM Component changes

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

- For each of the FLMCMD services a new panel has been added, providing the ability to select a service from a menu and then enter the service parameters in an ISPF interface panel. These panels can also be called directly, by entering the FLMCMD service command without parameters.
- The new Unit of Work utility allows you to use an ARCHDEF member as a member list from which you can use the standard SCLM functions such as select, edit, build, view build map, and promote. You can add your own customized line commands to work with the Unit of Work member List.
- The new SCLM Explorer utility provides the ability to select any architecture definition or part member and then navigate up or down the hierarchy of related ARCHDEFs or parts.

## 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 documents call this ″running in GUI mode.″

There are no changes to the ISPF Client/Server for this release.

## ISPF Migration Considerations

If you are migrating to z/OS V1R6.0 from an earlier release of z/OS or from OS/390 V2R10.0, no migration actions are needed. If you are migrating to z/OS V1R6.0 from an earlier release of OS/390, follow the migration actions for OS/390 V2R10.0.

When migrating from one version of ISPF to another, you must reassemble and relink the SCLM project definition.

**Note**

This book contains terminology, maintenance, and editorial changes. Technical changes or additions to the text and illustrations are indicated by a vertical line to the left of the change.

Starting with z/OS V1R2, you may notice changes in the style and structure of some content in this book—for example, headings that use uppercase for the first letter of initial words only, and procedures that have a different look and format. The changes are ongoing improvements to the consistency and retrievability of information in our books.

# What's in the z/OS V1R6.0 ISPF library?

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

## z/OS V1R6.0 ISPF

| Title | Order Number |
|---|---|
| *z/OS V1R6.0 ISPF Dialog Tag Language Guide and Reference* | SC34-4824–03 |
| *z/OS V1R6.0 ISPF Planning and Customizing* | GC34-4814–03 |
| *z/OS V1R6.0 ISPF User's Guide Volume I* | SC34-4822–03 |
| *z/OS V1R6.0 ISPF User's Guide Volume II* | SC34-4823–03 |
| *z/OS V1R6.0 ISPF Services Guide* | SC34-4819–03 |
| *z/OS V1R6.0 ISPF Dialog Developer's Guide and Reference* | SC34-4821–03 |
| *z/OS V1R6.0 ISPF Reference Summary* | SC34-4816–03 |
| *z/OS V1R6.0 ISPF Edit and Edit Macros* | SC34-4820–03 |
| *z/OS V1R6.0 ISPF Messages and Codes* | SC34-4815–03 |
| *z/OS V1R6.0 ISPF Software Configuration and Library Manager Project Manager's and Developer's Guide* | SC34-4817–03 |
| *z/OS V1R6.0 ISPF Software Configuration and Library Manager Reference* | SC34-4818–03 |

© Copyright IBM Corp. 1980, 2004

# 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.

These action bars give you another way to move around in ISPF, as well as the ability to nest commands. 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 document:

**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*.

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

**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.

**modal pop-up window**
A type of window that requires you to interact with the panel in the pop-up before continuing. This includes canceling 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 xxiv for more information.

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

**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 xxi 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

© Copyright IBM Corp. 1980, 2004

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.

**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 ISPF in GUI mode).

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

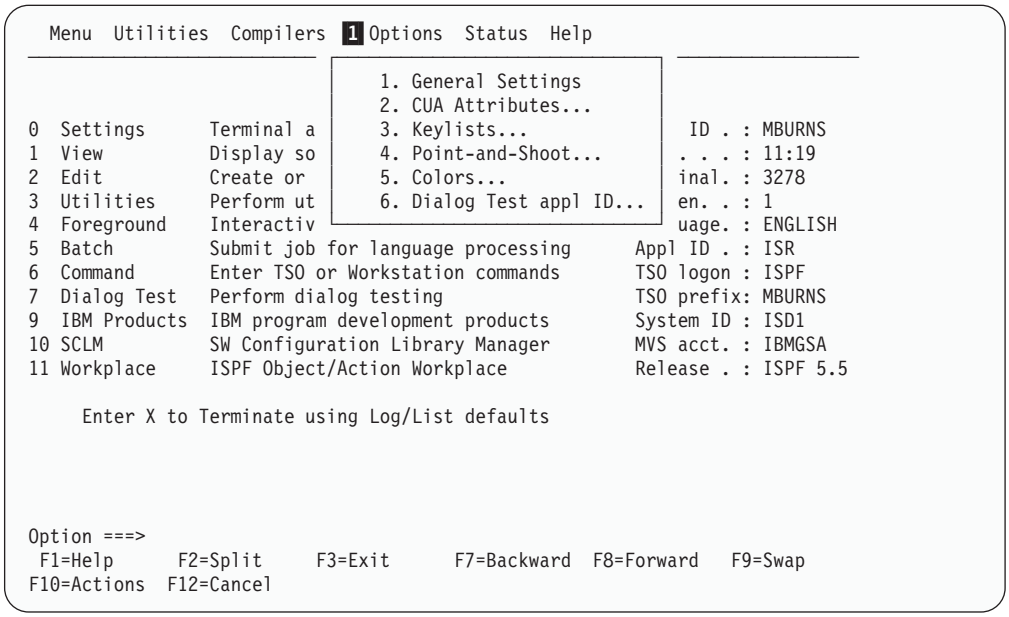
  – In 3270 mode, on the command line, type ACTIONS and the mnemonic letter that corresponds to an underscored letter in the action bar choice text. This results in the display of the pull-down menu for that action bar choice.
  – In 3270 mode, on the command line enter the mnemonic letter that corresponds 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 document 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.
2. Some users program their mouse emulators as follows:
   - Mouse button 1 – position the cursor to the pointer and simulate Enter
   - Mouse button 2 – simulate F12 (Cancel).
3. 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.
4. 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 Options on the ISPF Primary Option Menu action bar.




**1** The selected action bar choice is highlighted.

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

To select a choice from the Options 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 6, ISPF displays the Dialog Test Application ID pop-up, as shown in Figure 2 on page xxii.

**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
 ─┌──────────────────────────────────────┐ ──────────────────────────────────────
  │    Dialog Test Application ID        │ ption Menu
  │                                      │
0 │ Change the application ID for        │ ters            User ID . : MBURNS
1 │ Dialog Test.                         │ istings         Time. . . : 11:19
2 │                                      │ data            Terminal. : 3278
3 │     Application ID . . ISR           │ s               Screen. . : 1
4 │                                      │ cessing         Language. : ENGLISH
5 │                                      │ processing      Appl ID . : ISR
6 │ Command ===>                         │  commands       TSO logon : ISPF
7 │  F1=Help    F2=Split   F3=Exit       │                 TSO prefix: MBURNS
9 │  F9=Swap   F12=Cancel                │ products        System ID : ISD1
1 └──────────────────────────────────────┘  Manager        MVS acct. : IBMGSA
11 Workplace     ISPF Object/Action Workplace            Release . : ISPF 5.5

     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
```

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

```
 1  Menu  Utilities  Compilers  Options  Status  Help
──────────────────────────────────────────────────────────────────────────────
                          ISPF Primary Option Menu
   2                                                      3
0  Settings      Terminal and user parameters            User ID . : MBURNS
1  View          Display source data or listings         Time. . . : 12:29
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 : ISPF
7  Dialog Test   Perform dialog testing                  TSO prefix: MBURNS
9  IBM Products  IBM program development products        System ID : ISD1
10 SCLM          SW Configuration Library Manager        MVS acct. : IBMGSA
11 Workplace     ISPF Object/Action Workplace            Release . : ISPF 5.5

     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*

## Command Nesting

You can use the action bars to *suspend* an activity while you perform a new one.

For example, if you are editing a data set and want to allocate another data set, 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 ended the function, ISPF returns you directly to your edit session.

By contrast, if you used the jump function (=3.2), ISPF would end your edit session before displaying the Data Set Utility.

## 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 xxiv.

```
  Menu  RefList  RefMode  Utilities  Workstation  Help
 ______________ ┌──────────────────┐ ──────────────────────────────────────
                │ 1  1. List Execute │ ry Panel
                │    *. List Retrieve │                              More:     +
ISPF Library:   └──────────────────┘
   Project . . . PDFTDEV
   Group . . . . STG      . . .          . . .          . . .
   Type  . . . . GML
   Member  . . .                 (Blank or pattern for member selection list)

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

Workstation File:
   File Name . . . . .
                                        Options
Initial Macro  . . . .                  /  Confirm Cancel/Move/Replace
Profile Name . . . . .                     Browse Mode
Format Name  . . . . .                     View on Workstation
Data Set Password  . .                  /  Warn on First Data Change
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 xxii, 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

z/OS V1R6.0 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 nonblank 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:** 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, the check box is filled 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.

# How to Navigate in ISPF without Using Action Bars

If you use a non-programmable terminal to access z/OS V1R6.0 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.

# Chapter 1. Introduction to ISPF Services

ISPF services help you develop interactive ISPF applications called *dialogs*. These services can make your job easier because they perform many tedious and repetitious operations. In addition, the ISPF services allow you to start a dialog in batch mode and let it run in the background while you work with another application in the foreground.

PDF component services communicate with the dialog through dialog variables. Thus, you can use PDF component services with DM component services. For information about DM component services and writing dialogs, refer to the *ISPF Dialog Developer's Guide and Reference*.

You can also use PDF component services within edit macros, or you can use edit macros through the EDIT service. For information about writing edit macros, refer to *ISPF Edit and Edit Macros*.

## Description of the Services

The services are described in alphabetical order and each service description consists of the following information:

**Description** A description of the function and operation of the service. This description also refers to other services that can be used with this service.

**Format** The syntax used to code the service, showing commands and calls.

**Parameters** A description of any required or optional keywords or parameters.

**Return Codes** A description of the codes returned by the service. For all services, a return code of 12 or higher implies a severe error. This error is usually a syntax error, but can be any severe error detected when using the services.

**Examples** Sample usage of the services.

For each service, the command procedure or command invocation format is shown, followed by the call or call invocation format.

The command formats are provided as CLIST or REXX command procedures, using ISPEXEC.

Call formats are shown in PL/I syntax, although you are not limited to PL/I calls. For example, ";" ends statements in the formats described. This is a PL/I convention, but you should use the syntax appropriate for your programming language.

Consider using the Edit model facilities when you code requests for ISPF services. This will save keying the parts of dialog elements that are constant regardless of the function in which they are used. See *ISPF Edit and Edit Macros* for a description of these facilities.

© Copyright IBM Corp. 1980, 2004

### Notation Conventions

This document uses the following notation conventions to describe the format of the ISPF services:

- Uppercase commands and their uppercase parameters to show required entry.
- Lowercase parameters to show variables (substitute your values for them).
- 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.
- Underscores to show defaults.

## Using ISPQRY to Test Whether ISPF Is Active

A program can determine if ISPF services are currently available to it through use of ISPQRY. To test the availability of ISPF, the function issues:

```
CALL ISPQRY;
```

There are no parameters associated with the call to ISPQRY. No messages are written to the terminal. The response from ISPQRY is one of the following return codes:

**0** The services are available to the caller.

**20** The services are *not* available to the caller.

## Invoking the ISPF Services

Dialog developers use a command or a call statement to invoke ISPF services from functions at the point where the service is needed.

Functions coded in a command procedure language invoke ISPF services by means of the ISPEXEC command. For example:

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

This example invokes a service to display information on a terminal. A panel definition named XYZ, prepared by the developer and pre-stored in a panel file, specifies both the content and the format of the display.

Functions coded in APL2® invoke ISPF services by using ISPEXEC in an APL2 function. For example:

```
RC ← ISPEXEC 'DISPLAY PANEL(XYZ)'
```

This example invokes the display service to display information on a terminal by using panel definition XYZ from the ISPF panel file to control the content and format of the display.

Functions coded in a programming language other than FORTRAN, Pascal, or APL2 invoke ISPF services by calling either ISPLINK or ISPEXEC. For example, in PL/I:

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

or alternatively, set BUFLEN to 18, then:

```
CALL ISPEXEC (BUFLEN, 'DISPLAY PANEL(XYZ)');
```

This example invokes a service to display panel XYZ. FORTRAN and Pascal use only 6 characters, such as ISPLNK or ISPEX, in a called module's name.

Thus, the FORTRAN or Pascal call is in the following format:

```
lastrc = ISPLNK ('DISPLAY ', 'XYZ ')
```

or alternatively:

```
lastrc = ISPEX  (18, 'DISPLAY PANEL(XYZ)')
```

ISPLINK and ISPEXEC can be called from programs coded in any language that uses standard OS register conventions for call interfaces and the standard convention for signaling the end of a variable length parameter list. Assembler programs must include code to implement the standard save area convention.

## Load Module Search Order

When you are using STEPLIB to test new maintenance, releases, or versions of ISPF, and an ISPLLIB is allocated, those data sets allocated to STEPLIB that contain ISPF load modules except for the SISPSASC data set, should also be allocated to ISPLLIB. This prevents the possibility of mixed code (production code versus code to be tested). For more information, refer to the *ISPF Dialog Developer's Guide and Reference*.

If you are using the ISPF client/server (ISPF C/S) feature, the SISPSASC data set must be in STEPLIB or LNKLST. The modules in this data set are not searched for in ISPLLIB. For more information about SISPSASC refer to the *ISPF Planning and Customizing*.

## Invoking Services from Command Procedures

To invoke ISPF services for a command invocation, use one of the following:

- The ISPEXEC command in a command invocation written in CLIST or REXX
- Option 7.6 of ISPF, the Dialog Services option of the Dialog Test facility.

The following services are *not* available using the ISPEXEC call from a command procedure:

```
GRERROR   VCOPY     VMASK
GRINIT    VDEFINE   VREPLACE
GRTERM    VDELETE   VRESET
```

These services are available by using the CALL from programs.

### The ISPEXEC Interface

The general format for a command invocation is:

```
ISPEXEC service-name parameter1  parameter2  parameter3 ...
```

The command invocation statement must be specified in uppercase.

### ISPEXEC Parameter Conventions

**service-name**
Alphabetic; up to 8 characters long.

**parameter1**
Positional parameter; required for some services.

**parameter2 parameter3 ...**
Keyword parameters. They can take either of two forms:

```
keyword
keyword (value)
```

Some keyword parameters are required and others are optional, depending on the service. Optional parameters are enclosed in brackets ([ ]). You can code keyword parameters in any order, but if you code duplicate or conflicting keywords, ISPF uses the last instance of the keyword.

## Using Command Invocation Variables

You can use a CLIST or REXX variable, in the form of a name preceded by an ampersand (&), as the service name or as a parameter anywhere within a statement. Each variable is replaced by its current value before execution of the ISPEXEC command. Refer to *TSO/E Version 2 CLISTs*, *TSO/E REXX/MVS User's Guide*, and *TSO/E REXX/MVS Reference* for further information.

## Attention Interrupt Handling

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 interrupt exits in the *CLIST Implementation and Reference* and the *TSO Terminal User's Guide*.

## Passing Dialog Variables as Parameters

Some ISPF services allow the names of dialog variables to be passed as parameters. The ISPEXEC interface scans these variables for their values in the ISPF function, shared, and profile variable pools. Variable names are 8 characters or fewer, with the exception of FORTRAN and Pascal variable names, which are limited to 6 or fewer characters. These names should not be preceded by an ampersand unless substitution is desired. For example:

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

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

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

```
ISPEXEC  VGET  name-list  [
ASIS|SHARED|PROFILE]
```

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

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

The last two lines of the example, with and without the parentheses, are equivalent.

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

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

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

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

## Invoking ISPF Services with Program Functions

Programs call ISPF services by invoking an ISPF subroutine interface. The two basic call interfaces are ISPEXEC and ISPLINK. However, FORTRAN and Pascal use the alternate name forms ISPEX and ISPLNK, because these languages limit a module name to 6 characters. A program cannot use an ISPLINK call to invoke APL2.

Call statements in this document are shown in PL/I syntax. Service names and keyword values are shown as literals, enclosed in single quotes (’); for example:

```
CALL  ISPLINK ('TBOPEN  ', 'XTABLE ', 'NOWRITE ');
```

or, alternatively:

```
 ...set BUFLEN equal to 21...
CALL  ISPEXEC (BUFLEN, 'TBOPEN XTABLE NOWRITE');
```

Some languages, such as COBOL, do not allow literals within a call statement. Use of literals is never required for any language. All parameters can be specified as variables.

### The ISPLINK Interface

For calls in PL/I, the general call format for invoking ISPF services from functions by using ISPLINK is:

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

### CALL ISPLINK Parameters

These parameters are positional. They must appear in the order described for each service.

Parameters shown inside brackets ([ ]) are optional, but ISPF assumes default values for those parameters you do not choose.

If you want to omit a parameter and use its default value, you must account for it by inserting a blank enclosed in single quotes (' ') in its place. This is how you would omit parm2 from this example call:

```
CALL ISPLINK (service-name, parm1, ' ', parm3);
```

If you need only the first few of a list of parameters, you can omit all other parameters to the right of the last parameter you need, provided that you are certain that none of the remaining parameters are necessary for your invocation. For example, if you are using a service that has five parameters, but you need to use only the first three, code it like this:

```
CALL ISPLINK (service-name, parm1, parm2, parm3);
```

You must show the last parameter in the calling sequence with a ‘1’ as the high order bit in the last entry of the address list. PL/I, COBOL, Pascal, and FORTRAN call statements automatically generate this high-order bit. Standard register conventions are used. Registers 2 to 14 are preserved across the call. However, you must use the VL keyword in Assembler call statements.

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

**service-name or keyword**
A left-justified character string that you code exactly as shown in the service-name description. The description of the particular service shows the service-name or keyword character string, each of which can be up to 8 characters long. All service names and keywords must be padded with blanks to their maximum length of 8 characters.

**single name**
A left-justified character string. If the string is less than the maximum length for the particular parameter, it must have a trailing blank to delimit the end of the string. The minimum length for a single name is 1 character. The maximum length for most names is 8 characters; exceptions include the data set name and volume serial.

**numeric value**
A fullword fixed binary number.

**numeric name**
A dialog variable in which a number is stored. If these variables are defined in a program module, they can be either fullword fixed binary variables or character string variables. If the values are returned as characters, they are right-justified with leading zeros.

**name-list (string format)**
A list of dialog variable names coded as a character string. Each name is from 1 to 8 characters in length. The string must be enclosed in parentheses. Within the parentheses, you can separate the names with either commas or blanks. For example:

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

When the list consists of a single name, you do not need parentheses. You must include a trailing blank if parentheses are *not* used and if the name is fewer than 8 characters long. A maximum of 254 names can be listed in the string format.

**name-list (structure format)**
A list of dialog variable names passed in a structure. Each name is from 1 to 8 characters long. The structure must contain the following information in the given order:

1. **Count**
   A fullword fixed binary integer containing the number of names in the list.
2. **Reserved**
   A fullword fixed binary integer that must contain a value of either 0 or 8.
3. **List of names**
   Each part of the list must be an 8-byte character string. Within each part, the name of the variable must be left-justified and must have trailing blanks. The maximum number of names in the list is 254.

**Note:** In general, either form of the name-list (the string format or the structure format) is acceptable where a name-list is referred to in the syntax. However, the ISPEXEC command syntax requires the string format for name-list.

**subfield with keyword**
A left-justified character string that must be coded *exactly* as shown. If the subfield does not contain the maximum number of characters, you must specify trailing blanks to fill out the field. For example, if you choose the NO option from STATS(YES|NO ), then 'NO ' is passed as a parameter.

**data-set-list**
A list of data set names or a ddname coded as a character string. The string must be enclosed with parentheses. If a ddname is used, only one must be specified; for example:

```
'(MYDD1)'
```

If a list of data set names is used, a maximum of 15 data set names can be specified. Data set names must conform to TSO data set naming conventions. When several data set names are included in the list, they must be separated by commas or blanks. For example:

```
'('USERID1.PANELS1',PANELS2,PANELS3,'PROJECT1.PANELS')'
```

## The ISPEXEC Interface

You can use the command function form for service requests in a program function by using the call format of ISPEXEC. Excluding calls in FORTRAN, Pascal, and APL2, the general call format for invoking ISPF services from program functions by using ISPEXEC is:

```
CALL  ISPEXEC (buf-len, buffer);
```

The following services are *not* available when you use CALL ISPEXEC, but are available when you use ISPLINK:

```
GRERROR     VDELETE
GRINIT      VMASK
GRTERM      VREPLACE
VCOPY       VRESET
VDEFINE
```

## CALL ISPEXEC Parameters

**buf-len** Specifies a fullword fixed binary integer containing the length of the buffer.

**buffer** Specifies a buffer containing the name of the service and its parameters just as they would appear in an ISPEXEC invocation for a command invocation written in CLIST.

The maximum buffer size is 32767 bytes.

All services that are valid through ISPEXEC command invocation statements are valid through the CALL ISPEXEC interface.

## Using Parameters as Symbolic Variables

The ISPEXEC call interface allows you to specify parameters as symbolic variables. A symbolic variable is one that is preceded by an ampersand (&). Before a scan syntax check of a statement, variable names and the ampersands that precede them are replaced with the values of the corresponding variables. A single scan takes place.

Standard register conventions are used. Registers 2 to 14 are preserved across the call. However, you must use the VL keyword in Assembler call statements.

## FORTRAN and Pascal

The general call format for invoking ISPF services from FORTRAN or Pascal functions is either of the following:

```
lastrc = ISPLNK (service-name, parameter1, parameter2, ...)
lastrc = ISPEX (buf-len, buffer)
```

The parameters for ISPLNK and ISPEX are the same as those for ISPLINK, as described in "CALL ISPLINK Parameters" on page 5, and for ISPEXEC, as described in "CALL ISPEXEC Parameters" on page 7.

The *lastrc* variable is both a FORTRAN and a Pascal integer variable that contains the return code from the specified ISPF service. The *lastrc* variable is any valid FORTRAN or Pascal name.

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

ISPF services can be issued from dialog function modules that reside either below or above the 16-megabyte line. The dialog interface module ISPLINK (and alias entry points ISPLNK, ISPEXEC, ISPEX, and ISPQRY) has the attributes RMODE(ANY) and AMODE(ANY). This allows a 31-bit addressing mode caller. Data areas below the 16-megabyte line are also supported.

**Note:** The ISPLINK module is shipped with the RMODE(ANY). The load module is link-edited RMODE(24) and AMODE(ANY) to maintain compatibility with ISPF dialogs that have the AMODE(24) attribute and that use a LOAD and CALL interface to ISPLINK. ISPLINK can reside above the 16-megabyte line.

**FORTRAN Examples:**

```
INTEGER     LASTRC*4
CHARACTER   SERVIS*8,TABLE*8,OPTION*8
DATA        SERVIS/'TBOPEN  '/
DATA        TABLE/'XTABLE  '/
DATA        OPTION/'NOWRITE '/

⋮
LASTRC=ISPLNK(SERVIS,TABLE,OPTION)
```

```
INTEGER    LASTRC *4
CHARACTER  SERVIS *8 ,DATAID *8 ,OPTION *8
DATA       SERVIS/'LMOPEN '/
DATA       OPTION/'INPUT '/
⋮
LASTRC = ISPLNK(SERVIS, DATAID, OPTION)
```

For FORTRAN service requests, you can use literals in assignment statements to initialize parameter variables. You must use previously defined constants in assignment statements. For example:

```
CHARACTER  LMOPEN *8 ,SERVIS *8
DATA       LMOPEN/'LMOPEN  '/
⋮
SERVIS = LMOPEN
```

**Pascal Example:**

```
FUNCTION ISPLNK:INTEGER; EXTERNAL;
CONSTANT SERVIS='LMOPEN ';
         OPTION='INPUT  ';
```

```
VAR      LASTRC:INTEGER;
         DATAID:STRING(8);

BEGIN
:
LASTRC:=ISPLNK(SERVIS,DATAID,OPTION);
```

For functions written in Pascal, arguments can also be passed as variables or as literals.

## APL2

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**
The character-vector is a single-character vector that contains all parameters to be passed to the dialog service. The format is the same as dialog service statements for command languages. The first parameter in the vector must be the name of the service to be invoked.

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

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:.........
```

For information about using APL2 with ISPF, refer to the *ISPF Dialog Developer's Guide and Reference*.

**APL2 Examples:**

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

The following example uses the LMOPEN service and checks the return code that is placed in variable LASTCC.

```
LASTCC <- ISPEXEC 'LMOPEN DATAID INPUT'
-> (LASTCC = 0) / NORMALCONT
⋮
```

## PL/I

In PL/I programs, you should include the following declare statements:

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

**PL/I Examples:**

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

```
DECLARE  SERVICE CHAR(8) INIT('LMOPEN  '),
         DATAID  CHAR(8),
         OPTION  CHAR(8) INIT('INPUT   ');

⋮
CALL  ISPLINK (SERVICE, DATAID, OPTION);
```

For service calls in PL/I, you can use literals in assignment statements to initialize parameter values, as in:

```
SERVICE='LMOPEN  ';
```

## COBOL

COBOL does not allow literals within a call statement. Therefore, ISPF does not require the use of literals. You can specify all parameters as variables, as in the following examples:

**COBOL Examples:**

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

```
WORKING-STORAGE SECTION.
   77  SERVIS      PICTURE A(8) VALUE 'LMOPEN  '.
   77  DATAID      PICTURE A(8).
   77  OPTSHUN     PICTURE A(8) VALUE 'INPUT   '.
    .
    .
    .
PROCEDURE DIVISION.
   CALL 'ISPLINK' USING  SERVIS DATAID OPTSHUN.
```

For service calls in COBOL, you can use literals in assignment statements to initialize parameter variables, as in:

```
MOVE 'LMOPEN ' TO SERVIS.
```

## C

The general call format for invoking ISPF services from C functions is either of the following:

```
retcode = isplink (service-name, parameter1, parameter2...);

retcode = ISPEXEC (buflen, buffer)
```

The retcode variable is a C integer variable used to store the return code on the service you are using. For more information about using C with ISPF, refer to the *ISPF Dialog Developer's Guide and Reference*.

**C Examples:**

```
#include <stdio.h>
#include <string.h>
#pragma linkage (isplink, OS)
#define SERVICE "'LMOPEN '"
#define OPTION "'INPUT '"
main ()
{
extern int isplink();

int retcode;

char8 DATAID;

.
.
.

strcpy (DATAID,  "DATA    ");
retcode = isplink (SERVICE, DATAID, OPTION);
}
```

## Assembler

You can use the CALL Assembler macro to invoke ISPF services from Assembler routines as follows:

```
CALL ISPLINK,(SERVICE, parameter-1,parameter-2,...),VL

CALL ISPEXEC,(BUFLEN,BUFFER),VL
```

When using the CALL macro, you must use the VL keyword.

The return code from a call to ISPLINK or ISPEXEC is returned to the Assembler routine in register 15.

The following example shows an Assembler routine that invokes the LMINIT and LMFREE services.

**Assembler Example:**

```
SAMPLE   TITLE 'DO AN LMINIT AND THEN LMFREE'
SAMPLE   CSECT
         USING SAMPLE,15
         B     PASTID                    BRANCH AROUND I.D.
         DC    C'LMINIT  &SYSDATE'
PASTID   EQU   *
         STM   14,12,12(13)              SAVE CALLER'S REGS
         LR    12,15                     ESTABLISH A BASE
         DROP  15                        GIVE UP REG 15
```

```
         USING SAMPLE,12                  USE REG 12 AS BASE
         LA    11,SAVEOS                  POINT TO 'MY' SAVE AREA
         ST    13,4(0,11)                 STORE FORWARD POINTER
         ST    11,8(0,13)                 STORE BACKWARD POINTER
         LR    13,11                      LOCAL SAVE AREA POINTER
         SPACE
*********************************************************************
*        DEFINE VARIABLES TO ISPF                                   *
*********************************************************************
         CALL  ISPLINK,(VDEFINE,DATAID,DATA,CHAR,LNDATA),VL
         SPACE
*********************************************************************
*        INVOKE THE LMINIT SERVICE                                  *
*********************************************************************
         CALL  ISPLINK,(LMINIT,DATAID,B,B,B,B,B,B,DSN),VL
         SPACE
         LR    4,15                       PUT RETCODE IN REG 4
         SPACE
*********************************************************************
*        INVOKE THE LMFREE SERVICE                                  *
*********************************************************************
         CALL  ISPLINK,(LMFREE,DATA),VL
         SPACE
         LR    4,15                       PUT RETCODE IN REG 4
         SPACE
*********************************************************************
*        CLEAN UP VDEFINES                                          *
*********************************************************************
         CALL  ISPLINK,(VDELETE,DATAID),VL
         L     13,SAVEOS+4                GET CALLER'S SAVE AREA
         LM    14,12,12(13)               RESTORE CALLERS REGS
         SR    15,15                      GO BACK WITH RETURN CODE 0
         BR    14                         LEAVE THIS MODULE
         CNOP  0,8
         LTORG
LNDATA   DC    F'8'                       LENGTH OF DATA
VDEFINE  DC    CL8'VDEFINE '              VDEFINE SERVICE
VDELETE  DC    CL8'VDELETE '              VDELETE SERVICE
LMINIT   DC    CL8'LMINIT  '              LMINIT SERVICE
LMFREE   DC    CL8'LMFREE  '              LMFREE SERVICE
DATAID   DC    CL8'DATA    '              VARIABLE
CHAR     DC    CL4'CHAR'                  VARIABLE
DSN      DC    C'PDFUSER.SAMPLE.PDS'' '  DATA SET NAME
DATA     DC    CL8'        '              DATAID SAVE AREA
SAVEOS   DS    18F                        STANDARD SAVE AREA
B        DC    CL1' '                     SINGLE BLANK
         LTORG
         END   SAMPLE
```

## Return Codes from Services

Each service returns a numeric code, called a return code, indicating the results of the operation. These return codes are summarized in Table 1.

*Table 1. Service Return Codes*

| Operation Results | Return Code | Reason |
|---|---|---|
| Normal completion | 0 | Indicates that the service completed operation without errors. |

*Table 1. Service Return Codes (continued)*

| Operation Results | Return Code | Reason |
|---|---|---|
| Exception condition | 4, 8 | Indicates a condition that is not necessarily an error, but that the dialog should be aware of. A return code of 4 is informational, while an 8 generally indicates a non-terminating error condition, such as the end of a data set or member list. |
| Error condition | 10, 12, 14, 16, 20 | Indicates that the service did not complete operation because of errors. Use the CONTROL service to control errors with a return code of 12 or greater. Return codes of 10 and 14 are particular to PDF component services. |

Return codes and their meanings vary for each service and are listed with each service description in this chapter.

## Command Invocation Return Code Variable

For a command invocation, the return code is returned in the CLIST variable LASTCC.

## Call Invocation Return Code Variables

For call invocation, the return code is returned in register 15 or, in FORTRAN and Pascal programs, in registers 15 and 0. In APL2, the return code is placed on the execution stack by the ISPEXEC function.

**FORTRAN and Pascal**

FORTRAN and Pascal programs can examine the return code by using an integer variable, such as *lastrc* in the following example:

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

**PL/I**

PL/I programs can examine the return code by using the PLIRETV built-in function. The following declaration statements are required:

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

or alternatively:

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

**COBOL**

COBOL programs can examine the return code by using the built-in RETURN-CODE variable.

## Return Code of 12 or Higher

The dialog can use the ISPF dialog management CONTROL service to set the error mode to RETURN, or CANCEL, which is the default. See the CONTROL service in Chapter 2, “Description of the ISPF Services,” on page 21 for detailed information.

The error mode setting determines what happens when a return code is 12 or higher. There are two error modes:

**CANCEL** Displays and logs a message, then stops the dialog and displays the previous Primary Option Menu.

**RETURN** Formats an error message, but does not display or log it. Returns to the function that invoked the service, passing back the designated return code.

In CANCEL mode, control is not usually returned to the function that invoked the service. Consequently, the function does not see a return code of 12 or higher, so you do not have to include logic to process errors of this severity.

However, ISPLINK returns a code of 20 to the dialog when an invalid ISPF environment causes the error. In this situation, ISPF cannot display a panel to show the error. Control is returned to the dialog, even though the return code is 12 or higher.

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

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

## System Variables Used to Format Error Messages

If an error occurs, an error message is formatted before control returns to the function. The following list defines the contents of the system variables that are used to format error messages:

| Variables | Contents |
|---|---|
| **ZERRMSG** | Message ID. |
| **ZERRSM** | Short-message text in which variables have been resolved. |
| **ZERRLM** | Long-message text in which variables have been resolved. |
| **ZERRHM** | The name of a Help panel, if one was specified in the message definition. |
| **ZERRALRM** | The value YES if an alarm was specified in the message definition (.ALARM=YES); otherwise, the value NO. |

ZERRMSG, ZERRSM, and ZERRLM are changed only when the return code from a DM component service is greater than 8.

These system variables are in the function pool, if it exists. Otherwise, they are in the shared variable pool.

The function can display the message, log the message, or both, simply by invoking the appropriate service with the message ID ISRZ002. For example:

```
ISPEXEC  SETMSG  MSG( ISRZ002 )
ISPEXEC  LOG  MSG( ISRZ002 )
```

The service provides the short- and long-message text, the name of the corresponding help panel, and the alarm setting for your use.

### Return Codes from I/O and Command Routines

EDIF and BRIF invoke routines supplied on the service invocation to perform I/O and primary command processing. Specific return codes are expected of these routines and are grouped into four categories:

**0** Normal completion.

**4** ISPF should process the request.

**8** End of file.

**12, 16, and 20**
Error conditions; the specified functions did not complete because of errors.

Return codes for these functions are described in greater detail in the EDIF and BRIF sections in Chapter 2, "Description of the ISPF Services."

## A Summary of the ISPF Services

### Display Services

**ADDPOP** 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** 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.

**REMPOP** Removes a pop-up window from the screen.

**SELECT** Used to display a hierarchy of selection panels or invoke a function.

**SETMSG** 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 component service.

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

### File Tailoring Services

The file tailoring services, listed in the order in which 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.

## Library Access Services

**DSINFO** Returns information about a particular data set in dialog variables in the function pool.

**LMCLOSE** Closes a data set.

**LMCOMP** Compresses a partitioned data using either the new compress request exit or IEBCOPY if the exit is not installed.

**LMCOPY** Copies partitioned data set members or sequential data sets, allowing pack, lock, and automatic truncation options.

**LMDDISP** Displays the data set list for a specified dslist ID.

**LMDFREE** Removes the link between a dslist ID and a DSNAME LEVEL and VOLUME combination.

**LMDINIT** Associates a DSNAME LEVEL and VOLUME combination with a dslist ID. Thereafter, this dslist ID is used to identify the DSNAME LEVEL and VOLUME combination for processing by other library access services.

**LMDLIST** Creates a data set list for a specified dslist ID.

**LMERASE** Deletes an ISPF library or MVS data set.

**LMFREE** Releases the data set associated with a given data-id.

**LMGET** Reads one record of a data set.

**LMINIT** Associates one or more ISPF libraries or an existing data set with a data-id. Thereafter, this data-id is used to identify the data set for processing by other library access services.

**LMMADD** Adds a member to an ISPF library or a partitioned data set.

**LMMDEL** Deletes a member of an ISPF library or a partitioned data set.

**LMMDISP** Provides member selection lists for:

- Single partitioned data sets
- Concatenations of up to four partitioned data sets.

**LMMFIND** Finds a member of an ISPF library or a partitioned data set.

**LMMLIST** Creates a member list of an ISPF library or a partitioned data set.

**LMMOVE** Moves partitioned data set members or sequential data sets, allowing pack and automatic truncation options.

**LMMREN** Renames a member of an ISPF library or a partitioned data set.

**LMMREP** Replaces a member of an ISPF library or a partitioned data set.

**LMMSTATS** Sets and stores, or deletes ISPF statistics for partitioned data set members that have fixed-length or variable-length records.

**LMOPEN** Opens a data set.

**LMPRINT** Prints to the list data set, with formatting optional.

**LMPUT** Writes one record of a data set.

**LMQUERY** Provides requested information regarding the data set associated with a given data-id.

**LMRENAME** Renames an ISPF library.

**MEMLIST** Enables access to the Library Utility member list from within a dialog.

## PDF Component Services

PDF component services consist of BRIF (Browse Interface), BROWSE, EDIF (Edit Interface), EDIREC (edit recovery for EDIF), EDIT, VIEW, VIIF, and EDREC (edit recovery for EDIT and VIEW), along with the library access services mentioned earlier.

**BRIF** Provides browse functions for data accessed through dialog-supplied I/O routines. It allows you to browse data other than partitioned data sets or sequential files, such as subsystem data and in-storage data, and to preprocess the data being browsed.

**BROWSE** Can be used to look at any ISPF library, concatenation of ISPF libraries, or data set that can be allocated by using the LMINIT service, and certain other data types not supported by ISPF. You can browse host data sets on the workstation or workstation files on the host.

**EDIF** Provides edit functions for data accessed through dialog-supplied I/O routines. It allows you to edit data other than partitioned data sets or sequential files, such as subsystem data and in-storage data, and to preprocess the data being browsed.

**EDIREC** Initializes an edit recovery table (ISREIRT) for use by the EDIF service and determines whether recovery from the EDIF service is pending.

**EDIT** Can be used to look at any ISPF library, concatenation of ISPF libraries, or data set that can be allocated by using the LMINIT service. The EDIT service provides an interface to the PDF editor and bypasses the display of the Edit Entry Panel on the host. You can also edit host files on the workstation or workstation files on the host.

**EDREC** Initializes an edit or view recovery table, determines whether recovery is pending, and takes the action specified by the first argument.

**VIEW** Functions exactly like the EDIT service, with the following exceptions:

1. You must use the REPLACE or CREATE primary command to save data.
2. When you enter the END primary command after altering a file in VIEW mode, you will be prompted to either save the changes or exit without saving them.

**VIIF** Provides edit functions for data accessed through dialog-supplied I/O routines. It enables you to view data other than partitioned data sets or sequential files, such as subsystem data and in-storage data, and to preprocess the data being viewed.

## Table Services

**Services that Affect 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 it. |
| **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 it. |
| **TBSORT** | Sorts a table. |
| **TBSTATS** | Provides access to statistics for a table. |

**Services that Affect Table Rows**

| | |
|---|---|
| **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 backward 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. |

## Variable Services

**All Functions**

| | |
|---|---|
| **VERASE** | Removes variables from the shared pool and/or profile pool. |
| **VGET** | Retrieves variables from the shared pool or profile pool. |
| **VPUT** | Updates variables in the shared pool or profile pool. |

**Program Functions Only**

| | |
|---|---|
| **VCOPY** | Copies data from a dialog variable to the program. |
| **VDEFINE** | Defines function program variables to ISPF. |
| **VDELETE** | Removes the definition of function variables. |
| **VMASK** | Associates a mask with a dialog variable. |
| **VREPLACE** | Updates dialog variables with program data specified in the service request. |
| **VRESET** | Resets function variables. |

## Miscellaneous Services

**CONTROL** Allows a function to condition ISPF to expect certain kinds of display output, or to control the disposition of errors encountered by dialog management services.

**FILESTAT** Provides statistics about a site on the connected workstation.

**FILEXFER** Uploads files from the workstation or downloads files to the workstation.

**GETMSG** Obtains a message and related information and stores them in variables specified in the service request.

**GRERROR** Provides access to the address of the GDDM error record and the address of the GDDM call format descriptor module.

**GRINIT** Initializes the ISPF/ GDDM® interface and optionally requests that ISPF define a panel's graphic area as a GDDM graphics field.

**GRTERM** Terminates a previously established GDDM interface.

**LIBDEF** Provides applications with a method of dynamically defining application data element files while in an active ISPF session.

**LIST** Allows a dialog to write data lines directly (without using print commands or utilities) to the ISPF list data set.

**LOG** 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** Returns information for a specific area on a specific panel. The type, size, and position characteristics associated with the area are returned in variables.

**QBASELIB** Enables an ISPF dialog to obtain the current library information for a specified DDNAME.

**QLIBDEF** Allows an ISPF dialog to obtain the current LIBDEF definition information, which can be saved by the dialog and used later to restore any LIBDEF definitions that may have been overlaid.

**QTABOPEN** Allows an ISPF dialog to obtain a list of currently open ISPF tables. The TBSTATS or TBQUERY service can then be used to obtain more detailed information about each table.

**QUERYENQ** Allows an ISPF dialog to obtain a list of all system enqueues, or all system enqueues that match the specified criteria.

**TRANS** Translates data from one Coded Character Set Identifier (CCSID) to another.

**WSCON** Enables the user to connect to the workstation without using the GUI parameter on the ISPSTART command and the Initiate Workstation Connection panel from the ISPF settings.

**WSDISCON** Enables the user to disconnect from the workstation without having to terminate the ISPF session.

# Chapter 2. Description of the ISPF Services

The services are listed in alphabetical order.

Each service description consists of the following information:

**Description** A description of the function and operation of the service. This description also refers to other services that can be used with this service.

**Format** The syntax used to code the service, showing both command invocation and call invocation.

**Parameters** A description of any required or optional keywords or parameters.

**Return Codes** A description of the codes returned by the service. For all services, a return code of 12 or higher implies a severe error. This error is usually a syntax error, but can be any severe error detected when using the services.

**Examples** Sample usage of the services.

## ADDPOP – Start Pop-Up Window Mode

The ADDPOP service notifies the dialog manager that all subsequent panel displays are to appear in a pop-up window. No visible results appear on the screen until you issue a DISPLAY, TBDISPL, or SELECT PANEL call.

All subsequent panel displays will be in the pop-up window created with the ADDPOP call, until a REMPOP or another ADDPOP is called. Another ADDPOP call creates a separate pop-up window.

Each pop-up window created as a result of a successful ADDPOP service call can also have a window title. The title is embedded in the top of the window frame border and can be only one line 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.

### Command Invocation Format

```
ISPEXEC ADDPOP [POPLOC(field-name)]
               [ROW(row)]
               [COLUMN(column)]
```

### Call Invocation Format

```
CALL ISPEXEC  (buf-len, buffer);
CALL ISPLINK  ('ADDPOP  '  [,field-name]
              [,row] [,column ]);
```

### Parameters

The *field-name*, *row*, and *column* parameters are optional.

**Note:** When running in GUI mode, the field-name, row, and column parameters are ignored.

© Copyright IBM Corp. 1980, 2004

If you omit the field-name parameter when using the ADDPOP service, the Dialog Manager offset positions the pop-up window so that the title of the underlying panel is visible, and horizontally four character spaces to the right of the underlying panel.

If the pop-up window will not fit relative to the ADDPOP positioning parameters, the Dialog Manager overrides these parameters and adjusts the window so that it fits on the screen.

**field-name** Specifies that the dialog manager is to position the pop-up window relative to the specified field in the currently displayed panel. If omitted, the pop-up window is offset positioned relative to the active window.

**row** Specifies that the dialog manager is to adjust the field-specific location row or offset location row by the specified amount. This amount can be either positive or negative. The default value is 0.

**column** Specifies that the dialog manager is to adjust the field-specific location column or offset location column by the specified amount. This amount can be positive or negative. The default value is 0.

**buf-len** Specifies a fullword fixed binary integer containing the length of *buffer*.

**buffer** Specifies a buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC call for a command procedure.

## Return Codes

The following return codes are possible:

**0** Normal completion.

**12** An ADDPOP service call was issued before a panel was displayed or another ADDPOP service call was issued before a panel was displayed for the previous ADDPOP call.

**20** Severe error.

## Example

The following EXEC called from the ISPF Primary Option panel:

```
/*  REXX  */
ADDRESS  ISPEXEC
"ADDPOP"
"DISPLAY PANEL(PANELA)"
"ADDPOP POPLOC(FIELD2)"
ZWINTTL = "POPUP WINDOW TITLE"
"DISPLAY PANEL(PANELB)"
"ADDPOP COLUMN(5) ROW(3)"
ZWINTTL = ""
"DISPLAY PANEL(PANELC)"
EXIT
```

results in the panel displayed in Figure 5.

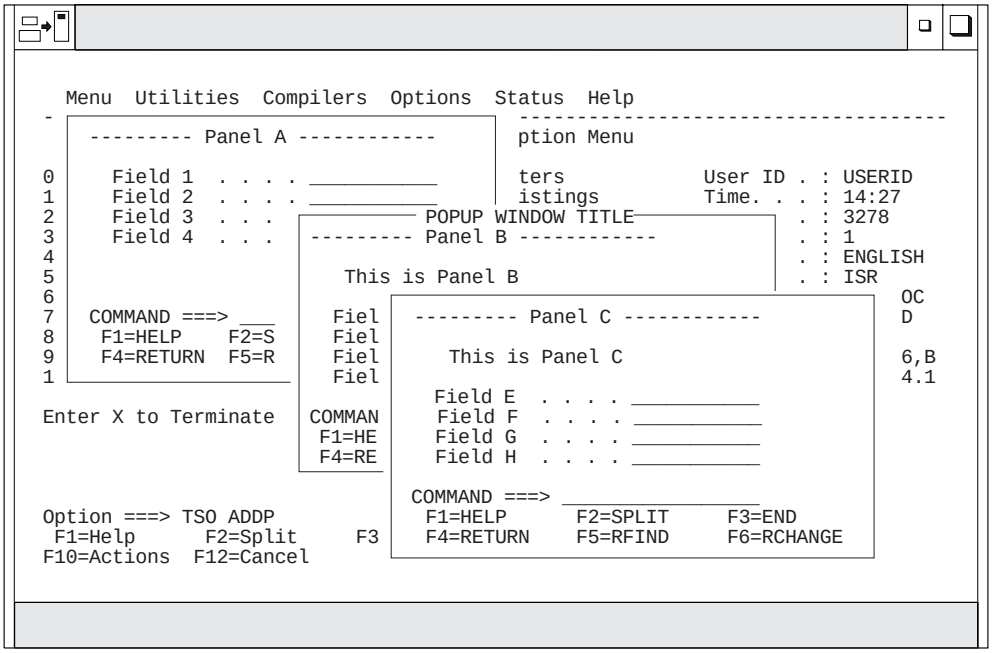



*Figure 5. Multiple Pop-up Windows*

# BRIF – Browse Interface

The Browse Interface (BRIF) service provides browse functions for data accessed through dialog-supplied I/O routines. The invoking dialog must perform all environment-dependent functions such as file allocation, opening, reading, closing, and freeing. The dialog is also responsible for any Enqueue/Dequeue serialization that is required. With the dialog providing the I/O routines, BRIF allows you to:

- Browse data other than partitioned data sets or sequential files, such as subsystem data and in-storage data.
- Do preprocessing of the data being browsed.

The invoking dialog provides addresses of routines that will:

- Respond to a read request for a specific record by its relative position in the data.
- Perform processing for the BROWSE primary command. If this routine is not provided, ISPF will process any request for the BROWSE primary command.

All addresses must be 31-bit addresses, and the routines must have an addressing mode (AMODE) of 31.

The dialog-supplied read, write, and command processing routines are called directly by ISPF at the same task level (TCB) that displays the ISPF screens. If you need to ensure that your program runs at the same task level as the routines, use the SELECT PGM( ) service to start your program. This may be a factor if your program expects to create or share data spaces or other task-specific resources between the main program and the read, write, or command routines.

## Command Invocation Format

Command procedures cannot be used to invoke this service.

## Call Invocation Format

```
CALL ISPLINK ('BRIF    ', [data-name] ,rec-format
                          ,rec-len  ,read-routine

                          ,[cmd-routine] ,[dialog-data]

                          ,[panel-name] ,[format-name]

                          ,['YES     '|'NO      ']);
```

## Parameters

**data-name**
This parameter allows you to specify a data name for the source data to be browsed. This name will be displayed in the Title line of the default Browse panel; if *data-name* is not specified, no name is displayed on the panel. This parameter must not have any embedded blanks, and its maximum length is 54 characters. This name is stored in ZDSNT in the function pool.

**rec-format**
The record format of the data to be browsed:
- F - fixed
- FA - fixed (ASA printer control characters)
- FM - fixed (machine code printer control characters)
- V - variable
- VA - variable (ASA printer control characters)
- VM - variable (machine code printer control characters)
- U - undefined.

**rec-len**
The record length, in bytes, of the data to be browsed. For variable and undefined record formats, this is the maximum record length. This parameter must be a positive numeric value with a maximum value of 32,760 bytes.

The dialog can hide data during a Browse session by specifying the record length to be less than the actual data being browsed. By doing this, BRIF displays only the data up to the specified record length.

**read-routine**
A fullword address indicating the entry point of a dialog-supplied read routine. See “Read Routine” on page 25 for more information about this parameter.

If a *read-routine* displays its own panel, then a `CONTROL DISPLAY SAVE` should be done at the beginning of the panel and a `CONTROL DISPLAY RESTORE` should be done at the end.

**cmd-routine**
A fullword address indicating the entry point of a dialog-supplied routine that processes the BROWSE primary command or any dialog-specific primary commands. See “Command Routine” on page 26 for more information about this parameter. If this parameter is not specified, ISPF initiates a recursive Browse session to handle any request for the BROWSE primary command.

If a *cmd-routine* displays its own panel, then a `CONTROL DISPLAY SAVE` should be done at the beginning of the panel and a `CONTROL DISPLAY RESTORE` should be done at the end.

**dialog-data**
A fullword address indicating the beginning of a dialog data area. This address

is passed to the dialog-supplied routines. If no address is supplied, zeros are passed to the dialog routines. This data area provides a communication area for the dialog.

**panel-name**
The name of the panel to use for displaying the data. The default is the standard Browse data display panel. Refer to *ISPF Planning and Customizing* for information about developing a customized panel.

**format-name**
The name of the format to be used to reformat the data. The default is no format. This parameter is provided to support the IBM 5550 terminal using the Double-Byte Character Set (DBCS).

**YES | NO**
Specifies whether the data is treated as mixed-mode DBCS data. If YES is specified, the BRIF service treats the data as mixed-mode DBCS data. If NO (the default value) is specified, the data is treated as EBCDIC (single-byte) data. This parameter is provided to support the IBM 5550 terminal using the Double-Byte Character Set (DBCS).

## Dialog-Supplied Routines

The dialog-supplied routines are invoked by using standard linkage. Addresses must be 31-bit addresses, and the addressing mode (AMODE) of the routines must be AMODE=31.

The dialog-supplied read, write, and command processing routines are called directly by ISPF at the same task level (TCB) that displays the ISPF screens. If you need to ensure that your program runs at the same task level as the routines, use the SELECT PGM( ) service to start your program. This may be a factor if your program expects to create or share data spaces or other task-specific resources between the main program and the read, write, or command routines.

**Note:** The dialog-supplied routines and the read and command exits can be written in languages that use the Language Environment® (LE) run-time environment. However, a mixture of LE-conforming main dialog code and service routine code is not supported. Dialogs and service routines must either all be LE-conforming or all be LE-nonconforming.

## Read Routine

The read routine is invoked with the following parameters:

- Fullword pointer to record data read (output from read routine)
- Fullword fixed binary data length of the record read if the rec-format parameter is V, VA, VM, or U (output from read routine)
- Fullword relative record number:
  - Record-requested input to read routine
  - Record-provided output from read routine when return code is 4 or 8.
- Fullword dialog data area address.

BRIF calls the read routine as the data records are required to be displayed. Data not being displayed is not retained.

After the first screen of data is displayed, the first SCROLL DOWN MAX command results in a request to the dialog read routine for relative record number 99999999 (this is the maximum number of records allowed for browsing). The read

routine is responsible for determining the relative record number of the last record in the data. It must return that last record number, and a pointer to the data with a return code of 4; the end of file is temporary, or 8, if the end of file is permanent. When BRIF receives this response, it uses the last record number to determine the relative record number of the first data record that should appear on the display (last record number minus the number of data lines on the display + 1). BRIF then calls the read routine requesting this first data display record, and subsequently requests all following records up to the last record in the data to fill the display.

The read routine should maintain the previous record number requested so that on the next read request a determination can be made whether the requested record is the next record in the data. This could save a considerable amount of processing time in the read routine, since data records are frequently requested in sequential order for partitions of data.

If an I/O error occurs while attempting to read to the end of data, the read routine returns the relative record number of the record causing the I/O error with a return code of 8. When BRIF requests this record number again to format the screen, the read routine then issues a return code of 16, indicating a read error.

The BRIF service requests and displays all additional records beyond the temporary end of data (return code 4) if you attempt to scroll down past the end of data or cause any interrupt (such as Enter) when the end-of-data line is present on the display.

If you decrease the number of records during the BRIF session, the read routine can set a new last record number that is smaller than the current value with return code 4.

When the BRIF service receives a return code 8, it sets the last record number as the permanent end of file. The BRIF service does not request any additional records beyond the permanent end of file.

## Command Routine

The dialog-supplied command routine, when specified, is called to process the BROWSE primary command or any dialog-specific primary commands. The Command Routine is invoked with two parameters:

- A Fullword fixed binary function code indicating the type of command.

  **10** Recursive Browse

  **20** A command not recognized by browse. The command can be a dialog-specific command or an invalid command. The command routine is responsible for getting the command from the variable ZCMD and any necessary parsing of the command. If the command routine was not specified or if the command routine returns a return code of 4, BRIF issues an `INVALID COMMAND` message.
- A Fullword dialog data area address.

## Return Codes

When a dialog routine terminates with a return code (12 or higher or an unexpected return code), the dialog can issue a SETMSG to generate a message on the next panel display. If the dialog does not set a message, the BRIF service will issue a default message.

## Read Routine Return Codes

**0** Normal completion.

**4** Temporary end of file.

**8** Record requested beyond end of data. The relative record number of the last data record and a pointer to the last data record are returned.

**16** Read error. Browse data obtained up to the read error is formatted and displayed with an indication that a read error was encountered.

**20** Severe error. (The BRIF service terminates immediately with a return code of 20.)

## Command Routine

**0** Normal completion.

**4** ISPF should process the requested function.

**12** Command deferred; retain the command on the Command line. Browse data is redisplayed.

**20** Severe error. (The BRIF service terminates immediately with a return code of 20.)

Errors that the BRIF service cannot handle must be handled by the dialog; for example, environment-dependent errors would be processed by the dialog.

## BRIF service

**0** Normal completion.

**12** No data to browse.

**16** Unexpected return code received from a dialog-supplied routine; unable to continue. When an unexpected return code is received, the BRIF service terminates immediately with a return code of 16.

**20** Severe error; unable to continue.

After the Browse session has been terminated, control is returned to the dialog with a return code indicating the completion status of the service.

## Example

This example invokes the BRIF service to browse data called SPOOL.DATA, which has a variable record format with a maximum record length of 132 characters. The READRTN read routine reads the data records. The CMDRTN command routine processes the BROWSE primary command and any dialog-specific primary commands.

**Call Invocation:**

```
CALL ISPLINK('BRIF    ','SPOOL.DATA ','V ',132,READRTN,CMDRTN,' ',
             ' ',' ','NO      ');
```

For a more complete example of using BRIF, including dialog-supplied I/O routines and source code, see the *ISPF Dialog Developer's Guide and Reference*.

# BROWSE – Browse a Data Set

The BROWSE service provides an interface to the Browse function and bypasses the display of the View Entry Panel. For a complete description of the Browse function see the *ISPF User's Guide Volume II*.

The BROWSE service allows you to use a customized panel for displaying data. Panel ISRBROB should be used as a model for your customized panel. You can also use the BROWSE service recursively, either through nested dialogs or by entering a BROWSE command while browsing. BROWSE allows you to browse up to a maximum of 99,999,999 records.

BROWSE can be used to look at any ISPF library, concatenation of ISPF libraries, or data set that can be allocated by using the LMINIT service. However, it cannot be used by a PL/I main program that also uses subtasking. The BRIF service allows you to browse data types not supported by ISPF.

## Command Invocation Format

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

                                  [PANEL(panel-name)]

                                  [FORMAT(format-name)]

                                  [MIXED(YES|NO)]

OR

ISPEXEC BROWSE  DATAID(data-id)    [MEMBER(member-name)]

                                   [PANEL(panel-name)]

                                   [FORMAT(format-name)]

                                   [MIXED(YES|NO)]
```

## Call Invocation Format

```
CALL  ISPLINK ('BROWSE  ', {dsname} ,[serial]

                                    ,[pswd-value]

                                    ,[panel-name]

                                    ,{data-id}

                                    ,[member-name]

                                    ,[format-name]

                                    ,['YES     '|'NO      '];
OR
CALL  ISPEXEC (buf-len, buffer);
```

## Parameters

**dsname**
The data set name, in TSO syntax, of the data set to be browsed. This is equivalent to the "other" data set name on the View Entry Panel. You can

specify a fully qualified data set name enclosed in apostrophes. If the apostrophes are omitted, the TSO data set prefix from the user's TSO profile is automatically attached to the data set name. The maximum length of this parameter is 56 characters.

For ISPF libraries and MVS partitioned data sets, you can specify a member name or pattern enclosed in parentheses. If a member name is not included, or a pattern is specified as part of the dsname specification when the DATASET keyword is used, a member selection list for the ISPF library, concatenation of libraries, or MVS partitioned data set is displayed. See the section on naming ISPF libraries and data sets in the *ISPF User's Guide Volume I* for a complete description of patterns and pattern matching.

**Note:** You can also specify a VSAM data set name. If a VSAM data set is specified, ISPF checks the ISPF configuration table to see if VSAM support is enabled. If it is, the specified tool is invoked. If VSAM support is not enabled, an error message is displayed.

**serial**
The serial number of the volume on which the data set resides. If you omit this parameter or code it as blank, the system catalog is searched for the data set name. The maximum length of this parameter is 6 characters.

**pswd-value**
The password if the data set has MVS password protection. Do not specify a password for data sets that are protected by Resource Access Control Facility (RACF®) or Password Checking Facility (PCF).

**panel-name**
The name of a customized browse panel that you create, to be used when displaying the data. Refer to *ISPF Planning and Customizing* for information about developing a customized panel.

**format-name**
The name of the format to be used to reformat the data. The format-name parameter is provided to support the IBM 5550 terminal using the Double-Byte Character Set (DBCS).

**YES | NO**
For the MIXED parameter, if YES is specified, the BROWSE service treats the data as mixed-mode DBCS data. If NO (the default value) is specified, the data is treated as EBCDIC (single-byte) data. This parameter is provided to support the IBM 5550 terminal using the Double-Byte Character Set (DBCS).

**data-id**
The data ID that was returned from the LMINIT service. The maximum length of this parameter is 8 characters.

You can use the LMINIT service in either of two ways before invoking the BROWSE service:

- You can use LMINIT to allocate existing data sets by specifying a data set name or ISPF library qualifiers. LMINIT returns a data ID as output. This data ID, rather than a data set name, is then passed as input to the BROWSE service.
- The dialog can allocate its own data set by using the TSO ALLOCATE command or MVS dynamic allocation, and then pass the ddname to LMINIT. Again, a data ID is returned as output from LMINIT and subsequently passed to the BROWSE service. This procedure is called the

*ddname interface* to BROWSE. It is particularly useful for browsing VIO data sets, which cannot be accessed by data set name because they are not cataloged.

**Note:** Using the data ID of a multivolume data set causes Browse to look at *all* volumes of that data set. If you want to look at just one volume of a multivolume data set, use the data set name and volume number.

**member-name**
A member of an ISPF library or MVS partitioned data set, or a pattern. If you do not specify a member name when the MEMBER keyword or call invocation is used, or if a pattern is specified, a member selection list for the ISPF library, concatenation of libraries, or MVS partitioned data set is displayed.

**buf-len**
A fullword fixed binary integer containing the length of the buffer parameter.

**buffer**
A buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC invocation for a command invocation.

## Return Codes

The following return codes are possible:

**0** Normal completion.

**12** Zero-length data; empty sequential data set or zero-length member of a partitioned data set.

**14** Specified member not found.

**16** One of the following:
- No members matched the specified pattern.
- No members in the partitioned data set.

**18** A VSAM data set was specified but the ISPF Configuration Table does not allow VSAM processing.

**20** Severe error; unable to continue.

## Example

This example invokes the BROWSE service to give you a member list of all members beginning with 'TEL'. A member name can be selected from this member list.

### Command Invocation

```
ISPEXEC BROWSE DATASET('ISPFPROJ.FTOUTPUT(TEL*)')
OR

ISPEXEC LMINIT DATAID(DDBROW)   +
               DATASET('ISPFPROJ.FTOUTPUT')

OR

ISPEXEC BROWSE DATAID(&DDBROW) MEMBER(TEL*)
```

### Call Invocation

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

or

Set the program variable BUFFER to contain:

```
BUFFER = 'BROWSE DATASET('ISPFPROJ.FTOUTPUT(TELOUT)')';
```

Set the program variable BUFLEN to the length of the variable BUFFER. Issue the following:

```
CALL ISPEXEC (BUFLEN, BUFFER);
```

# CONTROL – Set Processing Modes

The CONTROL service defines certain processing options for the dialog environment. It allows a function to condition ISPF to expect certain kinds of display output, or to control the disposition of errors encountered by other DM component services. The processing options control the display screen and error processing.

## Command Invocation Format

```
ISPEXEC  CONTROL { DISPLAY { LOCK                      } }
                 {         { LINE [START(line-number)] } }
                 {         { SM   [START(line-number)] } }
                 {         { REFRESH                    } }
                 {         { SAVE|RESTORE               } }
                 {         { ALLVALID                   } }
                 {                                        }
                 { NONDISPL [ENTER|END] [NOSETMSG]        }
                 {                                        }
                 { ERRORS   [CANCEL|RETURN]               }
                 {                                        }
                 { SPLIT    { ENABLE  }                   }
                 {          { DISABLE }                   }
                 {                                        }
                 { NOCMD                                  }
                 {                                        }
                 { SUBTASK  { PROTECT }                   }
                 {          { CLEAR   }                   }
                 { TSOGUI    [QUERY|OFF|ON]               }
                 { REFLIST   [UPDATE|NOUPDATE]              }
```

## Call Invocation Format

```
CALL  ISPEXEC (buf-len, buffer);
OR

CALL  ISPLINK ('CONTROL ', operand);

      Note: parameters that can appear in "operand" are:

                { 'DISPLAY '{ ,'LOCK    '                 } }
                {           { ,'LINE    ' [,line-number]  } }
                {           { ,'SM      ' [,line-number]  } }
                {           { ,'REFRESH '                 } }
                {           { ,'SAVE    '|'RESTORE '      } }
                {           { ,'ALLVALID'                 } }
                {                                           }
                { 'NONDISPL' [,'ENTER   '|'END     ']       }
                {                                           }
                { 'ERRORS  '   [,'CANCEL  '|'RETURN  '] ['NOSETMSG']    }
                {                                           }
                { 'SPLIT   '   { ,'ENABLE  '  }            }
                {              { ,'DISABLE '  }            }
                {                                           }
                { 'NOCMD   '                                }
                {                                           }
                { 'SUBTASK ' { ,'PROTECT ' }                }
```

```
{           { ,'CLEAR   ' }              }
{ 'TSOGUI  '   [,'QUERY  '|'OFF     '|'ON     ']     }
{ 'REFLIST '   [,'UPDATE '|'NOUPDATE']     }
```

## ADDPOP/REMPOP Service Relation to Control Service

The ADDPOP service performs the equivalent of a CONTROL DISPLAY SAVE prior to creating the pop-up window and the REMPOP service performs the equivalent of a CONTROL DISPLAY RESTORE after removing the current pop-up window. A dialog should not issue its own CONTROL DISPLAY SAVE/RESTORE around an ADDPOP/REMPOP sequence.

## Parameters

**DISPLAY**
Specifies that a display mode is to be set. The valid modes are LOCK, LINE, REFRESH, SAVE, and RESTORE. LINE is in effect until the next display of an ISPF panel. REFRESH occurs on the next display of an ISPF panel.

**LOCK**
Specifies that the next (and only the next) display output (such as displays by the DISPLAY or TBDISPL service) is to leave the terminal user's keyboard locked. ISPF processes the next display output as though the user had pressed the Enter key.

This facility can be used to display an "in process" message during a long-running operation. It is the dialog developer's responsibility to ensure that the keyboard is unlocked by the subsequent display of a message or panel. While the keyboard is locked, the screen is not protected from immediate overlay by non-full-screen messages.

**LINE**
Specifies that terminal line-mode output is expected. This could be from a TSO command or program dialog. The screen is completely rewritten on the next ISPF full-screen write operation, after the lines have been written.

**Notes:**

1. CONTROL DISPLAY LINE is automatically invoked by the SELECT service whenever a SELECT CMD request is encountered, unless the command begins with a percent (%) sign. For example:

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

   The MODE parameter of the SELECT service, described later in this chapter, can be used to override this use of the percent sign.
2. Upon return from a non-ISPF application that turns off full-screen mode, an ISPF application can no longer be in full-screen mode. To assure a return to full-screen mode in ISPF, the dialog should issue CONTROL DISPLAY LINE.

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

The line-number parameter must have an integer value. For a call, it must be a fullword fixed binary integer. The parameter should specify a line value that is not within three lines of the bottom of the logical screen. If the value is within three lines of the bottom of the logical screen, a default line value is used. This value is equivalent to the number of the bottom line of the screen, minus 3.

This parameter is meaningful only when entering line mode. It can be specified with the SM keyword, since SM reverts to LINE if the Session Manager is not installed. Once line mode has been set, subsequent attempts to set line mode (without intervening full-screen output) are ignored. Accordingly, the line-number, once set, cannot be changed.

For DBCS terminals, CONTROL DISPLAY LINE always clears the screen and places the cursor on line 1, regardless of the line-number value.

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

**Note:** If you specify the SM keyword when graphics interface mode is active (for example, a GRINIT service request has been issued, but a GRTERM service request has not yet been issued), Session Manager does not get control of the screen. In this case, the SM keyword is treated as LINE.

**REFRESH**
Specifies that the entire screen image is to be rewritten when the next ISPF-generated full-screen write is issued to the terminal. This facility should be used before or after invoking any program that uses non-ISPF services for generating full-screen output. Be aware that REFRESH does not always result in a return to full screen mode. See note following the description of the LINE keyword.

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

Use of the CONTROL service SAVE and RESTORE parameters allows DISPLAY, TBDISPL, BROWSE, or EDIT processing to be nested. The CONTROL service should be used to save and restore the environment at each level. SAVE and RESTORE must be issued in pairs. Issue SAVE following the screen display; issue RESTORE before the next request for the saved panel.

A command entered by the user in the command field of a displayed panel causes the dialog manager to issue a SELECT service request for the dialog to process the command. The current display environment is automatically saved before invoking the designated dialog. That environment is subsequently restored when the dialog ends.

The current DISPLAY environment that existed before the SAVE is not available to a nested processing level.

Table display service system variables, ZTD*, are not saved as part of the SAVE/RESTORE information. The values of these variables may be saved by the dialog developer before invoking another table display and restored before resuming processing of the initial table display. Also, the ZVERB variable is not saved.

**RESTORE**
Specifies the restoration of information previously saved by CONTROL

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

**ALLVALID**
Specifies that ISPF is to consider all displayed code points from X'40' to X'FE' as valid. This specification applies to all subsequent DISPLAY and TBDISPL service requests within the current SELECT level only and remains in effect until the SELECT level ends. It is not propagated to lower SELECT levels.

It is the responsibility of the dialog to insure that the code points are displayable without a hardware error before issuing this option.

**NONDISPL**
Specifies that no display output is to be issued to the terminal when processing the next panel definition. This option is in effect *only* for the next panel; after that, normal display mode is resumed. Initializing the ZCMD variable to a value may cause a panel to display after 'CONTROL NONDISPL' has been issued. This can be circumvented by using the COMMAND option of the DISPLAY service which will cause the panel specified on the DISPLAY service to be processed in CONTROL NONDISPL ENTER mode.

**Note:** NONDISPL mode stays active until the next panel definition is processed; that is, until the PROC section of a panel display has been completed. Error conditions, such as an error in the panels INIT section, or an action coded in an INIT section, such as .RESP=ENTER, causes panel processing to bypass the panels PROC section, leaving CONTROL NONDISPL active until the PROC section of the next panel is processed.

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

**END**
Specifies that the END command is to be simulated as the user response to the NONDISPL processing for the next panel.

**NOSETMSG**
Specifies that the SETMSG Service message is to be suppressed when the panel on which it was intended to be displayed is suppressed by the CONTROL NONDISPL ENTER Service, but an error when processing the panel causes the panel to be displayed. The NOSETMSG parameter is, in effect, only for the next panel. the NOSETMSG parameter is ignored on the CONTROL NONDISPL END Service.

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

**CANCEL**
Specifies that the dialog is to be terminated on an error resulting from a return code of 12 or higher from any service. A message is written to the ISPF log file, and a panel is displayed to describe the particular error situation. In batch mode, messages are written to the SYSTSPRT data set.

**RETURN**
Specifies that control is to be returned to the dialog on an error. System variables ZERR*xxxx*, as described in "Return Codes from Services" on page 12, contain the information for the message that describes the error. The message

is not written to the ISPF log file unless TRACE mode is in effect, and no error panel is displayed. If you want the dialog to abend with STAE you must specify CONTROL ERRORS RETURN, because specification of CONTROL ERRORS CANCEL nullifies the requested STAE.

**SPLIT**
Specifies the user's ability to enter split-screen mode, as defined by the ENABLE or DISABLE keyword.

**ENABLE**
Specifies that the user is to be allowed to enter split-screen mode. Split-screen mode is normally enabled. It is disabled only if explicitly requested by use of the CONTROL service. It remains disabled until explicitly re-enabled by the CONTROL service. Because SPLIT commands are not supported when ISPF is running in the batch environment, issuing CONTROL SPLIT ENABLE results in a severe error (return code 20).

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

**NOCMD**
Specifies that for the next displayed panel only, any command entered on the command line or through use of a function key is not to be honored. NOCMD is in effect for any redisplay of the panel.

**SUBTASK**
This option pertains to multi-task program dialogs that are invoked as TSO commands by the CMD interface of the SELECT service.

**PROTECT**
Specifies that ISPF is to establish an ESTAE routine to trap and ignore the abend that occurs when ISPF tries to POST a subtask that no longer exists.

If an abend does occur on a POST when the ESTAE protection is in effect, ISPF will return to a wait state until another service request occurs or the application terminates.

The new ESTAE will be in effect only around the POST, but once it is requested, it will be established each time ISPF is to POST the application, until the application cancels the protection request or the current SELECT level is terminated.

The scope of the ESTAE protection on the POST is strictly within the current SELECT level. It will not be automatically propagated to another SELECT level but must be requested again if it is to be used.

Any tables or other files that are opened by ISPF on behalf of the detached subtask (for example, by LIBDEF, table services, or file tailoring) will remain open until the application is terminated or the appropriate DM component service is used to close them. Thus, if such a subtask is to be restarted after being detached, it must have the logic to handle the situation when a table, or other file, it tries to open is already opened on entry to that routine.

Although both the parent task and subtask of a dialog can make DM component service calls, ISPF does not support asynchronous service requests. In other words, DM component service calls cannot be made while a service is in process for another caller.

Because the ESTAE protection is provided only on the POST of the DM component service caller, the following rule must be followed by the application:

- A subtask that can be detached while a DM component service that it invoked is in process cannot use any storage acquired under its TCB in the parameter list of a service call. That is, all parameters used in service calls must reside in storage that will not be released when the DETACH for the subtask is issued. Furthermore, any other resource which can be used by ISPF on behalf of the subtask must not be released while a DM component service is in process.

The parent task should acquire all the storage to be used by the subtask and pass it as a parameter on the ATTACH. Thus, all local variables to be used by the subtask would be declared in a DSECT and be based on the storage acquired by the parent task. This will prevent the possibility of an abend caused by an attempt by ISPF to access storage that was released and will still allow the subtask to use all DM component services.

**CLEAR**
Specifies that ESTAE protection on the POST of a subtask is to be terminated.

**buf-len**
Specifies a fullword fixed binary integer containing the length of "buffer".

**buffer**
Specifies a buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC call for a command procedure.

**TSOGUI**

**QUERY**
Gives the current status of the ISPF/TSO window:

**Return code = 0**
Either the user is not running ISPF GUI with TSO line mode support or TSOGUI is OFF. All TSO input and output is directed to the 3270 session.

**Return code = 1**
All TSO line mode output is displayed in the ISPF/TSO window and line mode input must be entered into the ISPF/TSO window's input field.

**OFF**
Specifies that the ISPF/TSO window is suspended and all full screen and line mode data appear in the 3270 window until CONTROL TSOGUI ON is issued.

**ON**
Specifies that the ISPF/TSO window is to be resumed and all TSO line mode output and input is directed to the ISPF/TSO window.

**Notes:**

1. CONTROL TSOGUI is ignored if you are not running ISPF GUI with TSO line mode support.
2. CONTROL TSOGUI defaults to ON during ISPF GUI session initialization.

**REFLIST**

**UPDATE**
Enable ISPF/PDF allocations to add entries to the data set and library reference lists.

**NOUPDATE**
Do not allow ISPF/PDF allocations to add entries to the data set and library reference lists.

**Notes:**

1. The CONTROL REFLIST command is used to enable or disable automatic updates to the reference lists. It is intended to be used around calls to ISPF/PDF services that normally cause entries in the reference lists. These services include EDIT, BROWSE, VIEW, and LMINIT.
2. When NOUPDATE is specified, the reference list is not updated, even if the user settings request updates. This is so programs can insure that they do not fill up the reference list with names that the user would never want to see, such as temporary or intermediate files.
3. The program invoking the CONTROL REFLIST NOUPDATE command to turn off reference list updates **must** specify CONTROL REFLIST UPDATE before it exits. It is recommended that you issue a CONTROL REFLIST NOUPDATE immediately before the service that would normally update the reference list (such as LMINIT, EDIT, or BROWSE) and issue a CONTROL REFLIST UPDATE immediately after the service returns.
4. There is only one CONTROL REFLIST setting for each logical screen (or split screen), and using this command can affect updates in the logical screen after the invoking program ends.

## Return Codes

The following return codes are possible:

**0** Normal completion.

**8** Split-screen mode already in effect. Applies only to a SPLIT DISABLE request. Split-screen mode remains enabled.

**20** Severe error.

## Example

Set the error processing mode to allow the dialog function to process return codes of 12 or higher.

```
ISPEXEC CONTROL ERRORS RETURN
```

or

Set the program variable BUFFER to contain:

```
CONTROL ERRORS RETURN
```

Set program variable BUFLEN to the length of the variable BUFFER. Issue the following:

```
CALL ISPEXEC (BUFLEN, BUFFER);
```

or alternately

```
CALL ISPLINK ('CONTROL ','ERRORS  ','RETURN  ');
```

# DISPLAY – Display Panels and Messages

The DISPLAY service retrieves a panel definition, performs any pre-display processing specified on the panel definition, initializes variable panel fields from the corresponding dialog variables, and displays the panel on the screen. A message can optionally be displayed with the panel. If the optional message is to be displayed in a message pop-up window, the position of the message pop-up window can be indicated by the MSGLOC parameter.

**Note:** When running in GUI mode, the MSGLOC parameter is ignored.

After the panel has been displayed, you can enter information and press the Enter key. All input fields are automatically stored into dialog variables of the same name, and the )PROC section of the panel definition is then processed. If any condition occurs that causes a message to be displayed (verification failure, MSG=value condition in a TRANS, or explicit setting of .MSG), processing continues to the )HELP or )END section. The )REINIT section is then processed if it is present. The panel is then redisplayed with the first, or only, message that was encountered.

When the user presses the Enter key again, all input fields are stored and the )PROC section is again processed. This sequence continues until the entire )PROC section has been processed without any message conditions being encountered. The panel display service finally returns, with a return code of 0, to the dialog function that invoked it.

Alternatively, when a panel is displayed, the user can enter a CANCEL, END, EXIT, or RETURN command. If the input fields are not in a scrollable area, they are stored and the )PROC section is processed. In scrollable areas, only the input fields that have been displayed will be stored. No messages are displayed, even if a MSG condition is encountered. The panel display service then returns to the dialog function with a return code of 8.

## Command Invocation Format

```
ISPEXEC  DISPLAY  [PANEL(panel-name)]

                  [MSG(message-id)]

                  [CURSOR(cursor-field-name)]

                  [CSRPOS(cursor-position)]

                  [COMMAND(stack-buffer-name)]

                  [RETBUFFR(ret-buffer-name)]

                  [RETLGTH(ret-length-name)]

                  [MSGLOC(message-field-name)]
```

## Call Invocation Format

```
CALL  ISPEXEC (buf-len, buffer);
OR

CALL  ISPLINK ('DISPLAY ' [,panel-name]

                          [,message-id]
```

```
[,cursor-field-name]

[,cursor-position]

[,stack-buffer-name]

[,ret-buffer-name]

[,ret-length-name]

[,message-field-name]
```

## Parameters

**panel-name**
Specifies the name of the panel to be displayed.

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

**cursor-field-name**
Specifies the name of the field where the cursor is to be placed.

If cursor-position is specified both by this parameter and by setting the control variable .CURSOR in the )INIT or )REINIT section of the panel being displayed, the value in .CURSOR overrides this parameter.

**cursor-position**
Specifies the character position within the field where the cursor is to be placed. This position applies regardless of whether the initial cursor placement was specified in the CURSOR calling sequence parameter, the .CURSOR control variable in the )INIT or )REINIT section of a panel, or is the result of default cursor placement. If cursor-position is not specified or is not within the field, the default is 1.

If cursor-position is specified both by this parameter and by setting the control variable .CSRPOS in the )INIT or )REINIT section of the panel being displayed, the value in .CSRPOS overrides this parameter.

**stack-buffer-name**
Specifies the name of a variable containing the chain of commands passed by the dialog to ISPF for execution. The maximum length of the actual command chain within this variable is 255.

**ret-buffer-name**
Specifies the name of a variable in which the unprocessed portion of the command chain is stored should an error occur before the complete chain is processed. This includes the command being processed when the error is detected.

**ret-length-name**
Specifies the name of a variable in which the length of the unprocessed portion of the command chain is stored should an error occur before the complete chain is processed. This includes the command being processed when the error was detected.

**message-field-name**
Used to position the message pop-up window. If the application specifies this parameter, the dialog manager positions the message pop-up relative to the named field.

If this parameter is omitted and a message is displayed in a message pop-up window, the window is displayed at the bottom of the logical screen or below the active ADDPOP pop-up window if one exists.

For compatibility with later versions, this parameter should be specified only when the message will display in a pop-up window.

**Note:** When running in GUI mode, this parameter is ignored.

**buf-len**
Specifies a fullword fixed binary integer containing the length of "buffer".

**buffer**
Specifies a buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC call for a command procedure.

All of the parameters are optional. The panel-name and message-id parameters are processed as follows:

- If panel-name is not specified, an error occurs unless a previous panel was displayed at the same nesting level.
- If panel-name is specified and message-id is not specified, the panel is retrieved, the )INIT section, if it exists, is processed, and the panel is displayed without a message.
- If panel-name and message-id are both specified, the panel is retrieved, the )INIT section, if it exists, is processed, and the panel is displayed with the specified message, which is typically a prompt or confirmation message.
- If panel-name is not specified and message-id is specified, the )REINIT section, if it exists, is processed and the current panel is overlaid with a message, which is typically an error message.
- If neither panel-name nor message-ID is specified, the )REINIT section, if it exists, is processed and the current panel is redisplayed without a message. Use the CONTROL service to save and restore the environment when a display series, in which the panel-name is not specified, is to be interrupted by another DISPLAY, TBDISPL, BROWSE, or EDIT operation.
- When a panel is displayed before invoking EDIT/VIEW, invoking the DISPLAY service without a panel name from within the EDIT/VIEW service can produce unpredictable results. The DISPLAY environment might be altered by the EDIT/VIEW service. Do not expect the DISPLAY environment that existed before invoking the EDIT/VIEW service to remain unchanged.

In the first two situations, processing of the panel definition proceeds normally, through the )INIT section, before display of the panel. If .MSG, .CURSOR, or .CSRPOS is set in the )INIT section, that setting overrides an initial message or cursor placement passed by the calling sequence parameters.

In the third and fourth situations, processing of the )INIT section is bypassed, and there is no automatic initialization of variables in the panel body, nor in the attribute section. However, the )REINIT section is processed. The )REINIT section provides for specified variables or attributes to be reset before a redisplay. Typically, the )REINIT section contains:

- Field attribute overrides, specified with the .ATTR control variable.
- Changes to displayed panel fields, specified in assignment statements and the REFRESH statement.

Each time the DISPLAY service is invoked, the )PROC section of the panel is processed *after* the terminal user enters a response to the display. Therefore, it is recommended that all reinitialization logic be placed in the )REINIT section, rather than at the end of the )PROC section.

## Using the COMMAND Option

The COMMAND option allows a dialog to pass a chain of commands in the variable specified by *stack-buffer-name* to ISPF for execution. The panel specified on the DISPLAY service request is processed in CONTROL NONDISPL ENTER mode. In addition, when ENTER is simulated by ISPF, the command chain is executed as though it were either entered on the command line of the panel by the user or entered through a function key. When the command chain is exhausted or one of the commands cannot be found in the active set of command tables, processing terminates and control returns directly to the dialog that issued the DISPLAY COMMAND call, except for those specific error conditions described below.

If the DISPLAY COMMAND service returns an error, the function pool variable specified by *ret-buffer-name* contains the unexecuted portion of the command chain, starting with the first command that cannot be found in the active set of command tables. If all commands have been processed, the variable will be blank.

The *ret-length-name* variable contains the length of the string in the *ret-buffer-name* variable. If all commands have been processed, either by the DISPLAY COMMAND dialog or a dialog invoked to process a command in the stack, the length will be zero.

One or more of the commands in the command chain can be processed by the dialogs initiated from previous valid commands in the chain. Processing those commands will be the same as if the command chain had been entered from the primary input field of the dialog's panel. Errors encountered because of these commands must be handled by the dialog.

There are two cases in which the panel specified on the original DISPLAY COMMAND service request is displayed:

- First, when a command error, which results in a message such as "command NOT ACTIVE" or "INVALID command PARM" occurs, the current panel is presented, along with the corresponding message, in normal DISPLAY mode. This occurs even if the current panel is the panel specified on the original DISPLAY COMMAND call. To return to the dialog, the user has to enter the END command or an equivalent.
- The second case is when a SPLIT or SPLITV command is executed from the stack as input from the original panel. That panel is displayed on part of the physical screen. Control is not immediately returned to the dialog if execution of the command chain results in SPLIT, SPLITV, or SWAP. In this case the user must re-activate the original screen, such as enter SWAP, to give the dialog control once again.

  **Notes:**

  1. If the panel displayed with the COMMAND option has its primary input field initialized to a nonblank value, that string will *not* be concatenated to the end of the command chain.
  2. A CONTROL NOCMD pending at the time the DISPLAY COMMAND service is issued will be cancelled.

3. ISPF does not support the jump function when the COMMAND option is being executed. ISPF deletes any equal signs (=) preceding a command, but the command remains in the stack.

## Return Codes

The following return codes are possible:

**0** Normal completion.

For the COMMAND option, the *ret-buffer-name* is set to blanks and the *ret-length-name* is set to zero. Passing an empty command chain buffer also results in a normal completion.

**4** One or more commands in the stack could not be found in the active set of command tables.

**8** User requested termination using the END or RETURN command. If CANCEL and EXIT are requested from a panel displayed using the DISPLAY service call and the panel was defined with the dialog tag language (DTL), the dialog manager returns the command in ZVERB and sets a return code of 8 from the display screen.

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

**16** Truncation or translation error in storing defined variables.

**20** Severe error.

## Example 1

Panel definition XYZ specifies display of variables AAA and KLM as input fields. Using this definition, invoke services to display these variables at the terminal and superimpose, on line 1, the short form text of message number ABCX013. Place the cursor, on the display, at the beginning of input field KLM, ready for entry of data by the person at the terminal.

```
ISPEXEC DISPLAY PANEL(XYZ) MSG(ABCX013) CURSOR(KLM)
```

***or*** Set the program variable BUFFER to contain:

```
DISPLAY PANEL(XYZ) MSG(ABCX013) CURSOR(KLM)
```

Set program variable BUFLEN to the length of the variable BUFFER. Issue the following:

```
CALL ISPEXEC (BUFLEN, BUFFER);
```

or alternately

```
CALL ISPLINK ('DISPLAY ','XYZ ','ABCX013 ','KLM ');
```

## Example 2

Pass a command stack to ISPF to cause:

- The display screen to split horizontally at the line on which the cursor was positioned when the DISPLAY COMMAND was issued
- Control to return to the top screen (SWAP)
- A command, CHECK, to be issued on the top screen (assume CHECK does not exist in the active set of command tables).

Function pool variable STACKA contains the command string:

```
SPLIT;SWAP;CHECK
```

Issue:

```
ISPEXEC DISPLAY PANEL(PANA) COMMAND(STACKA) RETBUFFR(BUFFA) RETLGTH(LGTHA)
```

or alternately

```
CALL ISPLINK ('DISPLAY ','PANA ',' ',' ',' ','STACKA ','BUFFA ','LGTHA ');
```

Because ISPF cannot find the command CHECK in a command table, processing of the command stack terminates at that point. ISPF places the unprocessed command, CHECK, in variable BUFFA, and sets variable LGTHA to 5. The DISPLAY service terminates with a return code of 4.

## Example 3

Pass a command stack to ISPF to cause the:

- Function key definition panel, containing the INVALID COMMAND message, to display
- Primary input field (PIF) of the panel to be set to CHECK
- Alarm to sound.

Function pool variable STACKA contains the command string:

```
KEYS;CHECK
```

Issue:

```
ISPEXEC DISPLAY PANEL(PANA) COMMAND(STACKA) RETBUFFR(BUFFA) RETLGTH(LGTHA)
```

or alternately

```
CALL ISPLINK ('DISPLAY ','PANA ',' ',' ',' ','STACKA ','BUFFA ','LGTHA ');
```

ISPF cannot find the command CHECK in any active command table. Because the unidentified command error is encountered by the KEYS dialog, rather than the DISPLAY service, it is the responsibility of the dialog to process the error. In this case, the KEYS dialog displays a message indicating that CHECK was not found. Upon return from the KEYS dialog, the DISPLAY service sets the return buffer, BUFFA, to blanks, sets variable LGTHA to 0, and terminates with a return code of 0.

## Example 4

Pass a command stack to ISPF to cause:

- PANA, containing the INVALID PFSHOW PARM message, to display
- The alarm to sound.

Function pool variable STACKA contains the command:

```
PFSHOW COLOR
```

Issue:

```
ISPEXEC DISPLAY PANEL(PANA) COMMAND(STACKA) RETBUFFR(BUFFA) RETLGTH(LGTHA)
```

or alternately

```
CALL ISPLINK ('DISPLAY ','PANA ',' ',' ',' ','STACKA ','BUFFA ','LGTHA ');
```

COLOR is not a valid parameter on the PFSHOW command. Therefore, PANA displays. In this case, the user exits from PANA normally (ENTER, END, or RETURN). The DISPLAY service returns control to the dialog with a return code of 0.

## Example 5

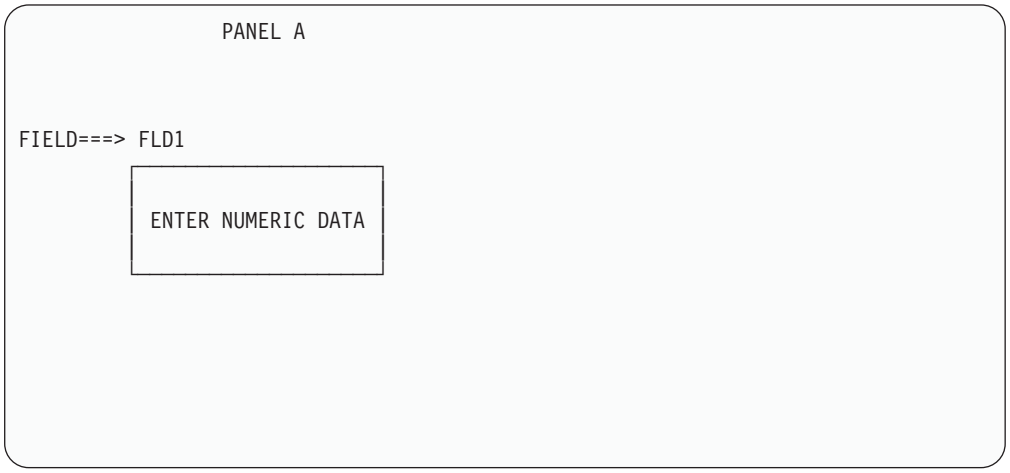
The following DISPLAY request displays message TSTA110 in a message pop-up window that permits interaction with the underlying panel. The message pop-up window is positioned relative to the field FLD1.

```
PROC 0
ISPEXEC DISPLAY PANEL(A) MSG(TSTA110) MSGLOC(FLD1)
```

Using the following message definition for TSTA110

```
TSTA110 .WINDOW=NORESP
'ENTER NUMERIC DATA'
```

Results in:




# DSINFO – Data Set Information Dialog Service

The DSINFO service returns assorted information about a particular data set in dialog variables in the function pool. The information returned is the same as that displayed when you use ISPF Option 3.2 or Option 3.4 commands. Additionally, DSINFO returns the unformatted DSCB format1. DSINFO does *not* require an LMINIT to be performed first.

## Command Invocation Format

```
ISPEXEC  DSINFO DATASET(dsname)
              [VOLUME(serial)]
```

## Call Invocation Format

```
CALL  ISPEXEC (buf-len, buffer);
OR

CALL  ISPLINK ('DSINFO  ' [,dsname]

                          [,serial]
```

## Parameters

**dsname** Specifies the data set name, in TSO syntax, of the data set that you want information about.

**serial** Specifies the serial number of the volume on which the data set can be found. This is only required if the data set is uncataloged.

**buf-len** Specifies a fullword fixed binary integer containing the length of "buffer".

**buffer** Specifies a buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC call for a command procedure.

The DSINFO service saves the following dialog variables in the function pool:

| Variable | Information | Type | Length |
|---|---|---|---|
| ZDSVOL | First or only volume | Character | 6 |
| ZDS#VOLS | Number of volumes | Character | 2 |
| ZDSDEVT | Device type | Character | 8 |
| ZDSORG | Data set organization | Character | 8 |
| ZDSRF | Record format | Character | 6 |
| ZDSLREC | Logical record length | Character | 7 |
| ZDSBLK | Block size | Character | 6 |
| ZDSSPC | Primary space units | Character | 8 |
| ZDS1EX | Primary space allocation | Character | 13 |
| ZDS2SPC | Secondary space units | Character | 8 |
| ZDS2EX | Secondary space allocation | Character | 13 |
| ZDSDSNT | Data set name type | Character | 8 |
| ZDSSEQ | Compressible YES/NO | Character | 4 |
| ZDSCDATE | Creation date (National format) | Character | 10 |
| ZDSXDATE | Expiration date (National format) | Character | 10 |
| ZDSRDATE | Referenced date (National format) | Character | 10 |
| ZDSTOTA | Allocated space units | Character | 13 |
| ZDSTOTU | Used space units | Character | 13 |
| ZDSEXTA | Allocated extents | Character | 7 |
| ZDSEXTU | Used extents | Character | 7 |
| ZDSDIRA | Allocated directory blocks | Character | 6 |
| ZDSDIRU | Used directory blocks | Character | 8 |
| ZDSDIR | PDSE directory blocks | Character | 8 |
| ZDS#MEM | Number of members | Character | 13 |
| ZDSPAGU | Pages used (PDSE) | Character | 13 |
| ZDSPERU | Percent used (PDSE) | Character | 13 |
| ZDSMC | Management class | Character | 8 |
| ZDSSC | Storage class | Character | 8 |
| ZDSDC | Data class | Character | 8 |
| ZDSCB1 | Format 1 data control block | Character | 96 |
| ZDSVTAB | Volume table (contains all of the volume names for a multivolume set) | Character | 354 |

## Return Codes

The following return codes are possible:

**0** Normal completion.

**8** User requested data set was not found.

**12** One of the following:

- FAMS error
- Obtain error
- Error obtaining directory information

**20** Severe error.

## Example

The following example shows an invocation of DSINFO to obtain information about a cataloged data set.

Command Invocation

```
ISPEXEC DSINFO DATASET(DSNAME)
```

Call Invocation

```
CALL ISPLINK('DSINFO  ',DSNAME);
```

***or*** Set the program variable BUFFER to contain:

```
BUFFER = 'DSINFO DATASET(DSNAME)';
```

Set program variable BUFLEN to the length of the variable BUFFER. Issue the following:

```
CALL ISPEXEC (BUFLEN, BUFFER);
```

# EDIF – Edit Interface

The Edit Interface (EDIF) service provides edit functions for data accessed through dialog-supplied I/O routines. The invoking dialog must perform all environment-dependent functions such as file allocation, opening, reading, writing, closing, and freeing. The dialog is also responsible for any Enqueue/Dequeue serialization that is required. With the dialog providing the I/O routines, EDIF allows you to:

- Edit data other than partitioned data sets or sequential files such as subsystem data, and in-storage data.
- Do preprocessing and post-processing of the data being edited.

The invoking dialog must provide addresses to routines that:

- Read the data sequentially from beginning to end, returning to Edit one record on each invocation.
- Write the data sequentially from beginning to end, accepting one record from Edit on each invocation.
- Perform processing for the MOVE, COPY, CREATE, REPLACE, and EDIT primary commands. If this routine is not specified, ISPF processes these commands.

All addresses must be 31-bit addresses, and the routines must have an addressing mode (AMODE) of 31.

When an Edit session is operating in recovery mode, a record of your interactions is automatically recorded in an PDF-controlled data set. Following a system failure, you can use the record to recover the data you were editing.

**Note:** Dialogs that invoke the EDIF service may invoke the EDIREC service first to determine if a pending recovery condition exists.

A dialog using EDIF can place data into the ZEIUSER dialog variable in the shared pool. When the system initializes the recovery data set, the system also saves the data in ZEIUSER in the Edit recovery table as an extension variable. This is done if RECOVERY is ON when first entering Edit or after you use the SAVE command. This data is then made available in dialog variable ZEIUSER at the time edit recovery is processed.

## Command Invocation Format

You cannot use command procedures to invoke this service.

## Call Invocation Format

The format for invoking EDIF can be different depending on whether you want to process a pending edit recovery. If you do not want to process a pending edit recovery, the format is:

```
CALL ISPLINK ('EDIF    ', [data-name] ,profile-name

                        ,rec-format ,rec-len

                        ,read-routine, write-routine

                        ,[cmd-routine] ,[dialog-data]

                        ,[edit-len] ,[panel-name]

                        ,[macro-name] ,[format-name]

                        ,['YES     '|'NO      ']

                        ,['YES     '|'NO      ']

                        ,[parm–var] );
```

You must use the EDIF service to recover data edited in a previous EDIF session. You must invoke the EDIREC service first to see if a recovery is pending. If you want to process a pending recovery, use the following format for EDIF, specifying YES for the recovery-request parameter.

```
CALL ISPLINK ('EDIF    ' ,[data-name] ,' '

                        ,[rec-format] ,[rec-len]

                        ,read-routine ,write-routine

                        ,[cmd-routine] ,[dialog-data]

                        ,' ' ,' ' ,' ' ,' ' ,' ' ,' '
                        ,'YES     ');
```

## Parameters

**data-name**
This parameter allows you to specify a data name for the source data to be

edited. This name appears in the title line of the default Edit panel. It is also the target data name for an edit recovery table entry when edit recovery is active. This name must not have any embedded blanks, and its maximum length is 54 characters. This name is stored in ZDSNT in the function pool.

**profile-name**
The name of the edit profile to be used. This parameter is required when recovery-request = NO or is not specified; otherwise, it is not allowed.

**rec-format**
The record format: F - fixed, V - variable. This parameter is required when recovery-request = NO or is not specified; otherwise, it is optional, but it must be the same record format that was specified when recovery was initiated for the data.

**rec-len**
The record length, in bytes. It must be a positive numeric value between 10 and 32760, inclusive. For variable record format, this is the maximum record length. This parameter is required when recovery-request = NO or is not specified; otherwise, it is optional, but it must be the same record length that was specified when recovery was initiated for the data.

**read-routine**
A fullword address indicating the entry point of a dialog-supplied read routine (required). See “Read Routine” on page 50 for more information about this parameter.

**write-routine**
A fullword address indicating the entry point of a dialog-supplied write routine (required). See “Write Routine” on page 50 for more information about this parameter.

**cmd-routine**
A fullword address indicating the entry point of a dialog-supplied routine that processes the MOVE, COPY, CREATE, REPLACE, and EDIT primary commands. See “Command Routine” on page 51 for more information about this parameter. If this parameter is not specified, ISPF processes these commands.

**dialog-data**
A fullword address indicating the beginning of a dialog data area. This address is passed to the dialog-supplied routines. If no address is supplied, zeros are passed to the dialog routines. This data area provides a communication area for the dialog.

**edit-len**
The length, in bytes, of the data to be displayed for editing. This parameter indicates that the data records should be considered to have a length shorter than rec-len during editing. Thus, the dialog may include data in the record that is not accessible for editing.

Edit-len must be a numeric value between 10 and 32760, inclusive, and must be less than or equal to parameter rec-len. Rec-len is the default. If the edit-len parameter is specified, the data that is not displayed are the bytes from (edit-len +1) to rec-len. That means the inaccessible record data is at the end of the record.

The edit-len parameter is optional when recovery-request = NO or is not specified; otherwise, it is not allowed. The edit-len parameter is not allowed when format-name is specified.

**panel-name**
The name of the panel to use for displaying the data. This parameter is optional when recovery-request = NO or is not specified; otherwise, it is not allowed. The default is the standard Edit data display panel. Refer to *ISPF Planning and Customizing* for information about developing a customized panel.

**macro-name**
The name of the initial macro to be executed. This parameter is optional when recovery-request = NO or is not specified; otherwise, it is not allowed. The default is no initial macro. Refer to *ISPF Edit and Edit Macros* for more information on macros.

**format-name**
The name of the format to be used to reformat the data. This parameter is optional when recovery-request = NO or is not specified; otherwise, it is not allowed. The default is no format. This parameter is provided to support the IBM 5550 terminal using the Double-Byte Character Set (DBCS). This parameter is not allowed when the edit-len parameter is specified.

**YES | NO (mixed-mode)**
Specifies whether the data is treated as mixed-mode DBCS data. This parameter is optional when recovery-request = NO or is not specified; otherwise, it is not allowed. If YES is specified, the EDIF service treats the data as mixed-mode DBCS data. If NO (the default) is specified, the data is treated as EBCDIC (single-byte) data. This parameter is provided to support the IBM 5550 terminal using the Double-Byte Character Set (DBCS).

**YES | NO (recovery-request)**
Specifies whether to process a pending edit recovery that was being edited with the EDIF service when a system failure occurred. If YES is specified, the edit recovery should proceed. This function is similar to the EDREC service with the PROCESS option. If YES is specified to process the edit recovery, you must specify the read routine and write routine, but you must not specify profile name, edit-len, panel-name, macro-name, format-name and mixed-mode. If NO (the default) is specified, no edit recovery is processed; EDIF edits the specified data.

**parm-var**
The name of an ISPF variable that contains parameters which are to be passed to the initial macro specified by *macro-name*. The variable value must not exceed 200 bytes in length. If no macro name is specified, parm-var must be blank or not specified.

## Dialog-Supplied Routines

All dialog-supplied routines are invoked using standard linkage. All addresses must be 31-bit addresses, and the addressing mode (AMODE) of the routines must be AMODE=31.

An EDIF read or write routine must have an assembler interface to be used in a call to EDIF.

The dialog-supplied read, write, and command processing routines are called directly by ISPF at the same task level (TCB) that displays ISPF screens. If you need to ensure that your program runs at the same task level as the routines, use the SELECT PGM( ) service to start your program. This may be a factor if your program expects to create or share data spaces or other task-specific resources between the main program and the read, write, or command routines.

**Note:** The read, write, and command exits can be written in languages that use the Language Environment (LE) run-time environment. However, a mixture of LE-conforming main dialog code and service routine code is not supported. Dialogs and service routines must either all be LE-conforming or all be LE-nonconforming.

## Read Routine

EDIF calls the read routine repeatedly to obtain all of the data records to be edited at the beginning of the Edit session. This routine is also called to obtain data records for the MOVE and COPY commands when the dialog is handling the processing for these commands. The dialog-supplied read routine is invoked with the following parameters:

- Fullword pointer to record data read (output from read routine)
- Fullword fixed binary data length of record read if rec-format is V
- Fullword fixed binary request code. Request settings are as follows:

  0 Read next record
  1 First read request
- Fullword dialog data area address.

## Write Routine

EDIF calls the write routine repeatedly to write the data records, for example, whenever data changes are to be saved with the SAVE, END, and RETURN commands, and the jump function. EDIF also calls the write routine to write data records for the CREATE and REPLACE commands when the dialog is handling the processing for these commands. The write routine is given flags that indicate the source and change status for each record.

The dialog-supplied write routine is invoked with the following parameters:

- Fullword pointer to record data to be written
- Fullword fixed binary data length of record to be written if rec-format is V. This is the length of the nonblank portion of the record. The entire record with trailing blanks up to the maximum rec-len is available.
- Fullword of source and change bits for the record. The bit representation is as follows:

Source bits:

| | |
|---|---|
| 1 = original record | |
| 2 = internal move | (Move line command) |
| 3 = internal copy/repeat | (Copy/Repeat line commands) |
| 4 = external move | (MOVE primary command) |
| 5 = external copy | (COPY primary command) |
| 6 = text inserted | (TE line command) |
| 7 = typed inserted | (Insert line command) |

Change bits:

| | |
|---|---|
| 8 = record changed | (global bit; set for all changes) |
| 9 = data overtyped | |
| 10 = change command | (CHANGE primary command) |
| or overlay change | (Overlay line command) |
| 11 = columns shifted | ((,((,),)) line commands) |
| 12 = data shifted | (<,<<,>,>> line commands) |
| 13 = text change | (TE, TF, TS line commands) |
| 14 = record renumbered | |
| 15-32 = unused | |

Multiple bits may be set on, indicating that more than one modification has occurred for the record. For example, a data record that is inserted by using the Insert line command and is later included in a text flow operation would have bits 7 (typed inserted), 8 (change), 9 (data overtyped) and 13 (text changed) turned on.

Records read in for the initial display are flagged as original records. Whenever there is hidden data, the inaccessible portion of inserted records contains blanks. Records are copied in their entirety; that is, including both the visible and hidden portions of the data. Deleted records are not presented to the write routine.

- Fullword fixed binary request code. Request settings are as follows:

  0 Write the next record
  1 First write request
  2 Last write request (final data record provided)
  3 First and last write request (only one data record)
  4 No data records to write (all records have been deleted)

- Fullword dialog data area address.

## Command Routine

The dialog-supplied command routine, when specified, processes the MOVE, COPY, CREATE, REPLACE, and EDIT primary commands. The command routine is invoked with the following parameters:

- Fullword fixed binary function code word. Decimal values of function settings are as follows:

  1*n* Move
  2*n* Copy
  3*n* Create
  4*n* Replace
  5*n* Recursive edit

  where *n* is 0 (beginning of function), 1 (successful completion), or 2 (unsuccessful completion). This *n* value will always be 0 for a recursive Edit function; that is, the Edit request code will be 50.

- Fullword dialog data area address.

To access parameters that can follow the command, the command routine must access the ZCMD dialog variable from the SHARED variable pool.

For a MOVE, COPY, CREATE, or REPLACE, the command routine initiates the processing for the requested function. When the return code from the command routine is zero, EDIF calls the read or write routine to transfer the data. After the read or write is completed, the command routine is called once more to handle any termination processing that may be required for the requested function. For example, the MOVE function would need to delete the data that was moved.

For the EDIT command, the command routine must perform all processing required to effect the desired results for the purposes of the dialog. For example, the dialog can consider the EDIT command to be an invalid command. The command routine is called only once for each EDIT command.

## Return Codes

When a dialog routine terminates with a return code (12 or higher or an unexpected return code), the dialog can issue a SETMSG to generate a message on the next panel display. If the dialog does not set a message, the EDIF service will issue a default message.

## Read Routine

**0** Normal completion.

**8** End of data records (no data record returned).

**16** Read error. If a read error is encountered when the system builds the initial edit display, the EDIF service terminates with a return code of 20. Otherwise, the edit data is redisplayed.

**20** Severe error. (The EDIF service terminates immediately with a return code of 20.)

## Write Routine Return Codes

**0** Normal completion.

**16** Output error, return to Edit mode.

**20** Severe error. (The EDIF service terminates immediately with a return code of 20.)

## Command Routine Return Codes

**0** Normal completion.

**4** ISPF should process the requested function.

**12** Command deferred; retain the command on the Command line. Edit data is redisplayed.

**20** Severe error. (The EDIF service terminates immediately with a return code of 20.)

## EDIF Service Return Codes

**0** Normal completion, data saved.

**4** Normal completion, data not saved.

**16** Unexpected return code received from a dialog-supplied routine. When an unexpected return code is received, the EDIF service terminates immediately with a return code of 16.

**20** Severe error; unable to continue.

After the Edit session has been terminated, control is returned to the invoking dialog with a return code indicating the completion status.

## Example

This example invokes the EDIF service to edit data called EDIFDSN, which has a fixed-record format with a record length of 80 characters. An edit profile (EDIFPROF), read routine (RDRTN), write routine (WRRTN), and command routine (CMDRTN) are supplied, as is a dialog data area (MYDATA).

### Call Invocation

```
CALL ISPLINK ('EDIF    ','EDIFDSN ','EDIFPROF ','F ',80,
     RDRTN,WRRTN,CMDRTN,MYDATA);
```

For a more complete example of using EDIF, including dialog-supplied I/O routines and source code, see the *ISPF Dialog Developer's Guide and Reference*.

## EDIREC - Initialize Edit Recovery

The EDIREC service initializes an edit recovery table (ISREIRT) for use by the EDIF service and determines whether recovery from the EDIF service is pending. EDIREC also allows you to cancel or defer the recovery of data modifications.

### Command Invocation Format

You cannot use command procedures to invoke this service.

### Call Invocation Format

```
CALL ISPLINK ('EDIREC  ', {'INIT    '[,command-name]}
                          {'QUERY   '}

                          {'CANCEL  '}

                          {'DEFER   '});
```

### Parameters

**INIT**
The edit recovery table for EDIF should be initialized in the user profile library if one does not already exist for the current application.

**command-name**
A command procedure (CLIST or REXX exec) or a program written as a command that initializes the EDIF table. If this parameter is omitted, the INIT option invokes an PDF-supplied command (ISREIRTI) that creates an 8-row EDIF recovery table that permits eight levels of concurrent Edit sessions with recovery active. The Edit sessions may be due to recursion or split-screen usage.

If you specify an application-supplied command with the INIT option, you should pattern the command after ISREIRTI. It can create a different number of rows, use a different naming convention for the backup data sets, or specify "keep" (instead of "delete") as the backup data set disposition. The format of the EDIF recovery table must be the same as that specified in ISREIRTI.

**QUERY**
The EDIF recovery table should be searched for a pending recovery. When the QUERY option is specified, EDIREC scans the EDIF recovery table for an entry containing a recovery pending condition. If an entry is found (return code 4), the dialog must then call EDIF with (recovery-request = YES) to recover the data, or call EDIREC with the CANCEL or DEFER option to dispose of the pending recovery condition.

The following variables are stored in the dialog function pool when EDIREC is called with the QUERY option and recovery is pending (return code 4):
- ZEIBDSN - Backup data name
- ZEITDSN - Target data name

- ZEIROW - Row number of entry in the recovery table.

The dialog can check the variables and use them to display information to the user. The dialog must not change them. If EDIREC QUERY indicates that recovery is not pending, the above variables are not meaningful.

ZEIUSER is an extension variable in the EDIF recovery table that is provided to contain user data. Whatever data is in dialog variable ZEIUSER in the shared pool is saved to the ZEIUSER variable in the EDIF recovery table when the recovery data set is initialized. This is done if RECOVERY is ON when first entering Edit or after using the SAVE command.

When EDIREC is called with the QUERY option and the return code is 4, indicating that recovery is pending, the data is read out of ZEIUSER in the table and returned to ZEIUSER in the shared and function pools. If recovery is not pending, this variable is not meaningful.

**CANCEL**
Cancellation of edit recovery. The backup data set is erased and the corresponding entry in the EDIF recovery table is freed.

**DEFER**
Edit recovery is to be deferred. Recovery is cancelled, but the backup data set is saved so that recovery can be processed at another time.

## Return Codes

The following return codes are possible:

**0** Normal completion.
- INIT - EDIF recovery table was successfully created.
- QUERY - Recovery is not pending.

**4** Normal completion.
- INIT - EDIF recovery table already exists for current application.
- QUERY - Entry found in EDIF recovery table (recovery is pending).

**20** Severe error; unable to continue.

## Example

This example invokes the EDIREC service to initialize the EDIF recovery table by using the command procedure USRCMD.

```
CALL ISPLINK('EDIREC  ','INIT    ','USRCMD ');
```

# EDIT – Edit a Data Set

The EDIT service provides an interface to the PDF editor and bypasses the display of the Edit Entry Panel. The EDIT interface allows you to use a customized panel for displaying data (use panel ISREFR01 as a model when creating your panel), and lets you specify the initial macro and the edit profile to be used. You can use EDIT to look at any ISPF library, concatenation of ISPF libraries, or data set that can be allocated by using the LMINIT service. You can use the EDIT service recursively, either through nested dialogs or by entering an EDIT command while editing. *ISPF Edit and Edit Macros* contains a complete description of the editor.

When EDIT is operating in recovery mode, an audit trail of your interactions is automatically recorded in a PDF-controlled data set. Following a system failure, you can use the audit trail to recover the data you were editing.

**Notes:**

1. Dialogs that invoke the EDIT service may invoke the EDREC service first to start edit recovery, because the EDIT service does not do edit recovery.
2. The EDIT service might alter the DISPLAY environment. Do not expect the DISPLAY environment that existed before invoking the EDIT service to remain unchanged.
3. The EDIT service cannot be issued by a PL/I main program that also uses subtasking.
4. When designing applications that will use the EDIT service, be aware that you cannot run EDIT in a pop-up window.

A dialog using EDIT can place data into the ZEDUSER dialog variable in the shared pool. The data in ZEDUSER is saved in the edit recovery table as an extension variable when the recovery data set is initialized. This is done if RECOVERY is ON when first entering Edit or after using the SAVE command. This data is then made available in dialog variable ZEDUSER at the time edit recovery is processed.

You can use EDIT to edit workstation files on the host and host data sets on the workstation. This function is called *distributed editing*.

The ZWSWFN variable is the workstation working file name that is generated by ISPF. The variable ZLRECL is the LRECL of the host data set being edited. Both can be used in the workstation EDIT parameters field. ISPF interprets any string that starts with an ampersand (&) as a system variable and evaluates it before passing to the workstation command. Strings that do not start with an ampersand are passed as is. The EDIT service edits the host data set or workstation file on the workstation, using the workstation editor configured in the ISPF Workstation Tool Integration Program. For more information about the Integration program, refer to the *ISPF User's Guide Volume II*.

**Restrictions when using distributed editing:**

1. This type of edit does not support uncataloged data sets.
2. Take care when uploading to the host files that contain extended ASCII characters. For example, uploading a Microsoft Word *.DOC file to the host, using ISPF Edit to edit it, then saving it, can result in a corrupted file.
3. Some characters that are normally valid for a directory or file name might not be supported when using distributed editing. For example, the caret symbol ( ^, or shift-6 in GUI mode) is supported as a valid element of a directory or file name on Windows and OS/2®. However, it is ignored by the TCP/IP File Transfer Protocol for OS/2.
4. Some workstation editors might not work for multiple modal invocations. After the first invocation of some editors, any subsequent invocation of the same editor passes control to the first invocation when a command is issued in the second invocation, and the second invocation shuts down. For example, because of the way the EPM editor works, the second invocation passes control to the first and ends, returning a *command complete* indication.

Besides the Edit program name, you can specify batch commands in the Workstation Edit field. If you have a file transferred to the workstation that you wish to do some work on besides Edit, you can do that in the beginning of the batch file before invoking the editor. Depending on the parameters passed to the batch command, you can also have conditional logic to perform other functions as well.

## Command Invocation Format

```
ISPEXEC  EDIT  DATASET(dsname)  [VOLUME(serial)]
                                [PASSWORD(pswd-value)]
                                [PANEL(panel-name)]
                                [MACRO(macro-name)]
                                [PROFILE(profile-name)]
                                [FORMAT(format-name)]
                                [MIXED(YES|NO)]
                                [LOCK(YES|NO)]
                                [CONFIRM(YES|NO)]
                                [WS(YES|NO)]
                                [PRESERVE]
                                [PARM(parm-var)]
OR

ISPEXEC  EDIT  DATAID(dsname)   [MEMBER(member-name)]
                                [PANEL(panel-name)]
                                [MACRO(macro-name)]
                                [PROFILE(profile-name)]
                                [FORMAT(format-name)]
                                [MIXED(YES|NO)]
                                [LOCK(YES|NO)]
                                [CONFIRM(YES|NO)]
                                [WS(YES|NO)]
                                [PRESERVE]
                                [PARM(parm-var)]
OR

ISPEXEC EDIT WSFN(ws-filename)  [ PANEL(panel-name)]
                                [MACRO(macro-name)]
                                [PROFILE(profile-name)]
                                [FORMAT(format-name)]
                                [MIXED(YES|NO)]
                                [LOCK(YES|NO)]
```

```
                                    [CONFIRM(YES|NO)]
                                    [WS(YES|NO)]
                                    [PRESERVE]
                                    [PARM(parm-var)]
```

## Call Invocation Format

```
CALL  ISPLINK ('EDIT    ', {dsname} ,[serial]
                                    ,[pswd-value]
                                    ,[panel-name]
                                    ,[macro-name]
                                    ,[profile-name]
                                    ,{data-id}
                                    ,[member-name]
                                    ,[format-name]
                                    ,['YES     '|'NO      ']
                                    ,['YES     '|'NO      ']
                                    ,['YES     '|'NO      ']
                                    ,{ws-filename-buffer-name}
                                    ,['YES'|'NO']
                                    ,['PRESERVE']
                                    ,' '
                                    ,[parm-var] );
```

*OR*

```
CALL ISPEXEC (buf-len, buffer);
```

## Parameters

**dsname**

The data set name, in TSO syntax, of the data set to be edited. This is equivalent to the "other" data set name on the Edit Entry Panel. You can specify a fully qualified data set name enclosed in apostrophes (' '). If the apostrophes are omitted, the TSO data set prefix from the user's TSO profile is automatically attached to the data set name. The maximum length of this parameter is 56 characters.

For ISPF libraries and MVS partitioned data sets, you can specify a member name or a pattern enclosed in parentheses. If you do not specify a member name or if you specify a member pattern as part of the dsname specification when the DATASET keyword is used, a member selection list for the ISPF library, concatenation of libraries, or MVS partitioned data set is displayed. See the *ISPF User's Guide Volume I* for more information about patterns and pattern matching.

**Note:** You can also specify a VSAM data set name. If a VSAM data set is specified, ISPF checks the ISPF configuration table to see if VSAM support is enabled. If it is, the specified tool is invoked. If VSAM support is not enabled, an error message is displayed.

**serial**
The serial number of the volume on which the data set resides. If you omit this parameter or code it as blank, the system catalog is searched for the data set name. The maximum length of this parameter is 6 characters.

**pswd-value**
The password if the data set has MVS password protection. Do not specify a password for RACF- or PCF-protected data sets.

**panel-name**
The name of a customized edit panel, created by you, to be used when displaying the data. Refer to *ISPF Planning and Customizing* for information about developing a customized panel.

**macro-name**
The name of the first edit macro to be executed after the data is read, but before it is displayed. Refer to *ISPF Edit and Edit Macros* for more information.

**profile-name**
The name of the edit profile to be used. If you do not specify a profile name, the profile name defaults to the ISPF library type or last qualifier of the "other" TSO data set name. See the *ISPF Edit and Edit Macros* for more information.

**format-name**
The name of the format to be used to reformat the data. The format-name parameter is provided to support the IBM 5550 terminal using the Double-Byte Character Set (DBCS).

**YES | NO**
For the MIXED parameter, if YES is specified, the EDIT service treats the data as mixed-mode DBCS data. If NO (the default value) is specified, the data is treated as EBCDIC (single-byte) data. This parameter is provided to support the IBM 5550 terminal using the Double-Byte Character Set (DBCS).

**YES | NO**
The LOCK parameter is no longer used since the removal of LMF from the ISPF product, but is left in for compatibility purposes. If YES is specified the edit service will fail with return code 12. If you want to be able to specify YES and have the editor ignore the value, change the FAIL_ON_LMF_LOCK keyword value in the ISPF Configuration Table to NO.

**YES | NO**
For the CONFIRM parameter, if you specify YES (the default) and then attempt to CANCEL, MOVE, or REPLACE data while in EDIT mode, ISPF displays a pop-up panel that requires you to confirm the action. Because members or data sets that are moved, cancelled, or replaced are deleted, CONFIRM acts as a safeguard against accidental data loss. If you want to terminate the edit session without saving the data, press the ENTER key. If you made a mistake and want to return to the edit session, enter the END command. If you specify NO as the CONFIRM value, you will not be required to confirm a CANCEL, MOVE, or REPLACE.

**YES | NO**
For the WS keyword, if you specify YES, the EDIT service edits the host data set or workstation file on the workstation, using the workstation editor

configured in the ISPF Workstation Tool Integration Program. For more information see the *ISPF User's Guide Volume II* If you specify NO as the WS value, the EDIT service edits the host data set or workstation file on the host using the PDF editor.

**data-id**
The data ID that was returned from the LMINIT service. The maximum length of this parameter is 8 characters.

You can use the LMINIT service in either of two ways before invoking the EDIT service:

- You can use LMINIT to allocate existing data sets by specifying a data set name or ISPF library qualifiers. LMINIT returns a data ID as output. This data ID, rather than a data set name, is then passed as input to the EDIT service.
- The dialog can allocate its own data sets by using the TSO ALLOCATE command or MVS dynamic allocation, and then passing the ddname to LMINIT. Again, a data ID is returned as output from LMINIT and subsequently passed to the EDIT service. This procedure is called the *ddname interface* to EDIT. It is particularly useful for editing VIO data sets, which cannot be accessed by data set name because they are not cataloged.

**member-name**
A member of an ISPF library or MVS partitioned data set, or a pattern. If you do not specify a member name when the MEMBER keyword or call invocation is used, or if a pattern is specified, a member selection list for the ISPF library, concatenation of libraries, or MVS partitioned data set is displayed. See the *ISPF User's Guide Volume I* for more information about patterns and pattern matching.

**ws-filename-buffer-name**
Specifies the name of a variable containing the path and the file name in the workstation's file system syntax of the workstation file to be edited. The maximum length of the path and the workstation file name within this variable is 256. If the path is omitted, the working directory configured in the ISPF Tool Configurator will be inserted in front of the workstation file name to resolve the relative path. For more information see the *ISPF User's Guide Volume II*.

**buf-len**
A fullword fixed binary integer containing the length of the buffer parameter.

**buffer**
A buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC invocation for a command invocation.

**PRESERVE**
When specified, the editor stores the original length of each record in variable length data sets and when a record is saved, the original record length is used as the minimum length for the record. The editor always includes a blank at the end of a line if the length of the record is zero or eight. Records can be extended by adding nonblank data to the record or by using the SAVE_LENGTH edit macro command. For more information, refer to the *ISPF Edit and Edit Macros* manual.

**parm-var**
The name of an ISPF variable that contains parameters which are to be passed

to the initial macro specified by *macro-name*. The variable value must not exceed 200 bytes in length. If no macro name is specified, parm-var must be blank or not specified.

## Return Codes

The following return codes are possible:

**0** Normal completion; data was saved.

**4** Normal completion; data was *not* saved for one of the following reasons.
- No data changes were made during the EDIT session. The CANCEL command was used to exit EDIT. Browse was substituted for EDIT because insufficient storage was available to read in the requested data.

**12** YES was specified for the LOCK parameter.

**14** Member or sequential data set in use.

**16** One of the following:
- No members matched the specified pattern.
- No members in the partitioned data set.

**18** A VSAM data set was specified but the ISPF Configuration Table does not allow VSAM processing.

**20** Severe error; unable to continue.

## Example 1

This example invokes the EDIT service for TELOUT, a member of the ISPFPROJ.FTOUTPUT data set.

### Command Invocation

```
ISPEXEC EDIT DATASET('ISPFPROJ.FTOUTPUT(TELOUT)') WS(YES)
OR

ISPEXEC LMINIT DATAID(EDT) DATASET('ISPFPROJ.FTOUTPUT')
ISPEXEC EDIT DATAID(&EDT) MEMBER(TELOUT) WS(YES)
```

### Call Invocation

```
CALL ISPLINK ('EDIT','ISPFPROJ.FTOUTPUT(TELOUT)',' ',' ',' ',' ',' ',' ',
    ' ',' ',' ',' ',' ',' ','YES');
OR
```

Set the program variable BUFFER to contain:

```
BUFFER = 'EDIT DATASET('ISPFPROJ.FTOUTPUT(TELOUT)'') WS(YES)';
```

Set the program variable BUFFLN to the length of the variable BUFFER. Issue the following command:

```
CALL ISPEXEC (BUFFLN, BUFFER);
```

## Example 2

This example invokes the EDIT service for a workstation file, c:\config.sys, using the PDF editor on the host.

### Command Invocation

Set the command variable WSFNNAME to contain:

```
WSFNNAME='c:\config.sys'
ISPEXEC EDIT WSFN(WSFNNAME) WS(NO)
```

### Call Invocation

Set the program variable to contain:

```
WSFNNAME='c:\config.sys';
CALL ISPLINK('EDIT',' ',' ',' ',' ',' ',' ',
    ' ',' ',' ',' ',' ',' ',' WFSNNAME','NO');
```

*OR*

Set the program variable WSFNNAME to contain:

```
WSFNNAME='c:\config.sys';
```

Set the program variable BUFFER to contain:

```
BUFFER='EDIT WSFN(WSFNNAME) WS(NO)';
```

Set the program variable BUFFLEN to the length of the variable BUFFER. Issue the following command:

```
CALL ISPEXEC(BUFFLEN,BUFFER);
```

## EDREC – Specify Edit Recovery Handling

The EDREC service initializes an edit recovery table, determines whether recovery is pending, and takes the action specified by the first argument.

**Note:** Dialogs that invoke the EDIT service should invoke the EDREC service first to start edit recovery, because the EDIT service does not perform edit recovery.

When you invoke the EDREC service, EDREC displays a special panel. Using this panel you can recover data, cancel recovery, defer recovery until a later time, or enter the END command to return to the next sequential command in your command invocation or to return to the next sequential instruction in your program.

The EDREC service attempts to use the panel that you specified in the EDIT service from which it is recovering. Make sure that this panel is available to the EDREC service. It must be in a library allocated to ISPPLIB or available through a LIBDEF.

**Note:** You can use the ZEDUSER variable to save LIBDEF information or the panel name when you invoke EDIT. This is different from edit recovery entered from option 2, because option 2 always uses its default panel.

### Command Invocation Format

```
ISPEXEC  EDREC {INIT    [CMD(command-name)]}
               {QUERY  }

               {PROCESS [PASSWORD(pswd-value)]

               [DATAID(data-id)]}

               {CANCEL }

               {DEFER  }
```

## Call Invocation Format

```
CALL ISPLINK ('EDREC   ' {,'INIT   ' [,command-name]}
                         {,'QUERY  '}

                         {,'PROCESS ' [,pswd-value][,data-id]}

                         {,'CANCEL  '}

                         {,'DEFER   '});
```

*OR*

```
CALL ISPEXEC (buf-len, buffer);
```

## Parameters

**INIT**
Initializes an edit recovery table in your profile library if one does not already exist for the current application. The edit recovery table is saved in the data set allocated to ddname ISPPROF in member *xxxx*EDRT, where *xxxx* is the ISPF application ID.

**command-name**
A CLIST or REXX exec that starts the table. If you omit this parameter, the INIT option invokes an PDF-supplied CLIST (ISREDRTI) that creates an eight-row edit recovery table, permitting eight levels of concurrent Edit sessions with recovery active. The Edit sessions can result from recursion or split-screen usage.

If you specify a command with the INIT option, the command should be patterned after ISREDRTI. It can create a different number of rows or use a different naming convention for the backup data sets, or specify “keep” instead of “delete” as the backup data set disposition. The format of the edit recovery table must be the same as that specified in ISREDRTI.

**QUERY**
Causes EDREC to search the edit recovery table for a pending recovery. When the QUERY option is specified, EDREC scans the edit recovery table for an entry containing a recovery pending condition. If the return code is 4, indicating an entry was found, the dialog must call EDREC with the PROCESS, CANCEL, or DEFER option.

EDREC QUERY is usually used in a loop, since there can be more than one pending recovery. Multiple recoveries can result from recursion or from split-screen usage of the dialog. Each subsequent call to EDREC with the QUERY option scans the table starting at the entry after the last one that was found. A typical loop, written in pseudo-code (showing the parameters themselves instead of sample values), is as follows:

```
SET DONE = NO
DO WHILE &DONE = NO
  ISPEXEC EDREC QUERY
  IF &LASTCC = 4 THEN -
    ISPEXEC EDREC PROCESS
  ELSE -
    SET DONE = YES
  END
```

As the preceding example shows, EDREC QUERY must be used before each invocation of any of the following EDREC functions: PROCESS, CANCEL, or DEFER.

The following variables are stored in the dialog function pool when EDREC is called with the QUERY option and the return code is 4, indicating that recovery is pending.

**ZEDBDSN** Backup data set name.

**ZEDTDSN** Target data set name.

**ZEDTMEM** Target member name, if applicable.

**ZEDTVOL** Volume serial of target data set, if a volume serial was specified on invocation of the EDIT service.

**ZEDROW** Row number of entry in edit recovery table.

The dialog can check the preceding variables and use them to display information to the user. If EDREC QUERY shows that recovery is not pending, the above variables are not meaningful.

ZEDUSER is an extension variable in the Edit Recovery Table that is provided to contain user data. Whatever data is in dialog variable ZEDUSER in the shared pool is saved to the ZEDUSER variable in the edit recovery table when the recovery data set is initialized. This is done if RECOVERY is ON when entering Edit or after using the SAVE command.

When EDREC is called with the QUERY option and the return code is 4, indicating that recovery is pending, or if ISPF option 2 edit recovery takes place, the data is read out of ZEDUSER in the table and returned to ZEDUSER in the shared and function pools. If recovery is not pending, this variable is not meaningful. The extension variable ZEDMODE indicates whether this is an edit session or a view session that is to be recovered.

**PROCESS**
Causes edit recovery to proceed.

**pswd-value**
The MVS password of the target data set. This parameter is valid only with the PROCESS option.

**data-id**
The data ID of the data set that will contain the recovered data. The recovered data should be saved in a data set other than the data set that was being edited when the system failure occurred. If you omit this parameter, EDREC attempts to save the recovered data in the original data set.

Before using the data ID parameter, the dialog must first invoke the LMINIT service to specify the target data set and then pass the data ID to the EDREC service. This procedure can also control the allocation of the target data set for recovery, even if it is not the original data set being edited. You *must* use this procedure if you originally specified the data set being edited to the EDIT service using the ddname interface.

**CANCEL**
Cancels edit recovery. The backup data set is erased and the corresponding entry in the edit recovery table is freed.

**DEFER**
Defers edit recovery. Recovery is canceled, but the backup data set is saved so that recovery can be processed in another Edit session.

**Attention:**

Use this parameter carefully. It can cause your original data set to be written over in the next Edit session.

**buf-len**
A fullword fixed binary integer containing the length of the buffer parameter.

**buffer**
A buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC invocation for a command invocation.

## Return Codes

The following return codes are possible:

| | | |
|---|---|---|
| **0** | Normal return. | |
| | **INIT** | Edit recovery table was successfully created. |
| | **QUERY** | Recovery is not pending. |
| | **PROCESS** | Recovery was completed and the data was saved. |
| **4** | Normal return. | |
| | **INIT** | Edit recovery table already exists for current application. |
| | **QUERY** | Entry found in edit recovery table; recovery is pending. |
| | **PROCESS** | Recovery was completed, but user did not save data. |
| **20** | Severe error; unable to continue. | |

## Example

This example invokes the EDREC service for INIT to create an edit recovery table if one does not exist.

### Command Invocation

```
ISPEXEC EDREC INIT
```

### Call Invocation

```
CALL ISPLINK ('EDREC   ', 'INIT    ');
OR
```

Set the program variable BUFFER to contain:

```
BUFFER = 'EDREC INIT';
```

Set the program variable BUFLEN to the length of the variable BUFFER. Issue the following command:

```
CALL ISPEXEC (BUFLEN, BUFFER);
```

# FILESTAT – Statistics for a file

Use the FILESTAT service to determine statistics about a file on the connected workstation. FILESTAT returns information that the file exists, the longest length record in the file, and the date and time on the file.

**Note:** Be aware that when the FILESTAT service is issued for certain reserved or restricted operating system file names, your workstation operating system or host system can cease to function, or "hang". See your workstation operating system documentation to note reserved or restricted file names.

## Command Invocation Format

```
ISPEXEC FILESTAT FILE(var-name)
                 [LRECL(var-name),DATE(var-name)]
```

## Call Invocation Format

```
CALL  ISPLINK ('FILESTAT',var-name
              ,[var-name,var-name])
```

## Parameters

**FILE var-name**
The variable name containing the workstation file for which you want statistics. If no parameters other than the file name are specified, the function will verify that the file exists with a return code of 0.

**LRECL var-name**
The variable name in which ISPF returns the longest record length of the file you specify. The variable is returned in character format and has a length of 8.

**DATE var-name**
The variable name in which ISPF returns the date and time that the specified workstation file was last changed. The date and time are returned in character format, and have a length of 19. The date format is 'yyyy/mm/dd hh:mm:ss'.

## Return Codes

The following return codes are possible:

**0** Workstation file exists.

**4** Workstation file does not exist.

**8** Error in variable specification.

**10** No workstation connection exists.

**12** Workstation device is not ready.

**20** Severe error in FILESTAT service.

## Example

This exec verifies the existence of LANGSEL.DES in the D:\DESIGN directory on the workstation. The lrecl of the workstation file is saved in variable LVAR and the date is save in variable DVAR.

```
/* REXX */
fivar = 'd:\design\langsel.des'
address ispexec 'filestat file(fivar) lrecl(lvar) date(dvar)'
```

# FILEXFER – Upload or Download File

The FILEXFER service is used to upload files from the workstation or download files to the workstation.

**Note:** Be aware that when the FILEXFER service is issued for certain reserved or restricted operating system file names, your workstation operating system or

host system can cease to function, or ″hang″. See your workstation operating system documentation to note reserved or restricted file names.

You use the HOST(var-name) and WS(var-name) keywords to specify the host data set name (and member if it is a PDS) and the workstation file name involved in the file transfer. For each, a variable is specified that contains the host or workstation identifier. The TO(HOST|WS) keyword determines the direction of the file transfer. TO(HOST) specifies that the file named in the WS(var-name) keyword should be uploaded to the data set (and member) named in the HOST(var-name) keyword. TO(WS) specifies that the data set (and member) named in the HOST(var-name) keyword should be downloaded to the file named in the WS(var-name) keyword.

The host data set specification should follow normal TSO naming conventions. If the host data set you specify is unquoted, the user's prefix is used as the high-level qualifier of the data set.

The workstation file name must include the drive and directory information. The host data set name should include a member name when applicable. ISPF variables are used for the file names. Use the ISPF VPUT service to put the variables in the variable pool before starting this command.

## Command Invocation Format

```
ISPEXEC  FILEXFER  HOST(var-name) WS(var-name) TO(HOST|WS)
                  [VOLUME(volume)]
                  [BINARY|TEXT]
                  [STATS|NOSTATS]
                  [CHKDATE (YES|NO)]
                  [SETDATE (YES|SCLM|NO)]
                  [MAKEPATH (YES|NO)]
```

## Call Invocation Format

```
CALL  ISPEXEC (buf-len, buffer);
OR

CALL  ISPLINK ('FILEXFER',host_var,ws_var,'HOST    '|'WS      ',
                          [volume,
                          'BINARY  '|'TEXT    ',
                          'STATS   '|'NOSTATS ',
                          'YES     '|'NO      ',
                          'YES     '|'SCLM    '|'NO      '
                                  'YES     '|'NO      '])
```

## Parameters

**HOST var-name**
An ISPF variable that contains the name of the host file that is to be transferred.

**WS var-name** An ISPF variable that contains the name of the workstation file that is to be transferred.

**TO (host or ws)**
An ISPF variable that tells where the file will be created upon the completion of the transfer.

- if TO HOST is specified:

  - in binary mode—data sets with fixed length records are padded with null characters if needed to fill the final record of the data set. Data sets with variable length records are not padded.
  - in text mode—for OS/2 and Windows systems, a Carriage-Return/Line-Feed combination is considered the end of a line. An end-of-file character that is the last character in the file is not transmitted to the host. End-of-file characters at other locations in the file are transmitted to the host along with the data following the end-of-file character.

    For AIX and HP-UX, a Line-Feed character is considered the end of a line.

    If a line is longer than the record length of the data set the line is split into as many records as are needed to hold the line.

    For data sets with fixed length records lines are padded with blanks to reach the record length.
- if TO WS is specified:
  - in binary mode—all data from each record in the host data set is concatenated to form the workstation file.
  - in text mode—blanks after the last nonblank character are trimmed form each record. Records consisting entirely of blanks are sent as a line with a single blank character. Zero-length records from a data set with variable length records are not transferred to the workstation, and can cause an error condition to occur.

If a data set containing end-of-line characters is transferred to the workstation and back to the host, the end-of-line characters are removed and the line is split at the location of the end-of-line characters. For Windows systems, an end-of-file character at the end of the data set is also removed when the data set is transferred to the workstation and back to the host.

**volume** An ISPF volume number for the location of the 'TO' file. Used only when working with uncataloged data sets.

**BINARY** Specifies that no ASCII/EBCDIC character translation should take place.

**TEXT** Specifies that ASCII/EBCDIC translation should take place.

**buffer** Specifies a buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC call for a command procedure.

**STATS** Specifies that ISPF statistics will be generated or maintained by incrementing the mod level and setting the last changed time to the time when the member is uploaded. This is the default.

**NOSTATS** Specifies that ISPF statistics will not be generated or maintained when the member is uploaded.

**buf-len** Specifies a fullword fixed binary integer containing the length of "buffer".

**CHKDATE**

**YES** On download, check the date and time of the source file and the target file. If they are different, perform the file

transfer and update the date and time of the target file to match those of the source file. If they are the same, do not perform the file transfer, and set the return code to 2.

If the source file you are downloading from the host to the PC does not have an entry in the date and time statistics, the current date is set on the host file at the time of the file transfer.

**NO** Transfer the files regardless of dates and times.

**SETDATE**

**YES** On upload, set the date and time of the host file to be the same as the workstation file.

**SCLM** On upload, set the date and time of the host file to be the same as the workstation file. In addition to this, set the SCLM bit on.

**NO** On upload, use the system data and time for the host files.

**MAKEPATH**

**YES** On transfer to the workstation, create the necessary subdirectories to store the file transferred. This parameter is ignored on a transfer to the host.

**NO** On transfer to the workstation, do not create any subdirectories to store the file transferred. This parameter is ignored on a transfer to the host. NO is the default for this parameter.

## Return Codes

The following return codes are possible:

**2** Source file and target file have the same date and time.

**4** LMSTAT failed.

**6** LMINIT failed.

**7** Error in variable specification.

**8** ″TO″ direction not valid.

**9** Host name is too long.

**10** No workstation connection exists.

**11** Return code 1 from DTTRANSFER. Host data set had null object handle.

**12** Return code 2 from DTTRANSFER. Workstation file had null object handle.

**13** Return code 3 from DTTRANSFER. Host data set could not be opened.

**14** Return code 4 from DTTRANSFER. Workstation file could not be opened.

**15** Return code 5 from DTTRANSFER. Error reading host data set.

**16** Return code 6 from DTTRANSFER. Error reading workstation file.

**17** Return code 7 from DTTRANSFER. Error writing host data set.

**18** Return code 8 from DTTRANSFER. Error writing workstation file.

**19** Return code 9 from DTTRANSFER. Error closing host data set.

**20** Severe error in transfer service.

**21** Return code 10 from DTTRANSFER. Error closing workstation file.

**22** Return code 11 from DTTRANSFER. User refused file access.

**23** Data set or member in use.

## Example

The following exec demonstrates a file transfer from the host to the workstation.

```
/*  REXX  */
VAR1 = 'MYMVS.FILE(STUFF)'
VAR2 = 'E:\MYOS2.FILE'
ADDRESS ISPEXEC 'VPUT (VAR1 VAR2)'
ADDRESS ISPEXEC 'FILEXFER HOST(VAR1) WS(VAR2) TO(WS) TEXT'
```

# FTCLOSE – End File Tailoring

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

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

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

The library parameter should be specified if a library other than that represented by the ISPFILE or LIBDEF definition is to be used. The library parameter is ignored if the "TEMP" option (temporary file) is specified on the FTOPEN service or if the ISPFILE definition specifies a sequential data set. A severe error occurs if file tailoring attempts to use a data set that is not a library.

The NOREPL parameter specifies that an existing member in the file tailoring output library is not to be overlaid by the current FTCLOSE service. If a member of the same name already exists, the FTCLOSE service request is terminated with a return code of 4 and the original member remains unaltered.

## Command Invocation Format

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

                  [NOREPL]
```

## Call Invocation Format

```
CALL  ISPEXEC (buf-len, buffer);
OR

CALL  ISPLINK ('FTCLOSE ' [,member-name]

                          [,library]

                          [,'NOREPL  ']
```

## Command Invocation Format

```
ISPEXEC  FTERASE  member-name  [LIBRARY(library)]
```

## Call Invocation Format

```
CALL  ISPEXEC (buf-len, buffer);
```

*OR*

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

## Parameters

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

**library** Specifies the name of a DD statement or lib-type on the LIBDEF service request that defines the output library that holds the member to be deleted. ISPFILE is the default if this parameter is omitted.

**buf-len** Specifies a fullword fixed binary integer containing the length of "buffer".

**buffer** Specifies a buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC call for a command procedure.

## Return Codes

The following return codes are possible:

**0** Normal completion.

**8** File does not exist.

**12** Output file in use; ENQ failed.

**16** Alternate output library not allocated.

**20** Severe error.

## Example

Erase member TELOUT in the file tailoring output library.

```
ISPEXEC FTERASE TELOUT
```

Set the program variable BUFFER to contain:

```
FTERASE TELOUT
```

Set program variable BUFLEN to the length of the variable BUFFER. Issue the following:

```
CALL ISPEXEC (BUFLEN, BUFFER);
```

or alternately

```
CALL ISPLINK ('FTERASE ','TELOUT ');
```

# FTINCL – Include a Skeleton

The FTINCL service specifies the skeleton that is to be used to produce the file tailoring output. If an FTOPEN service has not already been issued, the FTINCL service performs the equivalent of an FTOPEN, without the TEMP keyword, before processing the specified skeleton.

## Command Invocation Format

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

## Call Invocation Format

```
CALL  ISPEXEC (buf-len, buffer);
OR

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

## Parameters

**skel-name** Specifies the name of the skeleton.

**NOFT** Specifies that no file tailoring is to be performed on the skeleton: the entire skeleton is to be copied to the output file exactly as is with no variable substitution or interpretation of control records.

**buf-len** Specifies a fullword fixed binary integer containing the length of "buffer".

**buffer** Specifies a buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC call for a command procedure.

## Return Codes

The following return codes are possible:

**0** Normal completion.

**8** Skeleton does not exist.

**12** Skeleton in use; ENQ failed.

**16** Data truncation occurred or skeleton library or output file not allocated.

**20** Severe error.

## Example

Perform file tailoring using the file tailoring skeleton named TELSKEL, a member in the file tailoring skeleton library, to control processing.

```
ISPEXEC FTINCL TELSKEL
```

***or*** Set the program variable BUFFER to contain:

```
FTINCL TELSKEL
```

Set program variable BUFLEN to the length of the variable BUFFER. Issue the following:

```
CALL ISPEXEC (BUFLEN, BUFFER);
```

or alternately:

```
CALL ISPLINK ('FTINCL  ','TELSKEL ');
```

## FTOPEN – Begin File Tailoring

The FTOPEN service, which begins the file tailoring process, allows skeleton files to be accessed from the skeleton library specified by ddname ISPSLIB. The skeleton library must be allocated before invoking ISPF. ISPSLIB can specify a concatenation of files.

If output from file tailoring is not to be placed in a temporary file, the desired output file must be allocated to the ddname ISPFILE before invoking this service. ISPFILE can designate either a library or a sequential file. The skeleton files can contain variable-length records, with a maximum record length of 255.

The same rules apply for DBCS-related variable substitution in file tailoring as those described for file skeleton definition.

### Command Invocation Format

```
ISPEXEC  FTOPEN  [TEMP]
```

### Call Invocation Format

```
CALL  ISPEXEC (buf-len, buffer);
OR

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

49 ()TL4del( 482.6837•70287i76D2 018 72 0 m ([TEMP])Tj EL2i .970b])T  8 0

## Return Codes

The following return codes are possible:

**0** Normal completion.

**8** File tailoring already in progress.

**16** Skeleton library or output file not allocated.

**12** Output file in use; ENQ failed.

**20** Severe error.

## Example

Prepare for access (open) both the file tailoring skeleton and file tailoring output libraries.

```
ISPEXEC FTOPEN
```

Set the program variable BUFFER to contain:

```
FTOPEN
```

***or*** Set program variable BUFLEN to the length of the variable BUFFER. Issue the following:

```
CALL ISPEXEC (BUFLEN, BUFFER);
```

or alternately:

```
CALL ISPLINK ('FTOPEN  ');
```

# GETMSG – Get a Message

The GETMSG service obtains a message and related information from the message file. The short and long message text, help panel name, and alarm indicator can be obtained for a specified message-id. Values for all variables defined in the message are substituted when the message text is retrieved. If the desired message information is not present for the short message text, long message text, or help panel name, the corresponding variable name specified in the GETMSG service request is set to a null value. If the alarm indicator is not present on the message, a value of “NO” is returned in the alarm-name variable.

A message type of critical (.TYPE=CRITICAL) on the message definition statement overrides the values specified for the alarm and window keywords. For critical messages, the dialog manager sounds the alarm and places the message in a message pop-up window that requires a response. If GETMSG asks for the .ALARM value to be returned, the value returned will be YES, reflecting the fact that .TYPE=CRITICAL has forced that value. This is the case if .ALARM was not specified (which would normally default to NO) or if .ALARM=NO is actually defined for the message.

All the parameters except the message-id are optional. If the optional parameters are omitted, GETMSG simply validates the existence of the specified message.

## Command Invocation Format

```
ISPEXEC  GETMSG  MSG(message-id)  [SHORTMSG(short-message-name)]
                                  [LONGMSG(long-message-name)]

                                  [ALARM(alarm-name)]
```

```
                                    [HELP(help-name)]
                                    [TYPE(type-name)]
                                    [WINDOW(window-name)]
                                    [CCSID(ccsid-name)]
```

## Call Invocation Format

```
CALL  ISPEXEC (buf-len, buffer);
OR

CALL  ISPLINK ('GETMSG  ' ,message-id  [,short-message-name]
                                       [,long-message-name]
                                       [,alarm-name]
                                       [,help-name]
                                       [,type-name]
                                       [,window-name]
                                       [,ccsid-name]);
```

## Parameters

**message-id** Specifies the identification of the message for which information is to be retrieved.

**short-message-name**
Specifies the name of a variable into which the short message text, if any, is to be stored.

**long-message-name**
Specifies the name of a variable into which the long message text is to be stored.

**alarm-name** Specifies the name of a variable into which the alarm indicator of "NO" or "YES" is to be stored.

**help-name** Specifies the name of a variable into which the help panel name, if any, is to be stored.

**type-name** Specifies the name of the variable into which the message type, if any, (notify, warning or critical) is to be stored.

**window-type** Specifies the name of the variable into which the window type, if any (RESP or NORESP), is to be stored.

**ccsid-name** Specifies the name of the variable into which the CCSID, if any, is to be stored.

**buf-len** Specifies a fullword fixed binary integer containing the length of "buffer".

**buffer** Specifies a buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC call for a command procedure.

## Return Codes

The following return codes are possible:

**0** Normal completion.

**12** The specified message could not be found.

**20** Severe error.

## Example

For the message named ABCS102, return the text of the long message in variable ERRMSG and the help panel name in variable HPANEL.

```
ISPEXEC GETMSG MSG(ABCS102) LONGMSG(ERRMSG) HELP(HPANEL)
```

***or*** Set the program variable BUFFER to contain:

```
GETMSG MSG(ABCS102) LONGMSG(ERRMSG) HELP(HPANEL)
```

Set program variable BUFLEN to the length of the variable BUFFER. Issue the following:

```
CALL ISPEXEC (BUFLEN, BUFFER);
```

or alternately

```
CALL ISPLINK ('GETMSG  ','ABCS102 ',' ','ERRMSG ',
              ' ','HPANEL ');
```

# GRERROR – Graphics Error Block Service

This service is used only with CALL ISPLINK or CALL ISPLNK calls.

The GRERROR service returns to the caller the address of the GDDM error record and the address of the GDDM call format descriptor module.

This service allows the dialog developer to examine the error record provided by GDDM from GDDM function calls. Since the dialog uses the same application anchor block (AAB) as ISPF and cannot use the FSEXIT function, this information would otherwise be unavailable. See the *GDDM Programming Reference* for information about the GDDM error record and the call format descriptor module.

## Command Invocation Format

```
ISPEXEC  *This service does not apply to
          command or APL2 procedures*
```

## Call Invocation Format

```
CALL  ISPEXEC  *This service cannot be used
                with this interface*
OR

CALL  ISPLINK  ('GRERROR ', error-record-pointer,
                         call-format-descriptor-module-pointer);
```

## Parameters

**error-record-pointer**
Specifies a 4-byte program variable where the address of the GDDM error record is returned.

**call-format-descriptor-module-pointer**
Specifies a 4-byte program variable where the address of the GDDM call format descriptor module is returned.

## Return Codes

The following return codes are possible:

**0** Normal completion

**8** ISPF/GDDM interface is not established

**20** Severe error.

# GRINIT – Graphics Initialization

This service is available only with CALL ISPLINK or CALL ISPLNK calls.

The GRINIT service initializes the ISPF/GDDM interface and optionally requests that ISPF define a panel's GRAPHIC area as a GDDM graphics field. This service also replaces the FSINIT or SPINIT GDDM calls.

Graphic areas are not supported in GUI mode. However, you have some options if you request that an ISPF/GDDM interface be initialized:

- If you specify a panel name in your GRINIT request, which indicates that you intend to define a graphic area in the panel, you can choose one of the following options:
  1. To display the panel with the graphic area in the host emulator session

     **Note:** If you are in split screen mode, the graphic area panel cannot be displayed on the host.
  2. To display the panel without the graphic area on your workstation.
- If you issue a GRINIT request *without* specifying a panel name, you can choose from the following options:
  1. To continue running you application until a graphic panel is encountered, at which time you can choose one of the options provided for GRINIT calls that *do* specify a panel name
  2. To terminate the GDDM initialization, which returns a code of 20.

If you have specified GUISCRD or GUISCRW values on the ISPSTART invocation that are different than the actual host screen size, GDDM cannot be initialized and the GRINIT service will end with a return code of 20.

GDDM or PGF functions are accessed by the dialog through the GDDM reentrant or system programmer interfaces. These interfaces are described in the *GDDM Base Application Programming Reference*.

The dialog must provide an 8-byte area, called an application anchor block (AAB), which is on a fullword 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
```

In addition, the following GDDM calls, while permitted, can interfere with the ISPF/GDDM session:

```
DSCLS  DSDROP  DSOPEN  DSRNIT  DSUSE  DSCMF
```

If a dialog uses GDDM calls to put alphanumeric fields on a display, these fields are displayed only if there are no fields in the body of the ISPF panel definition. Other fields are not displayed. This means that alphanumeric fields can be displayed by *either* ISPF or the dialog through the use of GDDM, but not by both.

In addition, when using GDDM to put alphanumeric fields on a display, it is the dialog's responsibility to ensure that split-screen mode is not active before the display of the panel and that split-screen mode is disabled during the display of the panel.

**Notes:**

1. Terminals running in partition mode or terminals running with multiple screen widths, including the 3290 and the 3278 Mod 5, are not supported for graphics interface mode.
2. TSO Session Manager is disabled while graphics interface mode is active.

## Command Invocation Format

```
ISPEXEC  *This service does not apply to
          command or APL2 procedures*
```

## Call Invocation Format

```
CALL  ISPEXEC  *This service cannot be used
                with this interface*
OR

CALL  ISPLINK ('GRINIT  '  ,application-anchor-block
                          [,panel-name]);
```

## Parameters

**application-anchor-block**
Specifies the name of a variable containing an 8-byte application anchor block. This storage area can be updated by ISPF.

**panel-name**
Specifies the name of the panel containing the GRAPHIC area.

## Return Codes

The following return codes are possible:

**0** Normal completion.

**8** The specified panel does not contain a GRAPHIC area.

**12** The specified panel could not be found.

**20** Severe error.

### Example

Initialize the ISPF/GDDM interface and request that the graphic area in panel OURLOGO be defined as a GDDM graphics field.

```
CALL ISPLINK ('GRINIT  ',ABC,'OURLOGO ');
```

## GRTERM – Graphics Termination Service

This service is available only with CALL ISPLINK or CALL ISPLNK calls.

The GRTERM service indicates that the caller has completed all GDDM processing and that GDDM can now be terminated.

If the user is running in split-screen mode and the other task has requested GDDM, GDDM will still be used for displays.

### Command Invocation Format

```
ISPEXEC  *This service does not apply to
          command or APL2 procedures*
```

### Call Invocation Format

```
CALL  ISPEXEC  *This service cannot be used
                with this interface*
OR

CALL  ISPLINK ('GRTERM  ');
```

### Return Codes

The following return codes are possible:

**0** Normal completion

**20** Severe error.

## LIBDEF – Allocate Application Libraries

The LIBDEF service provides for the dynamic definition of application data sets, thus allowing application data sets to be specified during an ISPF session. This eliminates the need for allocate statements to define all application data sets before invoking an ISPF session.

The LIBDEF service can be used to define the following application-level libraries:

- Panels
- Messages
- Tables
- Skeletons
- File tailoring output
- User link libraries
- Images

The same ddnames used to define ISPF libraries are used to define data sets on the LIBDEF service requests. An application-level definition for ISPPROF, the ISPF profile library, is not permitted, because ISPPROF contains user-related data.

The LIBDEF service provides four ways to define application-level data sets:

1. Using the DATASET Keyword

   The LIBDEF service request can be issued with the DATASET keyword, together with a list of data set names that contain the application's dialog elements. For table and file tailoring output libraries, only one data set can be specified. For other libraries, a maximum of 15 names can be supplied in the data set list. All the data sets defined by LIBDEF must be cataloged.

   If application PAYROLL uses panels PAYINIT and PAYTERM (members of the library 'ISPFPROJ.ABC.PANELS'), the LIBDEF service request to identify the panels to ISPF can be:

   ```
   ISPEXEC LIBDEF ISPPLIB DATASET ID('ISPFPROJ.ABC.PANELS')
   ```

   The DISPLAY service would then be issued as:

   ```
   ISPEXEC DISPLAY PANEL(PAYINIT)
   ```

   Allocate statements need not be specified before ISPF is invoked for the data sets defined by the LIBDEF service with the DATASET keyword.

2. Using the EXCLDATA Keyword

   The LIBDEF service request can be issued with the EXCLDATA keyword, together with a list of user link library data set names. The EXCLDATA keyword can be used only with cataloged user link libraries.

   For example, if application PAYROLL uses two programs, PAYINIT and PAYTERM, which are members of the partitioned data set ISPFPROJ.ABC.PROGRAMS, the LIBDEF service request for identifying the programs to ISPF can be issued as:

   ```
   ISPEXEC LIBDEF ISPLLIB EXCLDATA ID('ISPFPROJ.ABC.PROGRAMS')
   ```

   See "User Link Libraries" on page 86 for a discussion on the effect of the EXCLDATA specification on member searches.

   Allocate statements need not be specified before ISPF is invoked for the data set defined by the LIBDEF service with the EXCLDATA keyword.

3. Using the LIBRARY Keyword

   The LIBRARY keyword on a LIBDEF service request associates an allocated ddname with an ISPF data element type. For example, if application PAYROLL uses panels PAYINIT and PAYTERM, a LIBDEF service request used to identify the panels to ISPF is:

   ```
   ISPEXEC LIBDEF ISPPLIB LIBRARY ID(PAYDD)
   ```

   Before issuing this LIBDEF service request, you must issue:

   ```
   ALLOCATE FI(PAYDD) DA('ISPFPROJ.ABC.PANELS') SHR
   ```

   The DISPLAY service would then be issued as:

   ```
   ISPEXEC DISPLAY PANEL (PAYINIT)
   ```

4. Using the EXCLLIBR Keyword

   The EXCLLIBR keyword on a LIBDEF service request associates an allocated user link library ddname with the ISPF link library dialog element type. For example, if application PAYROLL uses programs PAYINIT and PAYTERM, a LIBDEF service request for identifying the programs to ISPF is:

   ```
   ISPEXEC LIBDEF ISPLLIB EXCLLIBR ID(PAYDD)
   ```

   Prior to issuing this LIBDEF service request, you must issue:

   ```
   ALLOCATE FI(PAYDD) DA('ISPFPROJ.ABC.PROGRAMS') SHR
   ```

   See "User Link Libraries" on page 86 for a discussion on the effect of the EXCLLIBR specification on member searches.

The DATASET (or EXCLDATA) and LIBRARY (or EXCLLIBR) keywords are mutually exclusive.

**Note:** The QLIBDEF service allows an ISPF dialog to obtain the current LIBDEF definition information. This information can be saved by the dialog and used later to restore any LIBDEF definitions that may have been overlaid. For each LIBDEF lib-type, the ID parameter and the type of ID is returned. For more information, see "QLIBDEF – Query LIBDEF Definition Information" on page 180.

An application invoked from ISPF issues LIBDEF requests to define the application-level libraries that will be in effect while the application is running. This feature might improve the search time for libraries that are defined at the application level, but it adds an extra search level for entities that exist in the ISPF product library definitions.

The currently allocated ISPF libraries must still be defined before invoking ISPF and cannot be changed while in an ISPF session. If no application-level libraries have been defined, the current set of allocated ISPF libraries is searched. If an application-level library is defined, it is searched before the allocated ISPF libraries. Within a given application, when a LIBDEF has been defined with either the DATASET (or EXCLDATA) or LIBRARY (or EXCLLIBR) keyword, and another LIBDEF request is issued with either keyword for the same lib-type, the second definition takes precedence over the first. If the user specifies the COND keyword on the service call, the application-level library is defined only if there is no application-level library already defined for the specified type (for example, messages or panels).

The absence of the DATASET (or EXCLDATA) or LIBRARY (or EXCLLIBR) keyword, or the presence of either keyword with a null data set list, indicates that an application-level definition for the specified type is removed, if one exists.

The LIBDEF service also allows users to define a generic library type. The generic library extends the use of the LIBRARY parameter on DM component services such as TBCLOSE, TBOPEN, or TBSAVE, by allowing the user to specify the name of a LIBDEF generic library.

When the DATASET keyword is specified with the LIBDEF service, it causes the newly defined application-level library to be searched before the allocated ISPF library for a particular type. To allow the user to continue to define user-level libraries that are to be searched first, the following new ddnames must be specified in ALLOCATE commands before ISPF is invoked:

**ISPMUSR** User message library
**ISPPUSR** User panel library
**ISPSUSR** User skeleton library
**ISPTUSR** User table library
**ISPTABU** User table output library
**ISPFILU** User file tailoring output library
**ISPLUSR** User link library
**ISPIUSR** User image library.

The LIBDEF service only affects the ISPF DDs. To alter the SYSPROC concatenation sequence, use the TSO/E ALTLIB command.

**Note:** When the user ddname for the library type is defined, data set names allocated to it are treated as being concatenated ahead of those specified on the LIBDEF service request. The rules governing concatenation of data sets apply.

Only the first 15 data sets allocated to these user ddnames will be searched by ISPF before the LIBDEF application-level library.

In the case of ISPLLIB, EXCLDATA can be used instead of DATASET, and EXCLLIBR instead of LIBRARY exclusively. Using one of these keywords (EXCLDATA or EXCLLIBR) indicates that when searching for the LOAD module, ISPF is only considering the application-level libraries defined by the LIBDEF service. That is, user libraries and ISPF base libraries are not used when EXCLDATA or EXCLLIBR is specified.

## Application Data Element Search Order

When two or more input libraries are to be searched for an item, the search begins with the first library in a list and continues through the list until the item is found. For example, if the item searched for is of type “Panels” and a “LIBDEF with DATASET” service call is in effect, the input libraries (ISPPUSR, the LIBDEF defined library, and ISPPLIB) are searched consecutively in the order shown. The search stops when the item is found or when the last library has been searched.

The search of two or more output libraries proceeds in the same way, except that the first definition found is used as the repository for the output.

Table 2 defines the search sequence for all item types.

*Table 2. Search Sequence for Libraries*

| | **LIBDEF with EXCLDATA** | **LIBDEF with EXCLLIBR** | **LIBDEF with DATASET** | **LIBDEF with LIBRARY** | **No LIBDEF** |
|---|---|---|---|---|---|
| Panels | Not valid | Not valid | ISPPUSR<br>LIBDEF<br>ISPPLIB | LIBDEF<br>ISPPLIB | ISPPLIB |
| Messages | Not valid | Not valid | ISPMUSR<br>LIBDEF<br>ISPMLIB | LIBDEF<br>ISPMLIB | ISPMLIB |
| Table Input | Not valid | Not valid | ISPTUSR<br>LIBDEF<br>ISPTLIB | LIBDEF<br>ISPTLIB | ISPTLIB |
| Skeleton | Not valid | Not valid | ISPSUSR<br>LIBDEF<br>ISPSLIB | LIBDEF<br>ISPSLIB | ISPSLIB |
| Images | Not valid | Not valid | ISPIUSR<br>LIBDEF<br>ISPILIB | LIBDEF<br>ISPILIB | ISPILIB |
| Linklib (See note following this table) | Job Pack Area<br>LIBDEF<br>Link Pack Area<br>LINKLIB | Job Pack Area<br>LIBDEF<br>Link Pack Area<br>LINKLIB | Job Pack Area<br>ISPLUSR<br>LIBDEF<br>ISPLLIB<br>STEPLIB<br>Link Pack Area<br>LINKLIB | Job Pack Area<br>LIBDEF<br>ISPLLIB<br>STEPLIB<br>Link Pack Area<br>LINKLIB | Job Pack Area<br>ISPLLIB<br>STEPLIB<br>Link Pack Area<br>LINKLIB |

*Table 2. Search Sequence for Libraries (continued)*

| | LIBDEF with EXCLDATA | LIBDEF with EXCLLIBR | LIBDEF with DATASET | LIBDEF with LIBRARY | No LIBDEF |
|---|---|---|---|---|---|
| Table Output | Not valid | Not valid | ISPTABU LIBDEF | LIBDEF | ISPTABL |
| File Tailoring Output | Not valid | Not valid | ISPFILU LIBDEF | LIBDEF | ISPFILE |
| Table Services (Input) with LIBRARY Parameter | Not valid | Not valid | (Unchanged) | LIBDEF | Allocated Library |
| Table Services (Output) with LIBRARY Parameter | Not valid | Not valid | LIBDEF | (Unchanged) | Allocated Library |
| File Tailoring Services (Output) with LIBRARY Parameter | Not valid | Not valid | LIBDEF | (Unchanged) | Allocated Library |

**Notes:**

1. If a program in Linklib is to be attached as a command processor (that is, by using the SELECT CMD parameter) and the command is not defined in the TSO command characteristics table (ISPTCM), the search sequence illustrated here does not apply. Refer to *ISPF Planning and Customizing* for information about customizing ISPTCM for the correct search order.
2. When using a SELECT with NEWAPPL, you must include PASSLIB to use the LIBDEFs you defined. For more details, see the description of the NEWAPPL parameter under "SELECT – Select a Panel or Function" on page 185.
3. The image library with the associated ddname ISPILIB is not a required ISPF library. If you plan to use ISPF's image support in GUI mode, you must allocate the image input data set to ddname ISPILIB before using the images or before invoking the LIBDEF service with libtype ISPILIB.

## LIBDEF Display Utility

The LIBDEF Display Utility displays all active and stacked LIBDEF definitions for the current logical screen in a scrollable list. Optionally, a specific LIBDEF library definition may be selected.

The ISPF system command, ISPLIBD [libtype] invokes the LIBDEF Display Utility. The optional parameter, libtype, identifies a specific LIBDEF library definition to be displayed. All LIBDEF definitions for the current logical screen are displayed if the parameter is omitted, if the parameter is longer than 8 characters, or if the parameter specifies ISPPROF as the library name.

For each LIBDEF definition displayed, the following information is provided:

- Stack indicator

  An "S" is displayed to the left of the library name when a stacked LIBDEF definition is presented.
- Library
- Type
- ISPxUSR indicator (for type DATASET only)

- Identification

For type DATASET/EXCLDATA this column contains the data set names. The associated ISPxUSR data set names is shown when the respective DDNAME is allocated. The ISPxUSR data sets are not shown as part of a stacked definition.

For type LIBRARY/EXCLLIBR this column contains the library name (ddname) followed by the first or only allocated data set name.

The LIBDEF Display Utility supports the use of a LOCATE command. LOCATE is used to locate a specific LIBDEF library name. Two command abbreviations, LOC and L, are also supported.

| | |
|---|---|
| **LOCATE ISPPLIB** | Locates the LIBDEF definition for ISPPLIB |
| **LOC ISPMLIB** | Locates the LIBDEF definition for ISPMLIB |
| **L ISPSLIB** | Locates the LIBDEF definition for ISPSLIB |

Figure 6 shows a LIBDEF Utility display of all LIBDEF definitions. Figure 7 on page 85 shows a display of a single LIBDEF definition, and Figure 8 on page 85 shows a LIBDEF stacked definition.

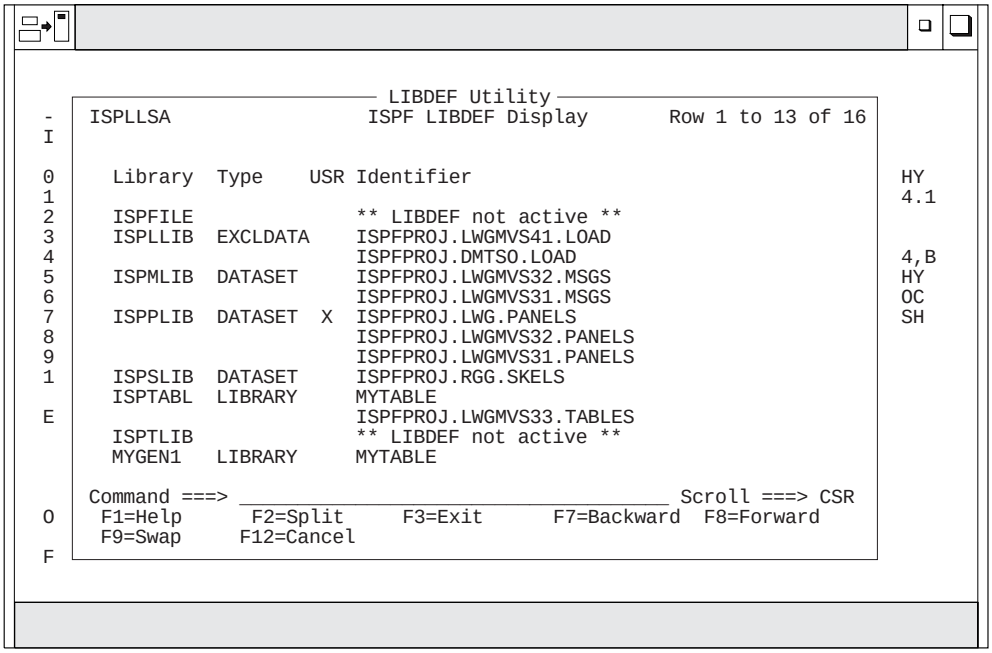



*Figure 6. ISPLIBD - all LIBDEF definitions*

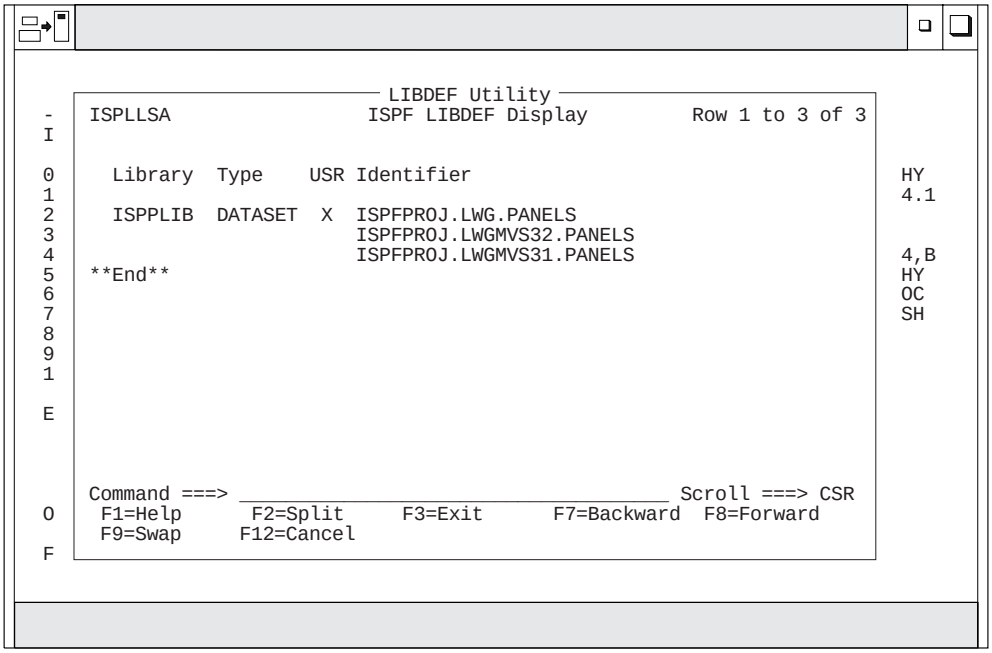



*Figure 7. ISPLIBD ISPPLIB - ISPPLIB LIBDEF definition*

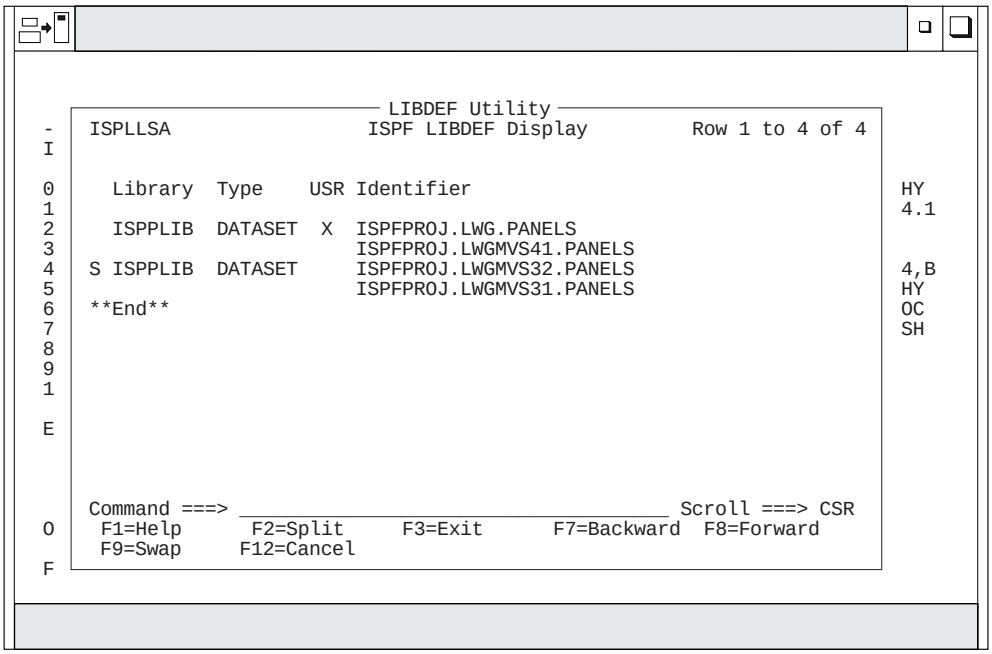



*Figure 8. ISPLIBD ISPPLIB - ISPPLIB LIBDEF stacked definition*

When you are in the Dialog Test utility (test environment), and you issue a LIBDEF for a panel data set from option 7.6, the LIBDEF is set up under the user environment. In order to display a panel from the library for which you issued the LIBDEF or to display the active LIBDEFs, you must go through a Dialog Test utility interface.

For example, from Dialog Test's option 7.6 issue:

```
LIBDEF ISPPLIB DATASET ID('xxxx.panels')
```

To display the active LIBDEFs, go to 7.1 (the Invoke Dialog Function/Selection Panel) and type `ISPLLS` at the PGM prompt and `ISPPLIB` at the PARM prompt; then press Enter.

**Note:** If you attempt to issue the ISPLIBD ISPPLIB command from the command line on the Dialog Test utility's option 7.6, the LIBDEF utility will indicate that ISPPLIB has no active LIBDEFs. This is because the Dialog Test utility runs in the test environment, not the user environment.

## User Link Libraries

The LIBDEF ISPLLIB service can be used to specify load libraries containing programs and command processors, which are part of an ISPF application. The LIBDEF ISPLLIB definition causes load modules to be searched in the specified load libraries by the SELECT service.

The LIBDEF library definitions are not searched by MVS member searches caused by the execution of ATTACH, LINK, LOAD, or XCTL macros within the selected program (SELECT PGM), or on the selection of authorized programs or commands. The LIBDEF library definitions are searched for selected commands (SELECT CMD).

The following rules apply:

- If the SELECT program service is invoked using ISPEXEC SELECT PGM(MYPROG), MYPROG is considered a member of the load libraries specified with LIBDEF ISPLLIB. If MYPROG then transfers control by using MVS contents supervision macros such as ATTACH, LINK, LOAD, or XCTL, any new requested program that exists only in the LIBDEF data set is not found, and an 806-04 abend occurs. This is because ISPF links to MYPROG, and MVS is not aware of the load library defined using LIBDEF ISPLLIB.
- If the SELECT program service is invoked using ISPEXEC SELECT CMD(MYCMD), MYCMD is considered a member of the load libraries specified with LIBDEF ISPLLIB. The command processor (a program coded to support a unique argument list format) can then use MVS contents supervision macros such as ATTACH, LINK, LOAD, or XCTL. This is because ISPF attaches MYCMD as a subtask to ISPF. The load library, defined using LIBDEF ISPLLIB, is passed as a task library to the subtask.

If LIBDEF is issued while in split screen, it will only affect the screen on which it is issued, because each screen is a separate ISPF session with its own TCB and tasklib.

## Message Libraries

Definition of a message library with LIBDEF will cause a search of that data set for the required message member before a search of the base message library. If the member in the LIBDEF-defined message library has the same name as a member in the base library, all messages within the base message data set member must be included in the LIBDEF-defined message data set member. If the message member found in the LIBDEF-defined message library does not contain the message being searched for, another search will not be made for the message in the base message library.

For example, if message ABCD009 is in the base library member ABCD00, but not in the LIBDEF-defined message library member ABCD00, message ABCD009 will not be found while the LIBDEF is active.

## Command Invocation Format

```
ISPEXEC LIBDEF lib-type [DATASET|EXCLDATA|LIBRARY|EXCLLIBR]
                        [ID(dataset-list)|ID(libname)]

                        [COND|UNCOND|STACK|STKADD]
```

## Call Invocation Format

```
   CALL ISPEXEC (buf-len, buffer);
OR

   CALL ISPLINK ('LIBDEF  ', lib-type
        [,'DATASET '|'EXCLDATA'|'LIBRARY '|'EXCLLIBR']

        [,dataset-list|libname]

        [,'COND    '|'UNCOND  '|'STACK   '|'STKADD  ']);
```

## Parameters

**lib-type** Indicates which type of ISPF ddname application-level library definition is being made. The value specified for lib-type must be padded with blanks, if needed, to make the total length 8 characters. For generic libraries it is the ddname as specified by the LIBRARY parameter of the corresponding table or file tailoring service.

Users can specify the following types of libraries:

| | |
|---|---|
| **ISPMLIB** | Message library |
| **ISPPLIB** | Panel library |
| **ISPSLIB** | Skeleton library |
| **ISPTLIB** | Table input library |
| **ISPTABL** | Table output library |
| **ISPFILE** | File tailoring output file |
| **ISPLLIB** | Load module library |
| **xxxxxxxx** | Generic library |
| **ISPILIB** | Image library |

ISPF ddname libraries can only be used for their intended purpose. Generic libraries can be used for table input, table output, or file tailoring output.

**DATASET** Indicates that ID specifies a list of cataloged data set names.

**EXCLDATA** Indicates that ID specifies a list of cataloged data set names. (Can only be used with ISPLLIB.)

**LIBRARY** Indicates that ID specifies a ddname. See libname.

**EXCLLIBR** Indicates that ID specifies a ddname. See libname. (Can only be used with ISPLLIB.)

**dataset-list** Indicates a list of cataloged data set names to be searched for the application. A maximum of 15 data set names cam be listed. (See "data-set-list" on page 7 for the specification of data set lists.)

**libname** Specifies the name of a previously allocated DD statement that defines the application-level library or libraries.

**COND** Specifies that the application-level library should be defined only if there is no active application-level library for the specified type.

**UNCOND** Specifies that the application-level library should be defined regardless of the existence of an application-level library for the specified type. This is the default.

**STACK** Specifies the current state of the lib-type LIBDEF definition is to be stacked before acting on the new request. Stacking occurs even when there is no active LIBDEF definition for the specified lib-type. A null definition is stacked when there is no active LIBDEF definition. This allows an application to issue a LIBDEF stack request for a particular lib-type without knowing if an active LIBDEF definition currently exists.

For example, it is valid to specify a LIBDEF definition for ISPPLIB and request that the current ISPPLIB LIBDEF definition be stacked, even when no current ISPPLIB LIBDEF definition exists. When the ISPPLIB LIBDEF definition that requested stacking is removed, there will be no active ISPPLIB LIBDEF definition in effect.

It is also valid to request stacking when resetting a particular LIBDEF definition. For example, it is valid to specify a reset of the ISPPLIB LIBDEF definition and request that the current ISPPLIB definition be stacked, even when no current ISPPLIB LIBDEF definition exists. A subsequent reset request of the ISPPLIB LIBDEF definition will restore the previously stacked ISPPLIB LIBDEF definition, including a restoration of a null definition.

**Note:** You can use STACK or STKADD on a LIBDEF statement. If both STACK and STKADD parameters are used on a single LIBDEF statement, ISPF uses only the last one specified.

**STKADD** Specifies the new LIBDEF request with the STKADD and DATASET parameters is to be added to the existing lib-type LIBDEF definition. STKADD concatenates the new LIBDEF request to the existing LIBDEFed lib-type definition. No stacking is done.

**Notes:**

1. You can use STACK or STKADD on a LIBDEF statement. If both STACK and STKADD parameters are used on a single LIBDEF statement, ISPF uses only the last one specified.
2. The STKADD parameter is restricted to use with the DATASET parameter. It is *not* for use with the EXCLDATA, LIBRARY, or EXCLLIBR parameters. ISPF issues a severe error message if STKADD is used with those parameters.

**buf-len** Specifies a fullword fixed binary integer containing the length of "buffer".

**buffer** Specifies a buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC invocation for a command procedure.

## Return Codes

The following return codes are possible:

**0** Normal completion

**4** Application library does not exist for this type (when removing the application library)

**8** Application library already exists for this type (when COND is used)

**12** ISPPROF was specified as the lib-type; invalid lib-type specified with EXCLDATA or EXCLLIBR.

**16** A libname was not allocated, or the dataset-list contains an invalid MVS dsname.

**20** Severe error.

**Note:** A return code of 0 can be received for a generic lib-type, even though the library does not exist. No allocation verification is done until the generic lib-type is referenced using the LIBRARY parameter on a file tailoring or table service request.

## Example 1 - The DATASET keyword

Assume that the user has issued the following ALLOCATE statements for a panel library before entering ISPF:

```
ALLOCATE DATASET('ISPFPROJ.ABC.MYPAN') FILE(ISPPUSR) SHR
ALLOCATE DATASET('ISPFPROJ.ABC.PANELS') FILE(ISPPLIB) SHR
```

Next, the LIBDEF service is invoked with the DATASET keyword to define an application-level panel library (a partitioned data set).

```
ISPEXEC LIBDEF ISPPLIB DATASET ID('ISPFPROJ.ABC.APPAN1',
                                  'ISPFPROJ.ABC.APPAN2')
```

or alternately

```
CALL ISPLINK('LIBDEF  ', 'ISPPLIB ', 'DATASET ',
       '('ISPFPROJ.ABC.APPAN1','ISPFPROJ.ABC.APPAN2'')');
```

This example assumes that ISPFPROJ.ABC.MYPAN contains panels unique to the user. Panels unique to the application are contained in partitioned data sets ISPFPROJ.ABC.APPAN1 and ISPFPROJ.ABC.APPAN2.

The search sequence for panel APPLPAN1 is as follows:
1. Search for the member APPLPAN1 in ISPFPROJ.ABC.MYPAN
2. Search for the member APPLPAN1 in ISPFPROJ.ABC.APPAN1
3. Search for the member APPLPAN1 in ISPFPROJ.ABC.APPAN2
4. Search for the member APPLPAN1 in ISPFPROJ.ABC.PANELS

If the LIBDEF service had not been invoked, only ISPFPROJ.ABC.PANELS would have been searched for member APPLPAN1. The user library would not be searched.

To clear the LIBDEF after setting it, use either

```
'ISPEXEC LIBDEF ISPPLIB'
or
'ISPEXEC LIBDEF ISPPLIB DATASET()'
```

or additionally

```
CALL ISPLINK('LIBDEF  ', 'ISPPLIB ', ' ',' ');
or
CALL ISPLINK('LIBDEF  ', 'ISPPLIB ', 'DATASET ','()');
```

## Example 2 - The EXCLDATA keyword

Assume that the user has issued the following ALLOCATE statements for a user link library before entering ISPF:

```
ALLOCATE DATASET('ISPFPROJ.ABC.MYMOD') FILE(ISPLUSR) SHR
ALLOCATE DATASET('ISPFPROJ.ABC.LLOAD') FILE(ISPLLIB) SHR
```

Next, the LIBDEF service is invoked with the EXCLDATA keyword to define an application-level link library (a partitioned data set).

```
ISPEXEC LIBDEF ISPLLIB EXCLDATA ID('ISPFPROJ.ABC.APMOD1',
                                   'ISPFPROJ.ABC.APMOD2')
```

or alternately

```
CALL ISPLINK('LIBDEF  ', 'ISPLLIB ', 'EXCLDATA',
       '('ISPFPROJ.ABC.APMOD1','ISPFPROJ.ABC.APMOD2'')');
```

This example assumes that MYMOD contains programs or commands unique to the user. Programs unique to the application are contained in partitioned data sets ISPFPROJ.ABC.APMOD1 and ISPFPROJ.ABC.APMOD2.

The search sequence for program APPLMOD1 is as follows:
1. Search for the member APPLMOD1 in ISPFPROJ.ABC.APMOD1
2. Search for the member APPLMOD1 in ISPFPROJ.ABC.APMOD2

If the LIBDEF service had not been invoked, only ISPFPROJ.ABC.LLOAD would have been searched for member APPLMOD1. The user library would not be searched.

## Example 3 - The LIBRARY keyword

Assume the user has issued the following ALLOCATE statements for an application-level panel library before entering ISPF:

```
ALLOCATE DATASET('ISPFPROJ.ABC.APPAN1',
                 'ISPFPROJ.ABC.APPAN2') FILE(APPLIB) SHR
ALLOCATE DATASET('ISPFPROJ.ABC.MYPAN') FILE(ISPPUSR) SHR
ALLOCATE DATASET('ISPFPROJ.ABC.PANELS') FILE(ISPPLIB) SHR
```

Next, the LIBDEF service is invoked with the LIBRARY keyword to define an application-level panel libname.

```
ISPEXEC LIBDEF ISPPLIB LIBRARY ID(APPLIB)
```

or alternately

```
CALL ISPLINK('LIBDEF  ', 'ISPPLIB ', 'LIBRARY ', 'APPLIB ');
```

The search sequence, using the APPLIB definition, for panel APPLPAN1 is as follows:
1. Search for the member APPLPAN1 in ISPFPROJ.ABC.APPAN1
2. Search for the member APPLPAN1 in ISPFPROJ.ABC.APPAN2.

The search sequence, using the ISPPLIB definition, for panel APPLPAN1 is as follows:
- Search for the member APPLPAN1 in ISPFPROJ.ABC.PANELS.

If the LIBDEF service had not been invoked, only ISPFPROJ.ABC.PANELS would have been searched for APPLPAN1. The user library would not be searched.

## Example 4 - The EXCLLIBR keyword

Assume the user has issued the following ALLOCATE statements for an application-level link library before entering ISPF:

```
ALLOCATE DATASET('ISPFPROJ.ABC.APMOD1',
                 'ISPFPROJ.ABC.APMOD2') FILE(APLLIB) SHR
ALLOCATE DATASET('ISPFPROJ.ABC.MYMOD') FILE(ISPLUSR) SHR
ALLOCATE DATASET('ISPFPROJ.ABC.LLOAD') FILE(ISPLLIB) SHR
```

Next, the LIBDEF service is invoked with the EXCLLIBR keyword to define an application-level user link library.

```
ISPEXEC LIBDEF ISPLLIB EXCLLIBR ID(APLLIB)
```

or alternately

```
CALL ISPLINK('LIBDEF ', 'ISPLLIB ', 'EXCLLIBR', 'APLLIB ');
```

The search sequence for program APPLMOD1, using the APLLIB definition, is as follows:

1. Search for the member APPLMOD1 in ISPFPROJ.ABC.APMOD1
2. Search for the member APPLMOD1 in ISPFPROJ.ABC.APMOD2.

If the LIBDEF service had not been invoked, only ISPFPROJ.ABC.LLOAD would have been searched for APPLMOD1. The user library would not be searched.

## Example 5

Assume the following LIBDEF commands are executed:

```
ISPEXEC LIBDEF ISPPLIB
ISPEXEC LIBDEF ISPPLIB STACK

ISPEXEC LIBDEF ISPPLIB DATASET ID('ISPFPROJ.LWG.PANELS') STACK

ISPEXEC LIBDEF ISPPLIB DATASET ID('ISPFPROJ.LWGMVS33.PANELS') STACK
```

The execution of these commands produces the following results:

1. The first LIBDEF resets the ISPPLIB LIBDEF definition. This is considered a ″null″ definition for ISPPLIB.
2. The second LIBDEF stacks the previous ″null″ definition for ISPPLIB and resets the ISPPLIB LIBDEF definition. This is the second ″null″ definition for ISPPLIB.
3. The third LIBDEF stacks the previous ″null″ definition for ISPPLIB and establishes the ISPPLIB definition for data set 'ISPFPROJ.LWG.PANELS'.
4. The fourth LIBDEF stacks the previous ISPPLIB definition for data set 'ISPFPROJ.LWG.PANELS' and establishes the ISPPLIB definition for data set 'ISPFPROJ.LWGMVS33.PANELS'.

Next, the following LIBDEF service calls are issued:

```
ISPEXEC LIBDEF ISPPLIB        (restores 'ISPFPROJ.LWG.PANELS')
 Return code = 0

ISPEXEC LIBDEF ISPPLIB        (restores stacked "null" definition)

 Return code = 0

ISPEXEC LIBDEF ISPPLIB        (restores stacked "null" definition)

 Return code = 0
```

```
ISPEXEC LIBDEF ISPPLIB

 Return code = 4
```

The preceding service calls produce the following results:

1. The first LIBDEF reset restores the ISPPLIB definition for data set 'ISPFPROJ.LWG.PANELS'.
2. The second LIBDEF reset restores the stacked "null" definition for ISPPLIB. This is the "null" definition which issued the keyword, STACK.
3. The third LIBDEF restores the stacked "null" definition. This is the "null" definition which did not issue the keyword, STACK.
4. The fourth LIBDEF receives a return code of 4 because there is nothing in the stack and there is no active ISPPLIB definition.

## Example 6 — STKADD

Assume the following LIBDEF commands are executed:

```
ISPEXEC LIBDEF ISPPLIB
ISPEXEC LIBDEF ISPPLIB STACK

ISPEXEC LIBDEF ISPPLIB DATASET ID('ISPFPROJ.LWG.PANELS') STACK

ISPEXEC LIBDEF ISPPLIB DATASET ID('ISPFPROJ.ABC.PANELS') STKADD
```

The execution of these commands produces the following results:

1. The first LIBDEF resets the ISPPLIB LIBDEF definition. This is considered a "null" definition for ISPPLIB.
2. The second LIBDEF stacks the previous "null" definition for ISPPLIB and resets the ISPPLIB LIBDEF definition. This is the second "null" definition for ISPPLIB.
3. The third LIBDEF stacks the previous "null" definition for ISPPLIB and establishes the ISPPLIB definition for data set 'ISPFPROJ.LWG.PANELS'.
4. The fourth LIBDEF concatenates the data set 'ISPFPROJ.ABC.PANELS' ahead of the data set 'ISPFPROJ.LWG.PANELS' in the current ISPPLIB definition.

After the third LIBDEF service call the LIBDEF Display Utility would show:

```
                              LIBDEF Utility
                            ISPF LIBDEF Display            Row 1 to 11 of 11
  Command ===>                                            Scroll ===> PAGE

  Library   Type          USR Identifier
  ISPFILE                     ** LIBDEF not active **
  ISPILIB                     ** LIBDEF not active **
  ISPLLIB                     ** LIBDEF not active **
  ISPMLIB                     ** LIBDEF not active **
  ISPPLIB   DATASET           ISPFPROJ.LWG.PANELS
S ISPPLIB                     ** LIBDEF not active **
S ISPPLIB                     ** LIBDEF not active **
  ISPSLIB                     ** LIBDEF not active **
  ISPTABL                     ** LIBDEF not active **
  ISPTLIB                     ** LIBDEF not active **
```

After the fourth LIBDEF service call the LIBDEF Display Utility would show:

```
                              LIBDEF Utility
                             ISPF LIBDEF Display            Row 1 to 11 of 11
  Command ===>                                             Scroll ===> PAGE

  Library   Type          USR Identifier
  ISPFILE                     ** LIBDEF not active **
  ISPILIB                     ** LIBDEF not active **
  ISPLLIB                     ** LIBDEF not active **
  ISPMLIB                     ** LIBDEF not active **
  ISPPLIB   DATASET           ISPFPROJ.ABC.PANELS
                              ISPFPROJ.LWG.PANELS
S ISPPLIB                     ** LIBDEF not active **
S ISPPLIB                     ** LIBDEF not active **
  ISPSLIB                     ** LIBDEF not active **
  ISPTABL                     ** LIBDEF not active **
  ISPTLIB                     ** LIBDEF not active **
```

Next, the following LIBDEF service calls are issued:

```
ISPEXEC LIBDEF ISPPLIB  (restores stacked "null" definition)
  Return code = 0
ISPEXEC LIBDEF ISPPLIB  (restores stacked "null" definition)
  Return code = 0
ISPEXEC LIBDEF ISPPLIB
  Return code = 4
```

## LIST – Write Lines to the List Data Set

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. The amount of data that can be written with one LIST request is one or more lines totaling up to 32 767 bytes, the maximum size of the dialog variable.

The list data set, if allocated, is normally processed when you exit ISPF. A LIST command is available to allow you to process the list data set without exiting ISPF.

### Command Invocation Format

```
ISPEXEC  LIST   BUFNAME(dialog-variable-name)
                LINELEN(line-length)

                [PAGE]

                [SINGLE|DOUBLE|TRIPLE]

                [OVERSTRK]

                [CC]
```

### Call Invocation Format

```
CALL  ISPEXEC (buf-len, buffer);
OR

CALL  ISPLINK   ('LIST    ', dialog-variable-name, line-length

                         ['PAGE    ']

                         ['SINGLE  '|'DOUBLE  '|'TRIPLE  ']

                         ['OVERSTRK']

                         ['CC      ']);
```

## Parameters

**dialog-variable-name**
Specifies the name of the dialog variable that contains the text (32,767 bytes maximum) to be written to the list data set.

**line-length** Specifies the length of each line in the buffer being passed to ISPF. ISPF truncates these lines if the line-length specified is greater than the truncation value in system variable ZLSTTRUN. The line-length must have an unsigned integer value and, for a call, must be a fullword fixed integer.

**PAGE** Specifies that the first data line of this LIST service request is to be written to the list data set preceded by a page eject carriage control character. The spacing of the remaining lines is determined by the SINGLE, DOUBLE, or TRIPLE keyword specified. PAGE is ignored if the CC keyword is specified.

**SINGLE** Specifies that each line of data is to be written to the list data set preceded by a single space carriage control character. SINGLE is the default line spacing keyword value. SINGLE is ignored if the CC keyword is specified.

**DOUBLE** Specifies that each line of data is to be written to the list data set preceded by a double space carriage control character. DOUBLE is ignored if the CC keyword is specified.

**TRIPLE** Specifies that each line of data is to be written to the list data set preceded by a triple space carriage control character. TRIPLE is ignored if the CC keyword is specified.

**OVERSTRK** Specifies that each line of data is to be written with overstrikes. That is, the line is first written with the line spacing specified, then written again with the line spacing suppressed. This allows a dialog to request text highlighting on printed output. OVERSTRK is ignored if the CC keyword is specified.

**CC** Specifies that carriage control is to be provided by the dialog as the first byte of each data line. Specifying CC nullifies specification of the PAGE, SINGLE, DOUBLE, TRIPLE, or OVERSTRK keyword. If CC is specified, the value specified for line-length should include one byte for the carriage control character.

**buf-len** Specifies a fullword fixed binary integer containing the length of "buffer".

**buffer** Specifies a buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC call for a command procedure.

## Return Codes

The following return codes are possible:

**0** Normal completion.

**8** Maximum line length or data set LRECL exceeded; data has been truncated.

**12** Specified dialog variable not found.

**20** Severe error.

## Formatting Data to be Written to the List Data Set

ISPF writes data to the list data set exactly as received in the dialog variable, which acts as the data buffer. The dialog must provide any data formatting or centering before passing the data to ISPF. The length of each data line passed to ISPF is the value of the line-length parameter specified on the LIST service request. If the line-length value is greater than or equal to the length of the passed data, ISPF writes the data as a single line in the list data set. If the line-length value is less than the length of the passed data, ISPF writes the data in multiple lines. If the line-length value specified is zero and CC is not specified, ISPF writes one blank line to the list data set. If CC is specified, the line length specified must be at least one (to accommodate the carriage control character); otherwise, a severe error results.

## List Data Set Characteristics Affect the LIST Service

The dialog user can specify the logical record length (LRECL) and maximum line length values for the list data set by using SETTINGS option 0. ISPF uses these two values in determining where truncation of lines written to the list data set is to occur.

The value in system variable ZLSTTRUN defines where ISPF is to truncate lines written to the list data set. This value is not directly alterable by the dialog. The value in ZLSTTRUN is the lesser of:

1. LRECL minus 1 (fixed-record format data sets) or LRECL minus 5 (variable-record format data sets)

   The logical record length can be established for the list data set before the ISPF session (by preallocating the data set), or, if that is not the case, it can be specified on SETTINGS option 0.
2. LINE LENGTH - Default value specified on SETTINGS option 0.

## Controlling Line Spacing, Page Eject, and Highlighting

Line spacing and page ejects can be under control of either the dialog or ISPF. If the dialog specifies CC on the LIST service request, the dialog controls all carriage functions, using the first byte of each line passed to ISPF as a carriage-control character. Therefore, when CC is specified on the LIST service request, ISPF ignores any SINGLE, DOUBLE, TRIPLE, PAGE, and OVERSTRK keywords.

ISPF causes an automatic page eject (regardless of CC keyword status) for a LIST service request that causes information to be written to a list data set for the first time in the session.

### How ISPF Controls Printer Functions (CC Not Specified)

When the dialog does *not* specify CC on the LIST service request, ISPF appends a carriage control byte ahead of each line to be written to the list data set.

The dialog can include the SINGLE, DOUBLE, or TRIPLE keyword on the LIST service request to tell ISPF how lines are to be spaced when written to the list data set. Single spacing is the default value. The dialog can also specify, along with the line spacing keyword, the OVERSTRK keyword on the LIST service to cause highlighting.

The optional PAGE keyword on the LIST request tells ISPF that the first line written by this request is to include a page eject control character. Thereafter, page ejects are caused by:

- ISPF providing the page eject carriage control when the lines-per-page value (1 to 999) in system variable ZLSTLPP is reached, or
- The dialog specifying the PAGE keyword on a subsequent LIST service request.

### How the Dialog Controls Printer Functions (CC Specified)

When the dialog specifies CC on the LIST service request, ISPF ignores any other printer control keywords. ISPF then relies on the dialog to supply the printer control information as the first byte of each line in the data buffer to be written. ISPF does not check the validity of the characters included for carriage control.

### Using System Variables ZLSTNUML and ZLSTLPP

**ZLSTNUML**

This 4-byte shared pool variable contains the number of lines that have been written to the current page in the list data set. If the list data set is not open the value in ZLSTNUML is zero.

ZLSTNUML is set by ISPF and is not directly alterable by a dialog.

**ZLSTLPP**

This 4-byte shared pool variable contains the value that specifies what the maximum number of lines per page written to the list data set is to be.

You can set the value in ZLSTLPP (lines-per-page) by using SETTINGS option 0. ZLSTLPP is not directly alterable by a dialog.

Dialogs that provide carriage control characters can test variables ZLSTNUML and ZLSTLPP for values to determine when printing should begin on a new page.

The ANSI-defined carriage control characters in the chart below are recognized by the LIST service for updating (incrementing the number of page line spaces used) the value of ZLSTNUML. If the dialog passes any other carriage control character along with the CC keyword, the character is written to the list data set, but does not affect the value of ZLSTNUML.

The carriage control characters, whether supplied to ISPF with each line to be printed or supplied by ISPF, cause the actions listed in the chart below:

| Character | Action (before printing) | ZLSTNUML is |
|---|---|---|
| blank | Space 1 line | Incremented 1 |
| 0 | Space 2 lines | " 2 |
| - | Space 3 lines | " 3 |
| + | Suppress spacing | Not changed |
| 1 | Skip to line 1 on new page | Set to 1 |

## How Carriage Control Characters Affect Truncation

ISPF counts only data characters, not the carriage-control character, in calculating the point at which truncation is to occur. A dialog can determine what the truncation value is by querying system variable ZLSTTRUN in the shared variable pool.

The carriage-control byte must be taken into account when calculating where truncation will occur. For example, assume that the truncation value in ZLSTTRUN is 79, indicating that a maximum of 79 characters per list data set line, not

including carriage-control, are allowed. Also, assume the dialog passes a line of 80 characters to be written to the list data set. Truncation is as follows:

- If the dialog has specified the CC (carriage-control) keyword on the LIST request, the first byte in the line passed to ISPF is the carriage-control character, followed by 79 data characters. Because ISPF does not count the carriage-control character as one of the truncation value (79), no truncation occurs.
- If the dialog has not specified the CC keyword, ISPF appends the carriage-control byte ahead of the line of 80 data characters passed by the dialog. In this case, the truncation value of 79 causes one data character to be truncated.

## Example 1

Using three LIST service requests, write three lines, containing the text 'Line 1', 'Line 2', and 'Line 3' respectively, to the list data set. The text is to start at the top of a new page, and be double spaced.

In preparation:
- Set dialog variable LINE1 to the value 'Line 1'
- Set dialog variable LINE2 to the value 'Line 2'
- Set dialog variable LINE3 to the value 'Line 3'

Then issue:

```
ISPEXEC LIST BUFNAME(LINE1) LINELEN(6) PAGE
ISPEXEC LIST BUFNAME(LINE2) LINELEN(6) DOUBLE
ISPEXEC LIST BUFNAME(LINE3) LINELEN(6) DOUBLE
```

or alternately

Set variable LEN to 6 and issue:

```
CALL ISPLINK ('LIST    ','LINE1 ',LEN,'PAGE    ');
CALL ISPLINK ('LIST    ','LINE2 ',LEN,' ','DOUBLE  ');
CALL ISPLINK ('LIST    ','LINE3 ',LEN,' ','DOUBLE  ');
```

## Example 2

Write the same three lines as in Example 1, but with one LIST service request.

In preparation, set dialog variable LSTTEXT to the value:

```
'Line 1Line 2Line 3'
```

Then issue:

```
ISPEXEC LIST BUFNAME(LSTTEXT) LINELEN(6) PAGE DOUBLE
```

or alternately

Set variable LEN to 6 and issue:

```
CALL ISPLINK ('LIST    ','LSTTEXT ',LEN,'PAGE    ','DOUBLE  ');
```

## Example 3

Write the same three lines as in the previous examples, but with the carriage control characters being passed to ISPF.

In preparation, set dialog variable LSTTEXT to the value:

```
'1Line 10Line 20Line 3'
```

The characters '1' and '0' preceding the word 'Line' in LSTTEXT are carriage control characters for page eject and double space respectively.

Then issue:

```
ISPEXEC LIST BUFNAME(LSTTEXT) LINELEN(7) CC
```

or alternately

Set variable LEN to 7 and issue:

```
CALL ISPLINK ('LIST    ','LSTTEXT ',LEN,' ',' ',' ','CC      ');
```

Note that the line-length value has been increased by one to account for the carriage control byte.

### Example 4

Print the same three lines as in Example 3. This time, assume that ZLSTTRUN has a value of 5. In preparation, set up conditions to cause the value of ZLSTTRUN to be 5. This value is the lesser of:

- The logical record length of the list data set minus one (fixed format) or the record length minus five (variable format).
- The value specified for list data set line length using SETTINGS option 0.

LSTTEXT is set the same way, and the LIST request issued the same way, as for Example 3. The difference in data written to the list data set for Example 4 compared to Example 3 illustrates the truncation:

```
  Example 3     Example 4
1Line 1        1Line

0Line 2        0Line

0Line 3        0Line
```

## LMCLOSE – Close a Data Set

The LMCLOSE service closes the data set associated with a given data ID. For each LMOPEN invocation, you should invoke a matching LMCLOSE service when processing is complete. Otherwise, unwanted data can be read from or written to the data set.

If LMINIT is issued with an enqueue (ENQ) of SHRW and LMOPEN is issued with the OUTPUT option, it is important that an LMCLOSE be issued when the dialog has finished processing the data set, since the DASD volume is reserved until LMCLOSE is invoked. On output, if the data is sequential, the LMCLOSE service writes the last physical block.

### Command Invocation Format

```
ISPEXEC LMCLOSE DATAID(data-id)
```

### Call Invocation Format

```
CALL ISPLINK ('LMCLOSE ',data-id);
OR

CALL ISPEXEC (buf-len, buffer);
```

## Parameters

**data-id** The data ID associated with the data set to be closed. The data ID is generated by the LMINIT service. The maximum length of this parameter is 8 characters.

**buf-len** A fullword fixed binary integer containing the length of the buffer parameter.

**buffer** A buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC invocation for a command invocation.

## Return Codes

The following return codes are possible:

**0** Normal completion.

**8** Data set is not open.

**10** No ISPF library or data set associated with the given data ID; that is, LMINIT has not been completed.

**20** Severe error; unable to continue.

## Example

This example invokes the LMCLOSE service to close the data set associated with the data ID in variable DDVAR.

### Command Invocation

```
ISPEXEC LMCLOSE DATAID(&DDVAR)
```

### Call Invocation

```
CALL ISPLINK('LMCLOSE ',DDVAR);
OR
```

Set the program variable BUFFER to contain:

```
BUFFER = 'LMCLOSE DATAID(&DDVAR)';
```

Set the program variable BUFLEN to the length of the variable BUFFER. Issue the following:

```
CALL ISPEXEC (BUFLEN, BUFFER);
```

# LMCOMP – Compresses a Partitioned Data Set

The LMCOMP service compresses a data set. The installation-supplied compress exit is used, or, if there is no exit, IEBCOPY is used. Completion of the LMINIT service specifying ENQ(EXCLU) is required before you invoke LMCOMP.

## Command Invocation Format

```
ISPEXEC LMCOMP DATAID(data-id)
```

## Call Invocation Format

```
CALL ISPLINK ('LMCOMP  ',data-id);
OR

CALL ISPEXEC (buf-len,buffer);
```

## Parameters

**data-id** The data ID associated with the data set to be compressed. The data ID has been generated by the LMINIT service. The data ID must be associated with only one data set. Concatenations are not allowed. The maximum length of this parameter is 8 characters.

**buf-len** A fullword fixed binary integer containing the length of the buffer parameter.

**buffer** A buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC invocation for

## Return Codes

The compress request exit routine is responsible for

# LMCOPY – Copy Members of a Data Set

The LMCOPY service copies members of a partitioned data set, or copies an entire sequential data set. Packing data, locking members, replacing members, and automatic truncation are optional. Only fixed- and variable-record format data sets can be packed.

Completion of the LMINIT service is required before you can invoke LMCOPY. See "LMINIT – Generate a Data ID for a Data Set" on page 119 for information that can help prevent some common I/O errors that might occur when using the LMCOPY service. LMCOPY requires that the "to data-id" and "from data-id" be closed before invocation.

**Notes:**

1. FROMID and TODATAID can refer to the same data set but they cannot have the same data-id.
2. LMCOPY does not support the copying of unmovable data sets (data set organization POU or PSU).
3. If the ALIAS option is in effect, LMCOPY automatically processes alias members as follows:
   - Either the main member or any alias member may be selected to copy the main member and all of its aliases. This will occur even if some of the members are not displayed in the current member selection list.
   - Alias members are copied for both load and non-load data sets as well as for PDS and PDSE data sets.

   Copying to the same dataset is not supported when aliases are automatically selected, as it would result in the "from" and "to" member names being the same.
4. If the NOALIAS option is in effect, LMCOPY does not copy alias members unless one of the following is true:
   - All members of the data set are selected.
   - A member pattern is used and both the main member and the alias member are included in that pattern.

   If the NOALIAS option is in effect, copying an alias member by itself will result in a new member being created, even if the main member has already been copied.

## Command Invocation Format

```
ISPEXEC LMCOPY FROMID(from-data-id)
               [FROMMEM(from-member-name)]

               TODATAID(to-data-id)

               [TOMEM(to-member-name)]

               [REPLACE]

               [PACK]

               [TRUNC]

               [LOCK]

               [SCLMSET(Y|N)]

               [ALIAS|NOALIAS]
```

## Call Invocation Format

```
CALL ISPLINK ('LMCOPY  ', from-data-id
                          ,[from-member-name]
                          ,to-data-id
                          ,[to-member-name]
                          ,['REPLACE ']
                          ,['PACK    ']
                          ,['TRUNC   ']
                          ,['LOCK    ']
                          ,['YES'|'NO'|' ']
                          ,['ALIAS   '|'NOALIAS ']);
```

*OR*

```
CALL ISPEXEC (buf-len,buffer);
```

## Parameters

**from-data-id**
The data ID associated with the data set to be copied. The data ID has been generated by the LMINIT service. The maximum length of this parameter is 8 characters.

**from-member-name**
The member name or pattern of the members to be moved. An asterisk (*) indicates that all members are to be moved. If the “from” data set is partitioned, this parameter is required. If it is sequential, this parameter is not allowed. The maximum length of this parameter is 8 characters.

**to-data-id**
The data ID associated with the data set to be copied to. The data ID has been generated by the LMINIT service. The maximum length of this parameter is 8 characters.

**to-member-name**
The name of the member being moved to the “to” data set. If a name is not specified, the name of the member in the “from” data set is used. If the “from” data set is sequential and the “to” data set is partitioned, this parameter is required. If the “to” data set is sequential, this parameter is not allowed. The maximum length of this parameter is 8 characters.

**REPLACE**
Like-named members in the “to” data set are to be replaced. If this parameter is not specified and a like-named member exists in the “to” data set, the copy function is performed on all other members except like-named members, and a return code of 12 is issued.

If a list of members is being copied and one cannot be replaced, a message is issued indicating how many members were copied and how many were not replaced.

**PACK**
Data is stored in the "to" data set in packed format. If this parameter is not specified, data is copied and stored as unpacked.

**TRUNC**
Truncation is to occur if the logical record length of the "to" data set is less than the logical record length of the "from" data set. If this parameter is not specified and the logical record length of the "to" data set is less than the logical record length of the "from" data set, the copy is not performed and a return code of 16 is issued.

**LOCK**
The LOCK parameter is no longer used since the removal of LMF from the ISPF product, but is left in for compatibility. If LOCK is specified, the LMCOPY service will fail with return code 12. If you want to be able to specify YES and have the LMCOPY ignore the value, change the FAIL_ON_LMF_LOCK keyword value in the ISPF Configuration Table to NO.

The members are to be locked. The data set associated with the from-data-id must be a LMF-controlled ISPF library or partitioned data set. The member is locked under the logon ID of the user performing the copy and lock. No other user can change that member in the LMF-controlled library until the current owner promotes it. If a member cannot be locked it is not copied.

If a list of members is being copied and one cannot be locked, processing stops and a message is issued indicating how many members were copied.

**SCLMSET**
ISPF maintains a bit in the PDS directory to indicate whether a member was last modified using SCLM or some function outside of SCLM. The SCLMSET value indicates how to set this bit. YES indicates to set the bit ON. NO indicates the bit should be OFF. If you want to keep the current setting for a certain member, omit the SCLMSET parameter.

**ALIAS | NOALIAS**
With ALIAS in effect, either the main member or any alias member may be selected to copy the main member and all of its aliases. This will occur even if a single member is specified or if some of the members are not displayed in the current member selection list.

With NOALIAS in effect, aliases must be copied manually to maintain the correct alias relationship. That is, the main member must be copied first followed by the aliases.

**buf-len**
A fullword fixed binary integer containing the length of the buffer parameter.

**buffer**
A buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC invocation for a command procedure.

## Return Codes

The following return codes are possible:

**0** Normal completion.

**4** Member not available, which indicates one of the following situations:
- The "From" data set is empty.
- No members matched the specified pattern in the "from" data set.

**8**

- The “From” member not found.
- Specified “To member name” same as specified “From member name”.

**10** No data set is associated with the given data ID.

**12** One of the following:

- A like-named member already exists in the ′TO′ data set and the Replace option was not specified.
- One or more members of the ′TO′ data set are ″in use″ by you or another user and could not be copied
- Invalid data set organization
- Data set attribute invalid for copying or copying packed data
- Open error.
- LOCK parameter is specified.

**16** Truncation error.

**20** Severe error; unable to continue.

## Example

This example invokes the LMCOPY service to copy all member names beginning with the letter ′L′ in the data set associated with the data ID in variable DDVAR to the data set associated with the data ID in variable DDVAR2. Like-named members in the “to” data set are replaced, the data is packed, and truncation will occur if necessary.

### Command Invocation

```
ISPEXEC LMCOPY FROMID(&DDVAR) FROMMEM(L*)             +
               TODATAID(&DDVAR2) REPLACE PACK TRUNC
```

### Call Invocation

```
CALL ISPLINK('LMCOPY  ',DDVAR,'L* ',DDVAR2,' ', 'REPLACE ',
                 'PACK    ','TRUNC   ');
```

*OR*

Set the program variable BUFFER to contain:

```
BUFFER = 'LMCOPY FROMID(&DDVAR) FROMMEM(L*)
         TODATAID(&DDAVAR2) REPLACE PACK TRUNC';
```

Set the program variable BUFLEN to the length of the variable BUFFER. Issue the following command:

```
CALL ISPEXEC (BUFLEN, BUFFER);
```

# LMDDISP – Data Set List Service

The LMDDISP service allows you to write your own data set list dialog. This service is similar to ISPF option 3.4, the data set list utility, which displays the list of data sets. The LMDDISP service displays any view of the data set list (Volume, Space, Attrib, or Total) that you want to display first. You can then scroll to any other view from the initial display view.

The LMDDISP service is given a data set list ID (dslist-id) which has been associated with a data set level or volume or both by the LMDINIT service. The

LMDINIT generates a data set list ID from a data set level or volume or both. The data set list ID must be freed by the LMDFREE service.

## Command Invocation Format

```
ISPEXEC LMDDISP LISTID(dslist-id)
                [VIEW(VOLUME|SPACE|ATTRIB|TOTAL)]
                [CONFIRM(YES|NO)]
                [PANEL(panel-name)]
                [CATALOG(Yes|NO)]
```

## Call Invocation Format

```
CALL ISPEXEC (buf-len,buffer);
OR

CALL ISPLINK('LMDDISP ', dslist-id
                       , ['VOLUME  '|'SPACE   '|'ATTRIB '|'TOTAL   ']
                       , ['YES     '|NO      ']
                       , [panel-name]);
                       , ['Yes '|'NO ']);
```

## Parameters

**dslist-id**
A data set list ID associated with a data set name level or a volume or both by the LMDINIT service. For more information about the data set level and how it determines which data set names are to be included in the data set list, see "LMDINIT – Initialize a Data Set List" on page 107.

**VOLUME | SPACE | ATTRIB | TOTAL**
The Volume view shows a data set list that contains data set names and the volumes on which they reside. The list is sorted by data set name. Volume is the default.

The Space view shows a data set list that contains data set names, tracks, percentages used, extents, and devices. The list is sorted by data set name.

The Attrib view shows a data set list that contains data set names, data set organizations, record formats, logical record lengths, and block sizes. The list is sorted by data set name.

The Total view shows a data set list that contains all information displayed by the Volume, Space, and Attrib views, plus the created, expired, and referred dates. The list is sorted by data set name and has two lines per data set.

**YES | NO**
This parameter controls whether the Confirm Delete panel appears when using the D (delete data set) line command from the displayed data set list. YES is the default.

If YES is specified, ISPF displays the Confirm Delete panel. This gives you the opportunity to change your mind and keep the data set. If you try to delete an expired data set, the Confirm Purge panel appears following the Confirm Delete panel.

If NO is specified, ISPF does not display the Confirm Delete panel. The data set is deleted without your having to take any additional actions unless you try to delete an unexpired data set. If this is the case, the Confirm Purge panel appears.

**panel-name**
The name of the panel to use for displaying a data set list. The default is the data set list found in option 3.4, the data set list utility. This can be a customized panel that you provided. Refer to *ISPF Planning and Customizing* for more information on developing a customized panel.

**YES | NO**
This parameter controls whether the name of the catalog where each data set was located is displayed on the Total view. This parameter is ignored if a value for the volume-serial parameter is passed on the LMDINIT call.

## Return Codes

The following return codes are possible:

**0** Normal completion.

**8** Error building data set list. The error condition is described in the ISPF system dialog variables.

**10** A data set list does not exist for the list-id specified via keyword LISTID.

**12** A keyword value is incorrect.

**20** A severe error occurred while processing the data set list.

## Example

The following example shows an invocation of LMDDISP which will display the Volume view of a data set list with the Delete Data Set Confirmation panel. The variable ID contains a data set list ID generated by the LMDINIT service.

### Command Invocation

```
ISPEXEC LMDDISP LISTID('id') VIEW(VOLUME) CONFIRM(YES)
```

### Call Invocation

```
CALL ISPLINK('LMDDISP ',DSLISTID,'TOTAL   ','NO      ');
OR
```

Set the program variable BUFFER to contain:

```
BUFFER = 'LMDDISP LISTID('id') VIEW(VOLUME) CONFIRM(YES)';
```

Set the program variable BUFLEN to the length of the variable BUFFER. Issue the following:

```
CALL ISPEXEC (BUFLEN, BUFFER);
```

# LMDFREE – Free a Data Set List ID

The data set list free service (LMDFREE) removes a data set list ID (dslist ID) generated by the data list initialize service (LMDINIT).

## Command Invocation Format

```
ISPEXEC LMDFREE LISTID(list-id)
```

## Call Invocation Format

```
CALL ISPLINK ('LMDFREE ', list-id);
OR

CALL ISPEXEC (buf-len, buffer);
```

## Parameters

**list-id** The LMDFREE service removes this dslist ID from the list of dslist IDs. The LMDLIST and LMDFREE service cannot use the dslist ID for the remainder of the TSO session.

**buf-len**
A fullword fixed binary integer containing the length of the buffer parameter.

**buffer** A buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC invocation for a command invocation.

## Return Codes

**0** Normal completion.

**8** Free dslist ID failed. The error condition is described in "System Variables Used to Format Error Messages" on page 14.

**10** No data set level or volume is associated with given dslist ID. LMDINIT has not been completed.

**20** Severe error; unable to continue.

## Example

In this example the LMDFREE service frees a dslist ID stored in function pool variable ID.

### Command Invocation

```
ISPEXEC    LMDFREE LISTID(&ID)
```

### Call Invocation

```
CALL ISPLINK   ('LMDFREE ', ID);
OR
```

Set the program variable BUFFER to contain:

```
BUFFER = 'LMDFREE LISTID(&ID)';
```

Set the program variable BUFLEN to the length of the variable BUFFER. Issue the following command:

```
CALL ISPEXEC (BUFLEN, BUFFER);
```

# LMDINIT – Initialize a Data Set List

The LMDINIT service generates a data set list ID for a given data set level or volume or both. A dialog uses the dslist ID to obtain a list of data sets and data set information from the data set list service (LMDLIST). The LMDINIT service is similar to the function provided by the LMINIT service.

To use LMDINIT, you must specify the following:
- A data set level or volume or both
- The name of the variable for LMDINIT to place the new dslist ID.

Each use of the LMDINIT service must eventually be followed by the LMDFREE service to release the dslist ID and the data set list storage space.

## Command Invocation Format

```
ISPEXEC LMDINIT       LISTID(dslist-id-var)
                      {LEVEL(dsname-level)}
                      {VOLUME(volume-serial)}
```

## Call Invocation Format

```
CALL ISPLINK ('LMDINIT '  ,dslist-id-var
                          ,{dsname-level}
                          ,{volume-serial});
OR

CALL ISPEXEC (buf-len, buffer);
```

## Parameters

**dslist-id-var**
The name of the ISPF function pool variable that stores the dslist ID of the data set list. The LMDINIT service always generates a unique dslist ID. The dslist ID is an input variable to the other library access services that work with data sets, and is an output parameter from the LMDINIT service. The maximum length for the dslist ID is 8 characters.

To invoke the service, you must specify the dslist ID variable name and Level, Volume, or both. The Level and Volume are described below.

In the LMDINIT service, dslist-id-var is the name of the variable that stores the data ID (for example, LISTID(DDVAR)). When you use the dslist ID keyword with other services, you must pass the value of the variable (for example, LISTID(&DDVAR)).

**dsname-level**
You may use this value to specify the level or levels of data sets displayed with the dslist ID. The dsname-level is a string containing valid TSO data set name qualifier patterns, separated by periods ('.'). You can use asterisks and percent signs as wildcards in the qualifiers. The LMDINIT service does not select data sets with fewer levels than the dsname-level. You may also use an optional data set list exit to control which data sets are included in the list.

**volume-serial**
Use this value to specify the volume serial of the VTOC that ISPF will use to generate the list of data sets. This field has the same restrictions and syntax as the Volume field under ISPF, option 3.4. Refer to the *ISPF User's Guide Volume II* for a complete description.

## Return Codes

**0** Normal completion. LMDINIT returns a unique dslist ID in the variable specified in keyword LISTID.

**8** The dslist ID was not created; the error condition is described in "System Variables Used to Format Error Messages" on page 14.

**12** A keyword value is incorrect.

**16** A truncation or translation error occurred in accessing dialog variables.

**20** Severe error; unable to continue.

## Example

In this example the LMDINIT service generates a dslist ID for a data set list containing only the data sets in volume APL001. The LMDINIT service places the dslist ID in variable VARNAME in the ISPF function pool.

### Command Invocation

```
ISPEXEC LMDINIT LISTID(VARNAME) VOLUME(APL001);
```

### Call Invocation

```
DCL DSVAR CHAR(8)
CALL ISPLINK   ('VDEFINE ', 'DSVAR ', DSVAR, 'CHAR    ',L8);

CALL ISPLINK   ('LMDINIT ', 'DSVAR ', ' ', 'APL001 ');
```

*OR*

Set the program variable BUFFER to contain:

```
BUFFER = 'LMDINIT LISTID(VARNAME) VOLUME(APL001)';
```

Set the program variable BUFLEN to the length of the variable BUFFER. Issue the following command:

```
CALL ISPEXEC (BUFLEN, BUFFER);
```

In this example the LMDINIT service generates a dslist ID for a data set list containing only the data sets with the first level qualifier “PROD” and a second level qualifier starting with “ABC”.

### Command Invocation

```
ISPEXEC LMDINIT LISTID(VARNAME) LEVEL(PROD.ABC*);
```

### Call Invocation

```
DCL DSVAR CHAR(8)
CALL ISPLINK   ('VDEFINE ', 'DSVAR ', DSVAR, 'CHAR    ',L8);

CALL ISPLINK   ('LMDINIT ', 'DSVAR ', 'PROD.ABC* ');
```

*OR*

Set the program variable BUFFER to contain:

```
BUFFER = 'LMDINIT LISTID(VARNAME) LEVEL(PROD.ABC*)';
```

Set the program variable BUFLEN to the length of the variable BUFFER. Issue the following command:

```
CALL ISPEXEC (BUFLEN, BUFFER);
```

# LMDLIST – List Data Sets

The data set list service (LMDLIST) generates and uses an internal list of data set names associated with a unique data set list ID (dslist ID) obtained from the LMDINIT service.

The names in the internal list can be passed to a dialog with data set information (if specified) using two options:

- The LIST option returns the information one name at a time through the function pool variables.
- The SAVE option writes the names and information to a data set.

The internal list is not dynamic. Data sets created after the invocation of the LMDINIT service will not added to the list. To update the list to include new data set names, use the LMDFREE service to release the current dslist ID and reissue the LMDINIT and LMDLIST services, or reissue the LMDINIT and LMDLIST services using a different dslist ID.

## Command Invocation Format

```
ISPEXEC LMDLIST LISTID(dslist-id)
                [OPTION(LIST|FREE|SAVE|SAVEC)]
                [DATASET(dataset-var)]
                [STATS(YES|NO)]
                [GROUP(group)]
```

## Call Invocation Format

```
CALL ISPLINK ('LMDLIST '    , dslist-id
                            , ['LIST    '|'FREE    '|'SAVE    '|'SAVEC   ']
                            , [dataset-var]
                            , ['YES     '|'NO      ']
                            , [group )];
OR

CALL ISPEXEC (buf-len, buffer);
```

## Parameters

**dslist ID**
A data set list ID associated with a data set name level or volume or both by the LMDINIT service. For information about the data set level and how data set names are included in the data set list see "LMDINIT – Initialize a Data Set List" on page 107.

**LIST | FREE | SAVE | SAVEC**
These options determine whether the LMDLIST service returns the names on the internal list to the dialog, frees the storage used by the list, or writes the list to a data set.

**LIST** When you use the LMDLIST LIST option for the first time, the LMDLIST service generates an internal list. If you initialize the dataset-var to blanks, the first name in the internal list is returned. If you set the dataset-var to a data set name, that data set name is returned in dataset-var. If the LMDLIST service does not find the named data set the next data set in the list is returned. Each time you use the LMDLIST service with the LIST option it returns the next name from the internal list until it reaches the end of data. The LMDLIST service only includes the data set names meeting the criteria you specify at the time you invoke the LMDINIT service.

**FREE** The FREE option releases the storage associated with the data set list. Each use of the LMDLIST service with the LIST option must eventually be followed by the LMDLIST service with the FREE option.

**SAVE** The SAVE option writes all data set names associated with the dslist ID to a data set. The name of the data set is determined by the presence

and value of the group parameter. You cannot use the SAVE option after the use of the LIST option without first invoking the LMDLIST FREE option.

**SAVEC**
The SAVEC option is the same as the SAVE option and also requests to have the catalog name associated with each data set written to the output data set. The catalog name will not be written if a value for the volume-serial parameter is passed on the LMDINIT call.

**dataset-var**
The LMDLIST service uses this variable to establish a position in the list. To start at the beginning of the list set the dataset-var to blanks. To start at a specific data set in the list set the dataset-var to the name of the data set. If the LMDLIST service does not find the data set you specify, it returns the next data set in the list.

**YES | NO**
Use the STATS parameter with the LMDLIST service LIST and SAVE options. The default is STATS(NO). If you specify STATS(YES) the LMDLIST service provides statistical information with the data set names in the following dialog variables in the function pool:

| | |
|---|---|
| ZDLVOL | Volume serial. |
| ZDLDEV | Device type. |
| ZDLDSORG | Data set organization. |
| ZDLRECFM | Record format. |
| ZDLLRECL | Logical record length. |
| ZDLBLKSZ | Block size. |
| ZDLSIZE | Data set size in tracks. |
| ZDLUSED | Percentage of used tracks or pages (PDSE). |
| ZDLEXT | Number of extents used. |
| ZDLCDATE | Creation date. |
| ZDLEDATE | Expiration date. |
| ZDLRDATE | Date last referenced. |
| ZDLMIGR | Whether the data set is migrated (YES or NO) based on the value of the Volume_of_migrated_data_sets keyword in the ISPF configuration table. If the volume name of the data set matches the value of Volume_of_migrated_data_sets, ZDLMIGR is set to YES, otherwise it is set to NO. |
| ZDLDSNTP | Dsname type (PDS, LIBRARY, or ' '). |
| ZDLSPACU | Space units. |
| ZUDMVOL | Whether the data set is multivolume (Y) or not (N). |

**group**
This 8-character value specifies the group name of the data set that the LMDLIST service writes to when you use the SAVE option. The entire name of the data set name is `'<userid>.<group>.DATASETS'` if your userid and TSO data set name are the same, otherwise it is `'<prefix>.<userid>.<group>.DATASETS'`. If you do not specify a group name, the LMDLIST service writes to the ISPF list data set.

**Note:** LMDLIST service allocates the output data set with a DISP=OLD for the SAVE option.

## Return Codes

The following return codes are possible:

**0** One of the following:
- LIST option - Normal completion. The name of next data set in the list is returned in the variable specified in keyword DATASET. Data set statistics are returned, if requested.
- FREE option - Normal completion. The internal storage associated with the data set list has been freed.
- SAVE option - Normal completion. The data set list has been successfully written to a data set.

**4** No data sets matched specified search criteria (the values for keywords LEVEL and VOLUME on the LMDINIT service).

**8** End of data set list.

**10** The data set list does not exist for dslist ID.

**12** A keyword value is incorrect.

**16** A truncation or translation error occurred in accessing dialog variables.

**20** Severe error; unable to continue.

## Example

In this example the LMDLIST service LIST option generates a list of all data set names. The variable ID contains a dslist ID generated by the LMDINIT service. The LMDLIST service places the first name in the variable DSNAME.

### Command Invocation

```
SET &DSNAME =
ISPEXEC   LMDLIST LISTID(&ID) STATS(YES) DATASET(DSNAME) OPTION(LIST)
```

### Call Invocation

```
DSNAME = ' ';
CALL ISPLINK  ('LMDLIST ', ID,'LIST    ',DSNAME,'YES     ');
OR
```

Set the program variable BUFFER to contain:

```
DSNAME = ' ';
BUFFER = 'LMDLIST LISTID(&ID) OPTION(LIST) DATASET(DSNAME)
          STATS(YES)';
```

Set the program variable BUFLEN to the length of the variable BUFFER. Issue the following command:

```
CALL ISPEXEC (BUFLEN, BUFFER);
```

# LMERASE – Erase a Data Set

The LMERASE service deletes an entire ISPF library or an MVS partitioned data set. All members of the data set are deleted and all statistics are erased. The data set name used must be the cataloged name, not an alias data set name.

## Command Invocation Format

```
ISPEXEC LMERASE {[PROJECT(project)]
                [GROUP(group)]

                [TYPE(type)]       }

                [PURGE(YES|NO)]
```

```
[DATASET(dataset)]

[VOLUME(volume)]

[PASSWORD(password)]
```

## Call Invocation Format

```
CALL ISPLINK('LMERASE ',[project]
                        ,[group]

                        ,[type]

                        ,['YES     '|'NO      ']

                        ,[dataset]

                        ,[volume]

                        ,[password]);
```

*OR*

```
CALL ISPEXEC (buf-len, buffer);
```

You must specify the data set (ISPF library, or MVS partitioned or sequential data set) as a three-level qualified name, or as a 44-character data set name string. If both are specified, ISPF will use the data set name string. If neither is specified, an error message is displayed.

## Parameters

**project** The highest-level qualifier in the specification of an ISPF library or of an MVS data set with a three-level qualified data set name. The maximum length of this parameter is 8 characters.

**group** The second-level qualifier in the specification of an ISPF library or of an MVS data set with a three-level qualified data set name. The maximum length of this parameter is 8 characters.

**type** The third-level qualifier in the specification of an ISPF library or of an MVS data set with a three-level qualified data set name. The maximum length of this parameter is 8 characters.

**YES | NO** If YES is specified, LMERASE deletes the data set regardless of its expiration date. If NO (the default) is specified, LMERASE deletes the data set only if its expiration date has passed.

**dataset** The name of an existing MVS partitioned or sequential data set. A member name or pattern cannot be included if the name is that of a partitioned data set. The maximum length of this parameter is 46 characters, with 2 characters for a beginning and ending single quotation mark, and 44 characters for the data set name. If the single quotation marks are omitted, the users data set prefix from the TSO profile is automatically appended to the front of the data set name.

**volume** The serial number of the DASD volume on which the data set resides. This parameter is associated with the data set parameter above, but is required only if the data set is not cataloged. If the

volume parameter is specified but the data set parameter is not, the volume is ignored. The maximum length of this parameter is 6 characters.

**password** The MVS password of the data set. This parameter is required only if the data set is password-protected. Do not specify a password for RACF- or PCF-protected data sets. The maximum length of this password is 8 characters.

**buf-len** A fullword fixed binary integer containing the length of the buffer parameter.

**buffer** A buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC invocation for a command invocation.

## Return Codes

The following return codes are possible:

**0** Normal completion.

**8** One of the following:
- Data set is not cataloged or other allocation failure.
- Data set delete failed.
- Data set name is an alias.
- Expiration date not expired and PURGE parameter omitted
- No data set specified as input
- PROJECT specified, but GROUP or TYPE not specified.

**12** Expiration date not expired and PURGE(NO) specified.

**20** Severe error; unable to continue.

## Example

This example invokes LMERASE to delete a data set with a three-level qualified data set name that has DEPT877 as its highest-level qualifier, PRIVATE as its second-level qualifier, and CLIST as its third-level qualifier.

### Command Invocation

```
ISPEXEC LMERASE PROJECT(DEPT877)          +
                GROUP(PRIVATE)            +
                TYPE(CLIST)               +
                PURGE(YES)
```

### Call Invocation

```
CALL ISPLINK('LMERASE ','DEPT877 ',
                        'PRIVATE ',
                        'CLIST ',
                        'YES     ');
```

*OR*

Set the program variable BUFFER to contain:

```
BUFFER = 'LMERASE PROJECT(DEPT877)
                  GROUP(PRIVATE)
                  TYPE(CLIST)
                  PURGE(YES)';
```

Set the program variable BUFLEN to the length of the variable BUFFER. Issue the following:

```
CALL ISPEXEC (BUFLEN, BUFFER);
```

# LMFREE – Free Data Set from its Association with Data ID

The LMFREE service removes a data ID that was generated by the LMINIT service. The ISPF library, concatenated ISPF libraries, or data set is no longer associated with the specified data ID. If the data set is still open, LMFREE closes it.

After LMFREE is invoked, that data ID can no longer be used to identify the data set for processing by other ISPF services that require data IDs. If the data ID is not allocated by using the DDNAME parameter in LMINIT, the allocation for the data set is also freed. If the data ID represents a concatenated set of ISPF libraries, the data sets are freed and are no longer concatenated.

For each LMINIT invocation, you should invoke a matching LMFREE service when the data ID is no longer needed. Otherwise, the ISPF library or data set associated with the data ID is not released until ISPF terminates.

If you modify the data sets associated with a data ID, then you must invoke the LMFREE and LMINIT services for the data ID before processing the data sets with another service. Failure to update the directory blocks associated with the data ID may cause I/O errors.

## Command Invocation Format

```
ISPEXEC LMFREE DATAID(data-id)
```

## Call Invocation Format

```
CALL ISPLINK('LMFREE  ',data-id);
OR

CALL ISPEXEC (buf-len, buffer);
```

## Parameters

**data-id** The data ID associated with the data set to be released. The data ID has been generated by the LMINIT service. The maximum length of this parameter is 8 characters.

**buf-len** A fullword fixed binary integer containing the length of the buffer parameter.

**buffer** A buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC invocation for a command invocation.

## Return Codes

The following return codes are possible:

**0** Normal completion.

**8** Free data ID failed; the error condition is described in “System Variables Used to Format Error Messages” on page 14.

**10** No ISPF library or data set is associated with the given data ID; that is, LMINIT has not been completed.

**20** Severe error; unable to continue.

## Example

This example invokes the LMFREE service to release the data set associated with the data ID in variable DDVAR.

### Command Invocation

```
ISPEXEC LMFREE DATAID(&DDVAR)
```

### Call Invocation

```
CALL ISPLINK('LMFREE  ',DDVAR);
OR
```

Set the program variable BUFFER to contain:

```
BUFFER = 'LMFREE DATAID(&DDVAR)';
```

Set the program variable BUFLEN to the length of the variable BUFFER. Issue the following:

```
CALL ISPEXEC (BUFLEN, BUFFER);
```

# LMGET – Read a Logical Record from a Data Set

The LMGET service reads one logical record from the data set associated with the given data ID. Completion of the LMINIT and LMOPEN services for the data set is required before LMGET is invoked.

If the data to be processed is a sequential data set, the first LMGET reads the first logical record. Later invocations read successive logical records; that is, the second invocation reads the second logical record, the third invocation reads the third logical record, and so on.

If the data is an ISPF library or MVS partitioned data set, previous completion of the LMMFIND service is required in addition to completion of LMINIT and LMOPEN. The LMGET service reads from the last member referred to by the LMMFIND service in the data sets being processed. Thus, if LMMFIND is issued referencing member A, LMGET reads from member A. If another LMMFIND is issued referencing member B, LMGET reads from member B, not member A.

When MODE(MULTX) is used, the read operation occurs in segments (rather than in single records), with each segment comprising multiple records. Each record is prefixed by a 2-byte binary integer field containing its length. The maximum size of each segment returned is 32 000 bytes. LMGET returns data to the dataloc-var in the following format:

```
+------+--------------+------+--------------+------+--------------+
|len   | record       |len   | record       |len   | record       |
+------+--------------+------+--------------+------+--------------+
<-----     this length is returned in datalen-var             ---->
```

The data read is always unpacked. If the data set contains packed data, LMGET unpacks the data.

## Command Invocation Format

```
ISPEXEC LMGET DATAID(data-id)
              MODE(MOVE|LOCATE|INVAR|MULTX)

              DATALOC(dataloc-var)
```

```
                DATALEN(datalen-var)

                MAXLEN(max-length)
```

## Call Invocation Format

```
CALL ISPLINK ('LMGET   ',data-id
                        ,'MOVE    '|'LOCATE  '|'INVAR   '|'MULTX   '

                        ,dataloc-var

                        ,datalen-var

                        ,max-length);

OR

CALL ISPEXEC (buf-len, buffer);
```

## Parameters

**data-id**
The data ID associated with the data set to be read. The data ID has been generated by the LMINIT service. The maximum length of this parameter is 8 characters.

**MOVE | LOCATE | INVAR | MULTX**
Whether the data is to be moved, located, or stored into an ISPF dialog variable, and whether the data should be read in single records (default) or in segments containing multiple records (MULTX). A calling program function can specify any mode, with information being passed through the data location variable. A command dialog can use INVAR and MULTX modes, with data being returned to the command in the data location variable.

**dataloc-var**
The name of the data location variable. In MOVE mode, the variable contains a binary virtual storage address at which the data read by LMGET is to be stored. In LOCATE mode, the address of the data read by LMGET is placed in the data location variable. In INVAR and MULTX modes, the data read by LMGET is itself placed in the data location variable. The maximum length of this parameter is 8 characters.

**datalen-var**
The name of the variable into which LMGET stores the actual length of the record read. In MULTX mode ISPF stores the length of data returned in the datalen-var. The maximum length of this parameter is 8 characters.

**max-length**
A fullword binary integer containing the maximum record length to be read in bytes. This parameter must be a nonzero positive integer value. In MOVE mode, the value is the maximum number of bytes of data to be moved. In INVAR mode, the value is the maximum number of bytes of data to be stored in the data location variable. The value is not changed by LMGET in either mode. In MULTX mode, the value is the maximum number of bytes to be stored from each record read, to make up the segment that will be stored in the data location variable. The parameter is ignored in LOCATE mode. If the max-length specification causes a DBCS character string to be divided in the middle, the result may be unpredictable.

**buf-len**
A fullword fixed binary integer containing the length of the buffer parameter.

**buffer**
A buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC invocation for a command invocation.

## Return Codes

The following return codes are possible:

**0** Normal completion.

**8** End-of-data set condition; no message formatted.

**10** No ISPF library or data set associated with the given data ID; that is, LMINIT has not been completed.

**12** One of the following:
- The data set is not open or is not open for input.
- An LMMFIND was not done for a partitioned data set.
- The parameter value is invalid.

**16** A truncation or translation error occurred in accessing dialog variables.

**20** Severe error; unable to continue.

## Example

This example invokes the LMGET service to read a record from the data set associated with the data ID in variable DDVAR, in INVAR mode, with LOCVAR as the data location variable, LENVAR as the actual record length variable, and 80 bytes as the maximum record length.

### Command Invocation

```
ISPEXEC LMGET DATAID(&DDVAR)  MODE(INVAR) DATALOC(LOCVAR)  +
              DATALEN(LENVAR) MAXLEN(80)
```

### Call Invocation

```
MAXLEN=80;
CALL ISPLINK('LMGET   ',DDVAR,'INVAR   ','LOCVAR ','LENVAR ',MAXLEN);
```

MAXLEN is a fullword integer variable.

*OR*

Set the program variable BUFFER to contain:

```
BUFFER = 'LMGET DATAID(&DDVAR)  MODE(INVAR) DATALOC(LOCVAR)
               DATALEN(LENVAR) MAXLEN(80)';
```

Set the program variable BUFLEN to the length of the variable BUFFER. Issue the following command:

```
CALL ISPEXEC (BUFLEN, BUFFER);
```

## Example (MULTX)

This REXX example invokes the LMGET service to process a data set in MULTX mode, returning blocks of data in segments no larger than 32 000 bytes. The data set has a record length of 80 bytes in this example, but for variable data the length will vary and must be calculated as shown:

```
/* REXX */
 ADDRESS ISPEXEC;
 'LMGET DATAID('DDVAR') MODE(MULTX) DATALOC(REC) DATALEN(DLEN) ,
  MAXLEN(80)'
 GETRC = RC
 DO FOREVER
   INDEX = 1
   DO WHILE INDEX < DLEN
     LEN = SUBSTR(REC,INDEX,2)
     LEN = C2D(LEN)
     IF LEN > 0 THEN CALL process_record
     INDEX = INDEX + LEN + 2
   END
   IF GETRC = 0 THEN DO
     'LMGET DATAID('DDVAR')' MODE(MULTX) DATALOC(REC) ,
     DATALEN(LEN) MAXLEN(80)'
     GETRC = RC
   END
   ELSE LEAVE
 END
```

## LMINIT – Generate a Data ID for a Data Set

The LMINIT service allows the dialog to associate a data ID with a specified ISPF library, concatenation of ISPF libraries or MVS partitioned data sets, or an MVS partitioned or sequential data set. The data ID is generated by LMINIT and can be used to identify the data set for processing by other library access services or the BROWSE or EDIT service. If the specified data set exists but has not been allocated, the LMINIT service allocates the data set. If two or more existing ISPF libraries are specified, the LMINIT service concatenates the libraries.

**Note:** The LMINIT service does not support data sets created by a method that does not set the format one DSCB field (DS1DSORG).

The input to the LMINIT service defines the physical and logical characteristics of the data set. This simplifies the invocation of the other library access services by supplying the information needed to invoke the service for a given data set. For instance, the dialog supplies the information required by the input fields on the ISPF View Entry Panel to LMINIT. Later invocations of the BROWSE service with that data set are made much simpler by using the data ID generated by the LMINIT service.

The LMINIT service must be completed before LMOPEN can be used. Otherwise, the data set cannot be opened for processing. If LMINIT is issued with an enqueue (ENQ) of SHRW and LMOPEN is issued with the OUTPUT option, it is essential that an LMCLOSE is issued when the dialog has finished processing the data set, since the DASD volume is reserved until LMCLOSE is invoked.

You can use the LMQUERY service to find out how the LMINIT parameters are set.

For each LMINIT invocation, you should invoke a matching LMFREE service. The LMFREE service removes the data ID generated by LMINIT. You should invoke the LMFREE service when the data ID is no longer needed. Otherwise, the ISPF library or data set associated with the data ID is not released until ISPF terminates.

If you modify the data sets associated with a data ID, then you must invoke the LMFREE and LMINIT services for the data ID before processing the data sets with another service. Failure to update the directory blocks associated with the data ID may cause I/O errors.

For example, if you use a service like LMCOPY or LMMOVE to modify a data ID that was defined by the LMINIT service, and the modified resource is needed for other services, then the data ID that references the modified resource must first be freed with LMFREE, then re-allocated with LMINIT. In more specific terms, say you perform an LMMOVE operation to move data from DATA-ID(A) to DATA-ID(B). Then you immediately use the LMMOVE service to move data from DATA-ID(B) to DATA-ID(C). The second operation (from B to C) might result in an I/O error. To correctly complete this task, make all updates to DATA-ID(B), free DATA-ID(B) with the LMFREE service, then use the LMINIT service for DATA-ID(B) so that the changes made to DATA-ID(B) can be referenced by other services. Any time this initialization is not done on a modified resource and references to that resource are made, an I/O error might occur.

## Command Invocation Format

```
ISPEXEC LMINIT DATAID(data-id-var)
      {PROJECT(project) GROUP1(group1) TYPE(type)
      [GROUP2(group2)] [GROUP3(group3)] [GROUP4(group4)]}

      {DATASET(dsname)               }

      {DDNAME(ddname)                }

      [VOLUME(serial)] [PASSWORD(password)]

      [ENQ(SHR|EXCLU|SHRW|MOD)] [ORG(org-var)]
```

## Call Invocation Format

```
CALL ISPLINK ('LMINIT  ', data-id-var
            {,project,group1 [,group2] [,group3] [,group4] ,type}

            {,dsname                          }

            {,ddname                          }

            ,[serial] ,[password]

            ,['SHR     '|'EXCLU   '|'SHRW    '|'MOD     ']

            ,[org-var]);

OR

CALL ISPEXEC (buf-len,buffer);
```

You must specify the data set (ISPF library, or MVS partitioned or sequential data set) as a ddname, a dsname, or a three-level qualified name. The search sequence LMINIT uses is ddname, then dsname, then the three-level qualified name. If LMINIT finds the name it is looking for, it uses that name. Otherwise, it looks for the next type of name in the sequence. If there is no three-level qualified name, LMINIT issues an error message.

## Parameters

**data-id-var** The name of the variable that will store the data ID to be

associated with the data set. The LMINIT service always generates a unique data ID. The data ID is an input parameter to most of the other library access services, and optionally to the BROWSE and EDIT services, but is an output parameter from the LMINIT service. The data ID length is 8 characters. Therefore, the maximum length of this parameter is 8 characters.

To invoke the service, you must specify the data ID variable name and an ISPF library name (project, group, and type), a dsname, or a ddname. The ISPF library name, dsname, and ddname are described below.

In the LMINIT service, *data-id-var* is the name of the variable that holds the data ID (for example, DATAID(DDVAR)). When you use the data ID keyword with other services, you must pass the value of the variable (for example, DATAID(&DDVAR)). The Library search order is from the lowest (group1) to the highest (group4). The search for a member stops when the first matching member name is located.

**project** The highest-level qualifier in the specification of an ISPF library or MVS three-level qualified data set. This parameter is required if neither the dsname nor the ddname parameter is specified. The maximum length of this parameter is 8 characters.

**group1** The second-level qualifier in the specification of an ISPF library or MVS three-level qualified data set. This parameter is required if neither the dsname nor ddname parameter is specified. The maximum length of this parameter is 8 characters.

**group2** Continues the second-level qualifier above. It is not required, but if present it represents an ISPF library in a concatenation sequence. The maximum length of this parameter is 8 characters.

**group3** Continues the second-level qualifier above. It is not required, but if present it represents an ISPF library in a concatenation sequence. The maximum length of this parameter is 8 characters.

**group4** Continues the second-level qualifier above. It is not required, but if present it represents an ISPF library in a concatenation sequence. The maximum length of this parameter is 8 characters.

**type** The third-level qualifier in the specification of an ISPF library or MVS three-level qualified data set. This parameter is required if neither the dsname nor the ddname parameter is specified. The maximum length of this parameter is 8 characters.

**dsname** The name of an existing MVS partitioned or sequential data set. A member name or pattern cannot be included in the dsname of a partitioned data set. The maximum length of this parameter is:

- For fully qualified data sets, 46 characters, with 2 characters for a beginning and ending single quotation mark, and 44 characters for the data set name.
- If the single quotation marks are omitted, the user's data set prefix from the TSO profile is automatically appended to the front of the data set name. The length of the data set name specified plus the length of the TSO prefix and the separator "." must not exceed 44 characters.

**ddname** The data set definition name of a data set that is already allocated

to the TSO user before invocation of the LMINIT service. This can be done by using the TSO ALLOCATE command or MVS job control language (JCL). The data set must be either partitioned or sequential.

If the ddname is allocated to one or more partitioned data sets, member names cannot be included. LMINIT allows up to 16 concatenated data sets.

**Note:** If the ddname is allocated to a multivolume data set, LMINIT is not supported. Do not try to LMINIT a multivolume data set by ddname.

Sequential data sets must be allocated as either OLD, SHR, NEW, or MOD. If the ddname is allocated as NEW, the record format, data set organization, record length, and block size must be specified when the ddname is allocated. For a partitioned data set, the number of directory blocks must also be specified when the ddname is allocated. The maximum length of this parameter is 8 characters.

**serial** The serial number of the DASD volume on which the data set resides. This parameter is associated with the dsname parameter above, but is required only if the data set is not cataloged. The maximum length of this parameter is 6 characters. Volume serial is associated with the dsname parameter and will be ignored when the dsname is not entered.

**password** The MVS password of the data set. This parameter is required only if the data is password-protected. If the password is invalid, it is detected by the LMOPEN service (see "LMOPEN – Open a Data Set" on page 160). Do not specify a password for RACF- or PCF-protected data sets. The maximum length of this parameter is 8 characters.

**SHR | EXCLU | SHRW | MOD**

The requirements for enqueuing (ENQ) the data within ISPF so that the dialog can use it in the desired manner. This parameter is ignored if the ddname parameter is specified.

SHR shows that the existing data can be shared; for example, it can be used by two or more users who want only to read the data. You can specify this option when using the INPUT option of the LMOPEN service. SHR is the default.

EXCLU shows that exclusive use of the data is required; for example, when you want to change the data no one else can have access to it. You can specify this option for either the INPUT or OUTPUT option of the LMOPEN service.

SHRW permits a *shared write* for the data. This option is used by the PDF Edit. It is used only for a partitioned data set. In this way, more than one user can read from the data, but members can be rewritten when necessary through an enqueue or dequeue used by Edit. Edit can now have the data ID open for INPUT and OUTPUT at the same time. A data set that is allocated with an enqueue of SHRW can be opened for either INPUT or OUTPUT using the LMOPEN service.

MOD shows that more records are to be added to the end of a sequential data set. MOD is used with the OUTPUT option of the LMOPEN service.

**org-var** The name of the variable into which the organization of the data is stored. The variable contains "PO" if the data set is partitioned or "PS" if it is physical sequential. If you specify a concatenated set of ISPF libraries, the organization of the first group of the concatenated libraries is returned. The maximum length of this parameter is 8 characters.

**buf-len** A fullword fixed binary integer containing the length of the buffer parameter.

**buffer** A buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC invocation for a command invocation.

## Return Codes

The following return codes are possible:

**0** Normal completion.

**8** Data ID not created; the error condition is described in "System Variables Used to Format Error Messages" on page 14.

**12** The parameter value is invalid.

**16** Truncation or translation error in accessing dialog variables.

**20** Severe error; unable to continue.

Refer to "System Variables Used to Format Error Messages" on page 14 for more information about dialog variables.

## Examples

This example invokes the LMINIT service to associate a data ID with data concatenated from the following ISPF libraries:

ISPF.TESTLIB1.PLIOPT
ISPF.TESTLIB2.PLIOPT
ISPF.TESTLIB3.PLIOPT
ISPF.TESTLIB4.PLIOPT

Store the generated data ID in variable DDVAR.

### Command Invocation

```
ISPEXEC LMINIT DATAID(DDVAR) PROJECT(ISPF)  +
               GROUP1(TESTLIB1)  +
               GROUP2(TESTLIB2) GROUP3(TESTLIB3)   +
               GROUP4(TESTLIB4) TYPE(PLIOPT)
```

### Call Invocation

```
DCL DDVAR CHAR (8);
CALL ISPLINK('VDEFINE ','DDVAR ',DDVAR,'CHAR    ',
              LENGTH(DDVAR));
CALL ISPLINK('LMINIT  ','DDVAR ','ISPF ',
             'TESTLIB1 ','TESTLIB2 ',
             'TESTLIB3 ','TESTLIB4 ','PLIOPT ');
OR
```

Set the program variable BUFFER to contain:

```
BUFFER = 'LMINIT DATAID(DDVAR) PROJECT(ISPF) GROUP1(TESTLIB1)
               GROUP2(TESTLIB2) GROUP3(TESTLIB3)
               GROUP4(TESTLIB4) TYPE(PLIOPT)';
```

Set the program variable BUFLEN to the length of the variable BUFFER. Issue the following command:

```
CALL ISPEXEC (BUFLEN, BUFFER);
```

The following example invokes the LMINIT service for a two-level dsname called SMITH.CLIST, using dsname.

### Command Invocation

```
ISPEXEC LMINIT DATAID(DDVAR)              +
               DATASET('SMITH.CLIST')     +
               ENQ(SHR)
```

### Call Invocation

```
CALL ISPLINK('LMINIT  ','DDVAR ',
             ' ',' ',' ',' ',' ',' ',' ',
             'SMITH.CLIST',' ',
             ' ',' ','SHR    ');
```

The following example invokes the LMINIT service for a new data set, using ddname.

### Command Invocation

```
ATTRIB MYLIST BLKSIZE(800)     +
       LRECL(80) RECFM(F B)    +
       DSORG(PS)
ALLOC DDNAME(MYDD) NEW         +
       SPACE(1,1) TRACKS KEEP  +
       USING(MYLIST)
ISPEXEC LMINIT DATAID(DDVAR) DDNAME(MYDD)
```

### Call Invocation

For this invocation, assume DDNAME(MYDD) has been allocated to the user using JCL.

```
CALL ISPLINK ('LMINIT  ','DDVAR ',
              ' ',' ',' ',' ',' ',' ',' ',
              ' ','MYDD ');
```

## LMMADD – Add a Member to a Data Set

The LMMADD service adds a member to the specified ISPF library or MVS partitioned data set. LMMADD then updates the data set directory with information about the member to be added. If the member already exists, the member name entry is not added. The LMINIT with either ENQ(SHRW) or ENQ(EXCLU), LMOPEN with OPTION(OUTPUT), and LMPUT services must be completed before LMMADD is used.

### Command Invocation Format

```
ISPEXEC LMMADD DATAID(data-id)
               MEMBER(member-name)

               [STATS(YES|NO)]

               [NOENQ]
```

## Call Invocation Format

```
CALL ISPLINK ('LMMADD  ',data-id
                        ,member-name

                        ,['YES     '|'NO      '])

                        ,['NOENQ']);

OR

CALL ISPEXEC (buf-len, buffer);
```

## Parameters

**data-id** The data ID associated with the data set to which a member is being added. The data ID has been generated by the LMINIT service. The maximum length of this parameter is 8 characters.

**member-name** The member name being added to the directory. The maximum length of this parameter is 8 characters.

**YES** | **NO** Whether the user data area in the directory should be updated so that the statistics of the member are stored in the format used by the PDF Edit.

If you specify YES, and the data set is partitioned and does not have unformatted records (RECFM=U), the directory is updated with the member statistics. At least a valid creation date (ZLCDATE) and the date of last change (ZLMDATE), as described below, must be provided in the member statistics.

If you specify NO (the default value), statistics are not updated.

The following dialog variables are used to pass statistical information from the dialog invoking the LMMADD service:

**ZLVERS**
Version number; a number from 1 to 99. If no value exists for this variable, ISPF sets the value to blanks.

**ZLMOD**
Modification level; a number from 0 to 99.

**ZLCDATE**
Creation date; a character value shown in your national format. If no value exists for this variable, ISPF sets the value to blanks.

**ZLMDATE**
Last change date; a character value shown in your national format. If no value exists for this variable, ISPF sets the value to blanks.

**ZLMTIME**
Last change time; a character value in the format hh:mm. ZLMTIME may also be specified as an 8-character field in the format hh:mm:ss. If the 6th character is not a colon, or if the 7th and 8th characters (ss) are not in the range '00' to '59', only the hour:minute specifications are used. The seconds value is set to the current time.

**ZLMSEC**
*Seconds* value of the last change time. This is a 2-character field.

**Note:** If the ZLMTIME variable does not contain a *seconds* value and ZLMSEC is not set, the *seconds* value is set to `00`. If both ZLMTIME and ZLMSEC specify a *seconds* value, the value in ZLMSEC is used.

**ZLCNORC**
Current number of records; a number from 0 to 65 535. If no value exists for this variable, ISPF sets the value to blanks.

**ZLINORC**
Beginning number of records; a number from 0 to 65 535.

**ZLMNORC**
Number of changed records; a number from 0 to 65 535.

**ZLUSER**
User ID of the last user to change the given member; the user ID has a maximum length of 7 characters.

**ZLC4DATE**
Creation date in 4 character year format; a character variable shown in your national format. If no value exists for this variable, PDF sets the value to blanks.

**ZLM4DATE**
Last modified date in 4 character year format; a character variable shown in your national format. If no value exists for this variable, PDF sets the value to blanks.

The preceding variables are stored in the function pool and therefore become immediately available to command invocations. You cannot use the VGET service to retrieve these variables, since the VGET service accesses the shared and profile pools. Likewise, you cannot use the VPUT service to change these variables.

**NOENQ** An optional parameter that specifies that PDF should not issue its standard ENQ during the processing of this service. This standard ENQ consists of a major name of SPFEDIT and a minor name of the data set name and member. PDF by default will issue the ENQ unless NOENQ is specified.

**buf-len** A fullword fixed binary integer containing the length of the buffer parameter.

**buffer** A buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC invocation for a command invocation.

## Return Codes

The following return codes are possible:

**0** Normal completion.

**4** The directory already contains the specified name.

**10** No ISPF library or MVS data set is associated with the given data ID; that is, LMINIT has not been completed.

**12** One of the following:
- The data set is not open or is not open for output.
- The parameter value is invalid.
- The data set organization is invalid.
- The values for some member statistics are invalid.

**14** No record has been written for the member to be added.

**16** A truncation or translation error occurred in accessing dialog variables.

**20** Severe error; unable to continue.

## Example

This example invokes the LMMADD service to add member MYPROG to the data set associated with the data ID in variable DDVAR.

### Command Invocation

```
ISPEXEC LMMADD DATAID(&DDVAR) MEMBER(MYPROG)
```

### Call Invocation

```
CALL ISPLINK('LMMADD  ',DDVAR,'MYPROG ');
OR
```

Set the program variable BUFFER to contain:

```
BUFFER = 'LMMADD DATAID(&DDVAR) MEMBER(MYPROG)';
```

Set the program variable BUFLEN to the length of the variable BUFFER. Issue the following:

```
CALL ISPEXEC (BUFLEN, BUFFER);
```

# LMMDEL – Delete Members from a Data Set

The LMMDEL service removes members matching the specified pattern from an ISPF library or MVS partitioned data set. All directory information associated with the member is deleted. The LMINIT and LMOPEN services must be completed before you use the LMMDEL service. The LMINIT must be done with either the ENQ(SHRW) or ENQ(EXCLU) option, and the LMOPEN must have been done for OUTPUT. An LMINIT with ENQ(EXCLU) is required when MEMBER(*) is specified.

## Command Invocation Format

```
ISPEXEC LMMDEL DATAID(data-id)
               MEMBER(member-name)

               [NOENQ]
```

## Call Invocation Format

```
CALL ISPLINK('LMMDEL  ',data-id
                       ,member-name)

                       ,['NOENQ']);

OR

CALL ISPEXEC (buf-len, buffer);
```

## Parameters

**data-id** The data ID associated with the data set from which a member is to be deleted. The data ID has been generated by the LMINIT service. The maximum length of this parameter is 8 characters.

**member-name** The member name or pattern of the members to be deleted. An asterisk (*) indicates that all members are to be deleted. The maximum length of this parameter is 8 characters.

**NOENQ** An optional parameter that specifies that PDF should not issue its standard ENQ during the processing of this service. This standard ENQ consists of a major name of SPFEDIT and a minor name of the data set name and member. PDF by default will issue the ENQ unless NOENQ is specified.

**buf-len** A fullword fixed binary integer containing the length of the buffer parameter.

**buffer** A buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC invocation for a command invocation.

## Return Codes

The following return codes are possible:

**0** Normal completion.

**8** The member was not found.

**10** No data set is associated with the given data ID; that is, LMINIT has not been completed.

**12** One of the following:
- The data set is not open or is not open for output.
- The parameter value is invalid.
- The data set organization is invalid.

**20** Severe error; unable to continue.

## Example

This example invokes the LMMDEL service to delete member MYPROG from the data set associated with the data ID in variable DDVAR.

### Command Invocation

```
ISPEXEC LMMDEL DATAID(&DDVAR) MEMBER(MYPROG)
```

### Call Invocation

```
CALL ISPLINK('LMMDEL  ',DDVAR,'MYPROG ');
OR
```

Set the program variable BUFFER to contain:

```
BUFFER = 'LMMDEL DATAID(&DDVAR) MEMBER(MYPROG)';
```

Set the program variable BUFLEN to the length of the variable BUFFER. Issue the following:

```
CALL ISPEXEC (BUFLEN, BUFFER);
```

# LMMDISP – Member List Service

LMMDISP provides a flexible and efficient way of performing many of the tedious tasks associated with processing member lists. A member list is a list of members from a single ISPF library, or concatenation of ISPF libraries or MVS partitioned data sets associated with a data ID.

The dialog invoking LMMDISP must first issue a successful call to both LMINIT and LMOPEN.

The LMMDISP service performs six member list functions for a dialog according to the value specified in the OPTION parameter. The six values that can be specified are:

**Display**
: This option creates and displays a member list for the specified data ID. A user can select members for processing from this member list by entering a valid line command next to the member name or by using the SELECT primary command. A member that does not exist on the member list can also be selected by using the SELECT primary command. The first member selected from this display is returned in ISPF dialog variables.

  A nonexistent member can only be selected if LMMDISP was invoked with the ALLOWNEW parameter.

**Get** This option is used to return the second, and remaining selected members from the most recent member list display. The GET option must be invoked for each selected member that is to be returned. The GET option can only return one selected member at a time.

**Put** This option saves information in the **Line Command** field, and the **User Data** field of the member list.

**Add** This option adds a member to a member list.

**Delete** This option deletes a member from a member list.

**Free** This option frees the storage associated with a member list.

The description of each option, including format, parameters, return codes, and examples, follows a discussion on dialog variables.

**Note:** Member lists generated by LMMLIST cannot be displayed by LMMDISP and member lists generated by LMMDISP cannot be used with LMMLIST. Member lists should be freed when switching between LMMLIST and LMMDISP with the same data ID by using OPTION(FREE).

## Dialog Variables

The following table contains variables that LMMDISP saves in the function pool before returning a selected member to the dialog that invoked it. The "Returned" column indicates when a given variable is returned. For example, STATS(YES) indicates that the variable is returned only if the dialog invokes LMMDISP with STATS(YES).

| Variable Name | Returned | Variable Description |
|---|---|---|
| ZLAC | STATS(YES) | 2-character field containing the authorization code of the member. |

| Variable Name | Returned | Variable Description |
|---|---|---|
| ZLALIAS | STATS(YES) | 8-character field containing the name of the real member that this member is an alias of. |
| ZLAMODE | STATS(YES) | 3-character field containing the AMODE of the member. |
| ZLATTR | STATS(YES) | 20-character field containing the load module attributes. |
| ZLC4DATE | STATS(YES) | Member creation date, 4-digit year. |
| ZLCDATE | STATS(YES) | Member creation date. |
| ZLCNORC | STATS(YES) | Current number of records. |
| ZLINORC | STATS(YES) | Initial number of records. |
| ZLLCMD | always | Line command used to select the member. |
| ZLLIB | STATS(YES) | Number from 1 to 4 representing position of library in concatenation sequence. |
| ZLM4DATE | STATS(YES) | Date member was last modified, 4-digit year. |
| ZLMDATE | STATS(YES) | Date member was last modified. |
| ZLMEMBER | always | Member name of selected member. |
| ZLMNORC | STATS(YES) | Number of modified records. |
| ZLMOD | STATS(YES) | PDF modification number. |
| ZLMSEC | STATS(YES) | Seconds value of the last change time. |
| ZLMTIME | STATS(YES) | Time member was last modified. |
| ZLMTOP | always | Member that appeared at the top of the screen when the display ended. |
| ZLPDSUDA [1] | STATS(YES) | Value of PDS directory user data area. |
| ZLRMODE | STATS(YES) | 3-character field containing the RMODE of the member. |
| ZLSIZE | STATS(YES) | 8-character field containing the load module size in hex. |
| ZLSSI | STATS(YES) | 8-character field containing the SSI information for a load module. |
| ZLTTR | STATS(YES) | 6-character field containing the TTR of the member. |
| ZLUDATA | always | User data area on member list. |
| ZLUSER | STATS(YES) | System USERID of user to last modify member. |
| ZLVERS | STATS(YES) | PDF version number. |
| ZSCLM | STATS(YES) | Indicates whether the system was last modified by SCLM or PDF. |

## DISPLAY Option

The DISPLAY option creates a member list and displays it. You can specify a customized panel, place the cursor, and have member list line commands validated.

LMMDISP with OPTION(DISPLAY) must be the first invocation of LMMDISP with a data ID once you have invoked LMINIT and LMOPEN with that data ID. This creates a member list for the data ID and displays it. Subsequent calls with the DISPLAY option simply display the member list again. Modification of parameters

1. ZLPDSUDA is put in the ISPF function pool only if STATS(YES) was specified and the selected member being returned had member statistics that did not conform to ISPF standards. For example, a load module member of a partitioned data set usually has load module statistics, and not PDF statistics.

MEMBER, COMMANDS, and FIELD are ignored after a member list has been created until it is freed by an LMMDISP invocation with OPTION(FREE).

When the member list panel is displayed, you can select members for processing by entering valid line commands next to the member names or by using the SELECT primary command.

If a member or members were selected, LMMDISP returns the first or only selected member in ISPF dialog variables. To retrieve the remaining selections, LMMDISP with OPTION(GET) must be invoked for each selected member.

## Command Invocation Format

```
ISPEXEC LMMDISP DATAID(data-id)
        [OPTION(DISPLAY)]

        [MEMBER(pattern)]

        [STATS(YES|NO)]

        [PANEL(panel-name)]

        [CURSOR(ZCMD|ZLLCMD|ZLUDATA)]

        [TOP(top-row)]

        [COMMANDS(S|ANY)]

        [FIELD(1|9)]
        [ALLOWNEW]
```

## Call Invocation Format

```
CALL ISPLINK('LMMDISP ', data-id
         ,['DISPLAY ']

         ,[pattern]

         ,['YES     '|'NO      ']

         ,[panel-name]

         ,['ZCMD    '|'ZLLCMD  '|'ZLUDATA ']

         ,[top-row]

         ,' '

         ,' '

         ,['S       '|'ANY     ']

         ,[1|9]
         ,['ALLOWNEW'];

OR

CALL ISPEXEC (buf-len, buffer);
```

## Parameters

**data-id** The variable in which the data ID that uniquely identifies the data set is stored.

**DISPLAY** Indicates to LMMDISP that it is to create a member list if one does not exist and display it.

**pattern** The character string that is used to specify which members are to be displayed. Refer to the *ISPF User's Guide Volume I* for a more complete description of patterns and pattern matching.

**YES | NO** Indicates if LMMDISP is to return member statistics via dialog variables. Refer to "Dialog Variables" on page 129 for a list of the dialog variables.

**panel-name** The name of the panel on which the member list is to be displayed. Refer to *ISPF Planning and Customizing* for requirements for customized panels. If this option is omitted, the panel is ISRML000.

**ZCMD | ZLLCMD | ZLUDATA**
The name of the field on which the cursor is placed when the member list is displayed. If ZLLCMD or ZLUDATA is specified, the cursor is placed on that field of the first member to appear on the display.

**top-row** The name that designates which member is to appear first on the display. If the member cannot be found and the list is sorted by name, the member immediately preceding the requested one in the member list is scrolled to the top. If the list is not sorted by name and the member is not found, the list is scrolled to the top.

**S | ANY** S indicates that LMMDISP is to allow only S as a valid line command for member selection. ANY indicates to LMMDISP that any character or character string is a valid line command.

**1 | 9** Indicates to LMMDISP the length of the **Line Command** field on the member list display.

If 9 is specified and the data sets associated with the specified data ID have formatted records, the **Created** field is left out of the member list display. If the data sets do not have formatted records (RECFM=U), the **Alias** field is left out of the member list display.

**buf-len** A fullword fixed binary integer containing the length of the buffer parameter.

**buffer** A buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC invocation for a command procedure.

**ALLOWNEW** Indicates that nonexisting members can also be selected. Omitting this parameter causes only existing members to be selected.

## Return Codes

The following return codes are possible:

**0** One or more members were selected and/or a primary command not recognized by LMMDISP was entered.

**4** The requested data sets were empty, or no members matched the specified pattern.

**8** END or RETURN was entered.

**10** No data set is associated with the given data ID; LMINIT has not been completed.

**12** Indicates one of the following conditions:
- Data set not open.
- Data set not partitioned.
- Invalid parameter value.
- Invalid data set organization.
- Invalid invocation syntax.

**16** A truncation or translation error occurred in accessing dialog variables.

**20** Severe error; unable to continue.

### Example

This example invokes the DISPLAY option of the LMMDISP service to display the data associated with the data ID in variable DDVAR.

```
ISPEXEC LMMDISP DATAID(&DDVAR)    +
        OPTION(DISPLAY)           +
        MEMBER(ISR*)              +
        STATS(YES)                +
        CURSOR(ZCMD)              +
        COMMANDS(S)               +
        FIELD(1)
```

**Call Invocation:**

```
CALL ISPLINK('LMMDISP ',
             DDVAR,
             'DISPLAY '
             'ISR* ',
             'YES     ',
             ' ',
             'ZCMD    ',
             ' ',
             ' ',
             ' ',
             'S       ',
             1);
```

*OR*

Set the program variable BUFFER to contain:

```
BUFFER = 'LMMDISP DATAID(&DDVAR)
          OPTION(DISPLAY)
          MEMBER(ISR*)
          STATS(YES)
          CURSOR(ZCMD)
          COMMANDS(S)
          FIELD(1)';
```

Set the program variable BUFLEN to the length of the variable BUFFER. Issue the following command:

```
CALL ISPEXEC (BUFLEN,BUFFER);
```

## GET Option

The GET option is used to return information about the second, and all other selected members from the member list that was created during the last member list display (LMMDISP with OPTION(DISPLAY)). One selected member is returned in the ISPF dialog variables for each invocation of LMMDISP with the GET option.

### Command Invocation Format

```
ISPEXEC LMMDISP DATAID(data-id)
        OPTION(GET)
        [STATS(YES|NO)]
```

## Call Invocation Format

```
CALL ISPLINK('LMMDISP ', data-id
        ,'GET     '

        ,' '

        ,['YES     '|'NO      ']);
```

*OR*

```
CALL ISPEXEC (buf-len, buffer);
```

## Parameters

**data-id** Variable in which the data ID that uniquely identifies the data sets is stored.

**GET** Indicates to LMMDISP that it is to return the next member and, optionally, the member statistics.

**YES**|**NO** Indicates whether LMMDISP is to return member statistics through dialog variables to the dialog. See "Dialog Variables" on page 129 for a list of dialog variables.

**buf-len** A fullword fixed binary integer containing the length of the buffer parameter.

**buffer** A buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC invocation for a command procedure.

## Return Codes

**0** Successful completion.

**8** No more selected members.

**10** No data set is associated with the given data ID; LMINIT has not been completed.

**12** Indicates one of the following conditions:
- Data set not open.
- Data set not partitioned.
- Invalid parameter value.
- Invalid data set organization.
- Invalid invocation syntax.
- Member list has not been created.

**16** A truncation or translation error occurred in accessing dialog variables.

**20** Severe error; unable to continue.

## Example

This example invokes the GET option of the LMMDISP service to get the next selected member of the member list of the data set associated with the data ID in variable DDVAR.

**Command Invocation:**

```
ISPEXEC LMMDISP DATAID(&DDVAR)    +
        OPTION(GET)    +
        STATS(YES)
```

**Call Invocation:**

```
CALL ISPLINK ('LMMDISP ', DDVAR
        ,'GET     '
        ,' '
        ,'YES     ');
OR
```

Set the program variable BUFFER to contain:

```
BUFFER = 'LMMDISP DATAID(&DDVAR)
          OPTION(GET)
          STATS(YES)';
```

Set the program variable BUFLEN to the length of the variable BUFFER. Issue the following command:

```
CALL ISPEXEC (BUFLEN,BUFFER);
```

# PUT Option

The PUT option saves information in the **Line Command** field and **User Data** field of a member in the member list. The **User Data** field is the field located between the member name and the member statistics on the member list display panel.

## Command Invocation Format

```
ISPEXEC LMMDISP DATAID(data-id)
        OPTION(PUT)

        MEMBER(member-name)

        [ZLLCMD(lcmd-value)]

        [ZLUDATA(udata-value)]
```

## Call Invocation Format

```
CALL ISPLINK('LMMDISP ', data-id
        ,'PUT     '

        ,member-name

        ,' '

        ,' '

        ,' '

        ,' '

        ,[lcmd-value]

        ,[udata-value]);

OR

CALL ISPEXEC (buf-len,buffer);
```

## Parameters

**data-id** Variable in which the data ID that uniquely identifies the data sets is stored.

**PUT** Indicates to LMMDISP that it is to save member list information for the member specified by member-name parameter.

**member-name** The name of the member for which this information is being saved.

**lcmd-value** Value to be stored in the **Line Command** field of member specified by *member-name*. If it is longer than the line command area, it will be truncated, though it must not exceed 9 characters. The length of this variable is the same as the value of the specification of keyword FIELD on the first member list display.

**udata-value** Value to be stored in the User Data field of member specified by *member-name*. The value must not exceed 8 characters, must not contain embedded blanks, and will be converted to uppercase.

**buf-len** A fullword fixed binary integer containing the length of the buffer parameter.

**buffer** A buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC invocation for a command procedure.

## Return Codes

**0** Successful completion.

**8** A specified member does not exist in the member list.

**10** No data set is associated with the given data ID; LMINIT has not been completed.

**12** Indicates one of the following conditions:
- Data sets not open.
- Data sets not partitioned.
- Invalid parameter value.
- Invalid data set organization.
- Invalid invocation syntax.
- Member list has not been created.

**16** A truncation or translation error occurred in accessing dialog variables.

**20** Severe error; unable to continue.

## Example

This example invokes the PUT option of the LMMDISP service to save information in the member list associated with the data ID in variable DDVAR.

**Command Invocation:**

```
ISPEXEC LMMDISP DATAID(&DDVAR)   +
            OPTION(PUT)    +
            MEMBER(ISRFIRST)    +
            ZLUDATA(*RENAMED)
```

**Call Invocation:**

```
CALL ISPLINK('LMMDISP ', DDVAR,
            'PUT     ',
            'ISRFIRST',
            ' ',
            ' ',
            ' ',
            ' ',
            ' ',
            ' ',
            '*RENAMED');
OR
```

Set the program variable BUFFER to contain:

```
BUFFER = 'LMMDISP DATAID(&DDVAR)
          OPTION(PUT)
          MEMBER(ISRFIRST)
          ZLUDATA(*RENAMED)';
```

Set the program variable BUFLEN to the length of the variable BUFFER. Issue the following command:

```
CALL ISPEXEC (BUFLEN,BUFFER);
```

# ADD Option

The ADD option adds a member to an existing member list. The member must not exist in the member list and does not have to exist in the data set concatenation.

## Command Invocation Format

```
ISPEXEC LMMDISP DATAID(data-id)
        OPTION(ADD)

        MEMBER(member-name)

        [ZLLCMD(lcmd-value)]

        [ZLUDATA(udata-value)]
```

## Call Invocation Format

```
CALL ISPLINK('LMMDISP ', data-id
        ,'ADD     '

        ,member-name

        ,' '

        ,' '

        ,' '

        ,' '

        ,[lcmd-value]

        ,[udata-value]);
```

*OR*

```
CALL ISPEXEC (buf-len,buffer);
```

## Parameters

**data-id** Variable in which the data ID that uniquely identifies the data sets is stored.

**ADD** Indicates to LMMDISP that it is to add a member to the member list.

**member-name** Name of member to add to the member list.

**lcmd-value** The value to be stored in the **Line Command** field of the member specified by *member-name*. If it is longer than the line command field, it will be truncated, though it must not exceed 9 characters.

The length of this variable is the same as the value of the specification of keyword FIELD on the first member list display.

**udata-value** The value to be stored in the User Data field of the member specified by *member-name*. The value must not exceed 8 characters, must not contain embedded blanks, and will be converted to uppercase.

**buf-len** A fullword fixed binary integer containing the length of the buffer parameter.

**buffer** A buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC invocation for a command procedure.

## Return Codes

**0** Successful completion.

**8** The member already exists in the member list.

**10** No data set is associated with the given data ID; LMINIT has not been completed.

**12** Indicates one of the following conditions:
- Data sets not open.
- Data sets not partitioned.
- Invalid parameter value.
- Invalid data set organization.
- Invalid invocation syntax.
- Member list has not been created.

**16** A truncation or translation error occurred in accessing dialog variables.

**20** Severe error; unable to continue.

## Example

This example invokes the ADD option of the LMMDISP service to add a member to the member list associated with the data ID in variable DDVAR.

**Command Invocation:**

```
ISPEXEC LMMDISP DATAID(&DDVAR)    +
        OPTION(ADD)    +
        MEMBER(NEWMEMB)    +
        ZLUDATA(*NEWMEMB)
```

**Call Invocation:**

```
CALL ISPLINK('LMMDISP ', DDVAR,
             'ADD     ',
             'NEWMEMB ',
             ' ',
             ' ',
             ' ',
             ' ',
             ' ',
             '*NEWMEMB');
```

*OR*

Set the program variable BUFFER to contain:

```
BUFFER = 'LMMDISP DATAID(&DDVAR)
          OPTION(ADD)
          MEMBER(NEWMEMB)
          ZLUDATA(*NEWMEMB)';
```

Set the program variable BUFLEN to the length of the variable BUFFER. Issue the following:

```
CALL ISPEXEC (BUFLEN,BUFFER);
```

# DELETE Option

The DELETE option deletes a member from an existing member list. The member must exist in the member list. The member is not deleted from the partitioned data set in which it resides, only from the member list itself.

## Command Invocation Format

```
ISPEXEC LMMDISP DATAID(data-id)
        OPTION(DELETE)

        MEMBER(member-name)
```

## Call Invocation Format

```
CALL ISPLINK('LMMDISP ', data-id
        ,'DELETE  '

        ,member-name
OR

CALL ISPEXEC (buf-len,buffer);
```

## Parameters

**data-id** Variable in which the data ID that uniquely identifies the data sets is stored.

**DELETE** Indicates to LMMDISP that it is to delete a member from the member list.

**member-name** Name of member to delete from the member list.

**buf-len** A fullword fixed binary integer containing the length of the buffer parameter.

**buffer** A buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC invocation for a command procedure.

## Return Codes

**0** Successful completion.

**8** A specified member does not exist in the member list.

**10** No data set is associated with the given data ID; LMINIT has not been completed.

**12** Indicates one of the following conditions:
- Data sets not open.
- Data sets not partitioned.
- Invalid parameter value.
- Invalid data set organization.
- Invalid invocation syntax.
- Member list has not been created.

**16** A truncation or translation error occurred in accessing dialog variables.

**20** Severe error; unable to continue.

### Example

This example invokes the DELETE option of the LMMDISP service to delete a member from the member list associated with the data ID in variable DDVAR.

**Command Invocation:**

```
ISPEXEC LMMDISP DATAID(&DDVAR)    +
        OPTION(DELETE)    +
        MEMBER(ISRFIRST)
```

**Call Invocation:**

```
CALL ISPLINK('LMMDISP ', DDVAR,
             'DELETE  ',
             'ISRFIRST');
```

*OR*

Set the program variable BUFFER to contain:

```
BUFFER = 'LMMDISP DATAID(&DDVAR)
          OPTION(DELETE)
          MEMBER(ISRFIRST)';
```

Set the program variable BUFLEN to the length of the variable BUFFER. Issue the following:

```
CALL ISPEXEC (BUFLEN,BUFFER);
```

## FREE Option

The FREE option frees the storage used by the member list.

### Command Invocation Format

```
ISPEXEC LMMDISP DATAID(data-id)
        OPTION(FREE)
```

### Call Invocation Format

```
CALL ISPLINK('LMMDISP ', data-id
            ,'FREE    ');
```

*OR*

```
CALL ISPEXEC (buf-len,buffer);
```

### Parameters

**data-id**
Variable in which the data ID that uniquely identifies the data sets is stored.

**FREE**
Indicates to LMMDISP that it is to free the member list and associated storage.

**buf-len**
A fullword fixed binary integer containing the length of the buffer parameter.

**buffer**
A buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC invocation for a command procedure.

### Return Codes

**0** Successful completion.

**8** No member list is associated with the given data ID.

**10** No data set is associated with the given data ID; LMINIT has not been completed.

**12** Indicates one of the following conditions:
- Data sets not open.
- Data sets not partitioned.
- Invalid parameter value.
- Invalid data set organization.
- Invalid invocation syntax.

**16** A truncation or translation error occurred in accessing dialog variables.

**20** Severe error; unable to continue.

### Example

This example invokes the FREE option of the LMMDISP service to free the storage space used by the associated data ID in the variable DDVAR.

**Command Invocation:**

```
ISPEXEC LMMDISP DATAID(&DDVAR)    +
        OPTION(FREE)
```

**Call Invocation:**

```
CALL ISPLINK('LMMDISP ', DDVAR
            ,'FREE    ');
OR
```

Set the program variable BUFFER to contain:

```
BUFFER = 'LMMDISP DATAID(&DDVAR)
          OPTION(FREE)';
```

Set the program variable BUFLEN to the length of the variable BUFFER. Issue the following command:

```
CALL ISPEXEC (BUFLEN,BUFFER);
```

## LMMFIND – Find a Library Member

The LMMFIND service finds a specified member of an ISPF library or partitioned data set associated with a given data ID. You can also use LMMFIND to return member statistics to you. If the data ID represents a concatenated set of ISPF libraries, LMMFIND finds the first occurrence of the member in the set of libraries.

The LMINIT and LMOPEN services must be completed before LMMFIND can be used.

### Command Invocation Format

```
ISPEXEC LMMFIND DATAID(data-id)
                MEMBER(member-name)

                [LOCK]

                [LRECL(lrecl-var)]

                [RECFM(recfm-var)]
```

```
                    [GROUP(group-var)]

                    [STATS(YES|NO)]
```

## Call Invocation Format

```
CALL ISPLINK ('LMMFIND ',data-id
                        ,member-name

                        ,['LOCK    ']

                        ,[lrecl-var]

                        ,[recfm-var]

                        ,[group-var]

                        ,['YES     '|'NO      ']);

OR

CALL ISPEXEC (buf-len, buffer);
```

## Parameters

**data-id**
The data ID associated with the data set to be searched. The data ID is generated by the LMINIT service. The maximum length of this parameter is 8 characters.

**member-name**
The name of the member to be found. The maximum length of this parameter is 8 characters.

**LOCK**
The LOCK parameter is no longer used since the removal of LMF from the ISPF product, but is left in for compatibility. If LOCK is specified, the LMMFIND service will fail with return code 12. If you want to be able to specify YES and have LMMFIND ignore the value, change the value of the FAIL_ON_LMF_LOCK keyword in the ISPF Configuration Table to NO.

The LMMFIND service is being used to find a member in an LMF-controlled ISPF library to either update it or prevent others from updating it. This parameter is valid only if the data set is an ISPF library or a concatenated set of ISPF libraries. The library management facility control data set must have been allocated under the ddname ISRCFIL. For more information about the LOCK parameter, refer to *ISPF Library Management Facility*.

**lrecl-var**
The name of the variable into which the data record length (or, if the record format is of variable length, the maximum data record length) is to be stored. The maximum length of this parameter is 8 characters.

**recfm-var**
The name of the variable into which the record format code is to be stored. An example is FB for fixed-length block data. The maximum length of this parameter is 8 characters.

**group-var**
The name of the variable that will store the name of the group that contains

the found member. This variable contains the group name after the service is executed only if the data is an ISPF library or a set of concatenated ISPF libraries and LMINIT is used with ISPF name parameters; otherwise, the variable is set to null. The maximum length of this parameter is 8 characters.

**YES | NO**

Whether statistics for the member are to be returned to the dialog invoking the service. If you specify NO (the default value), no statistics are returned. If you specify YES and the data ID represents a data set that has unformatted records (RECFM=U), the statistics are returned in the following dialog variables:

**ZLAC** A 2-character field containing the authorization code of the member.

**ZLALIAS** An 8-character field containing the name of the real member that this member is an alias of. If the member is not an alias this field is blank.

**ZLAMODE** A 3-character field containing the AMODE of the member.

**ZLATTR** A 20-character field containing the load module attributes. The attributes are 2-character strings separated by blanks. The following strings can appear in the attribute string:

- **NX** Not executable
- **OL** Only Loadable
- **OV** Overlay
- **RF** Refreshable
- **RN** Reentrant
- **RU** Reusable
- **SC** Scatter Load
- **TS** Test

**ZLLIB** Position in concatenated data set sequence; a number from 1 to 4.

**ZLRMODE** A 3-character field containing the RMODE of the member.

**ZLSIZE** An 8-character field containing the load module size in hex.

**ZLTTR** A 6-character field containing the TTR of the member.

**ZLSSI** An 8-character field containing the SSI information for a load module.

For other record formats (F or V), the statistics are returned in the following dialog variables:

**ZLC4DATE** Creation date in 4-character year format; a character value shown in your national format. If no value exists for this variable, PDF sets the value to blanks.

**ZLCDATE** Creation date; a character value shown in your national format. If no value exists for this variable, ISPF sets the value to blanks.

**ZLCNORC** Current number of records; a number from 0 to 65 535. If no value exists for this variable, ISPF sets the value to blanks.

**ZLINORC** Beginning number of records; a number from 0 to 65 535.

**ZLLIB** Position in concatenated data set sequence; a number from 1 to 4.

**ZLM4DATE** Last change date in 4-character year format; a character value shown in your national format. If no value exists for this variable, PDF sets the value to blanks.

**ZLMDATE** Last change date; a character value shown in your national format. If no value exists for this variable, ISPF sets the value to blanks.

**ZLMNORC** Number of changed records; a number from 0 to 65 535.

**ZLMOD** Modification level; a number from 0 to 99.

**ZLMSEC** Seconds value of the last change time. This is a two character field.

**ZLMTIME** Last change time; a character value in the format hh:mm.

**ZLUSER** User ID of last user to change the given member; an alphanumeric field with a maximum length of 7 characters.

**ZLVERS** Version number; a number from 1 to 99. If no value exists for this variable, ISPF sets the value to blanks.

**ZSCLM** Indicates whether the member was last modified by SCLM or PDF. A value of Y indicates the last update was made through SCLM. A value of N indicates that the last update was made through PDF.

The preceding variables are stored in the function pool and therefore become immediately available to command invocations. You cannot use the VGET service to retrieve these variables, since VGET accesses the shared and profile pools.

For an MVS partitioned data set, if the statistics are not stored in the data set directory in the same format used by Edit, only ZLLIB is set with the position in the concatenation.

**buf-len**
A fullword fixed binary integer containing the length of the buffer parameter.

**buffer**
A buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC invocation for a command invocation.

## Return Codes

The following return codes are possible:

**0** Normal completion.

**8** Member not found.

**10** No data set is associated with the given data ID; that is, LMINIT has not been completed.

**12** One of the following:
- Data set is not open or is not open for input.
- A parameter value is invalid.

- Data set is not partitioned.
- LOCK parameter was specified.

**16** A truncation or translation error occurred in accessing dialog variables.

**20** Severe error; unable to continue.

## Example

This example:

- Invokes the LMMFIND service to find member MYPROG in the data set associated with the data ID stored in DDVAR.
- Stores the record length in variable LENVAR, the record format code in FORMVAR, and the name of the group that contains member MYPROG in GRPVAR.

### Command Invocation

```
ISPEXEC LMMFIND DATAID(&DDVAR) MEMBER(MYPROG)         +
                LRECL(LENVAR)  RECFM(FORMVAR)         +
                GROUP(GRPVAR)
```

### Call Invocation

```
CALL ISPLINK ('LMMFIND ',DDVAR,'MYPROG ',' ',
              'LENVAR ','FORMVAR ','GRPVAR ');
```

*OR*

Set the program variable BUFFER to contain:

```
BUFFER = 'LMMFIND DATAID(&DDVAR) MEMBER(MYPROG)
                 LRECL(LENVAR)  RECFM(FORMVAR)
                 GROUP(GRPVAR)';
```

Set the program variable BUFLEN to the length of the variable BUFFER. Issue the following:

```
CALL ISPEXEC (BUFLEN, BUFFER);
```

# LMMLIST – List a Library's Members

The LMMLIST service, when used with the LIST or SAVE option, creates a list of the first occurrence of all the members in an ISPF library, a concatenated set of ISPF libraries, or an MVS partitioned data set associated with the given data ID.

When you invoke LMMLIST for the first time with the LIST option, the MEMBER variable determines the starting position within the member list. To position at the beginning, set the MEMBER variable to blanks. If the requested member is not found, the next member in the member list is returned. The member list is sorted by member name. Repeated innvocation of LMMLIST provides access to each member name in the member list.

Use LMMLIST with the SAVE option to write a list of member names to a data set. If a MEMBER variable is nonblank, the member name you specify will be the first member in the list.

You must complete the LMINIT and LMOPEN services before using LMMLIST. Use the LMMLIST FREE option to release the list storage space when it is not needed.

**Note:** Member lists generated by LMMLIST cannot be displayed by LMMDISP, and member lists generated by LMMDISP cannot be used with LMMLIST. Member lists should be freed when switching between LMMLIST and LMMDISP with the same data ID.

## Command Invocation Format

```
ISPEXEC LMMLIST DATAID(data-id)
                [OPTION(LIST|FREE|SAVE)]

                [MEMBER(member-var)]

                [STATS(YES|NO)]

                [GROUP(group)]

                [PATTERN(member-pattern)]

                [LONG]
```

## Call Invocation Format

```
CALL ISPLINK ('LMMLIST ',data-id
                        ,['LIST    '|'FREE    '|'SAVE    ']

                        ,[member-var]

                        ,['YES     '|'NO      ']

                        ,[group]

                        ,[member-pattern]

                        ,['LONG ']);

OR

CALL ISPEXEC (buf-len, buffer);
```

## Parameters

**data-id**
The data ID associated with the ISPF library, concatenated group of ISPF libraries, or MVS partitioned data set for which the member list is to be created. The data ID has been generated by the LMINIT service. The maximum length of this parameter is 8 characters.

**LIST | FREE | SAVE**
These options determine the action performed by the LMMLIST service.

**LIST** The first time that you invoke the LMMLIST service with the LIST option (the default value), it creates a member list for use by a dialog.

If member-var is initialized to blanks, the first name in the member list is returned. If member-var is set to a member name for a starting position within the member list, that member name is returned in member-var. If the member is not found, the next member in the member list is returned. If you request statistics information for the member, the statistics are returned.

Later invocations of LMMLIST with the LIST option return succeeding member names and their statistics, if requested, until the end of the list is reached, as indicated by return code. At this point, the dialog should invoke LMMLIST with the FREE option.

**FREE** The FREE option specifies that the storage acquired to create the member list is to be freed. Each creation of a member list should be matched by an invocation of LMMLIST with the FREE option.

**SAVE** The SAVE option writes all member names in a list specified by the data ID to a data set. The name of the data set is determined by the presence and value of the GROUP parameter.

**member-var**
The name of the variable into which the name of the member used for positioning in the member list is specified, or the name of the next member in the list is to be stored. The maximum length of this parameter is 8 characters.

When you invoke LMMLIST for the first time, member-var is used for selecting a starting position within the member list. If the member is found, that member name is returned in member-var. If the requested member is not found, the next member in the member list is returned. To start at the beginning of the list, set member-var to blanks.

The member-var parameter serves the same purpose for the SAVE option as it does for the LIST option. When LMMLIST is used with OPTION (SAVE), a list of member names is written to a data set. If member-var is nonblank, the member name you specify is the first member in the list.

**YES | NO**
The STATS parameter can only be used with the LIST and SAVE options. The default is STATS(NO). If you specify STATS(YES) the LMMLIST service provides member statistics with the member names. This parameter is fully described under "LMMFIND – Find a Library Member" on page 141.

**group**
This 8-character value specifies the group name of the data set that the LMMLIST service writes the member names list with the SAVE option. The entire data set name is <prefix>.<group>.MEMBERS. If you do not specify a group name the LMMLIST service writes to the ISPF LIST data set.

**Note:** LMMLIST service allocates the output data set with a DISP=OLD for the SAVE option.

**member-pattern**
The character string that is used to specify which members are to be returned. See the section on naming ISPF libraries and data sets in the *ISPF User's Guide Volume I* for a more complete description of patterns and pattern matching.

**LONG**
When SAVE is selected to save the member list to a dataset, LONG formats all dates in yyyy/mm/dd format for the member. Additionally, for PDS datasets not containing load libraries, the untranslated member name is written after the member name.

**buf-len**
A fullword fixed binary integer containing the length of the buffer parameter.

**buffer**
A buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC invocation for a command invocation.

## Return Codes

The following return codes are possible:

**0** One of the following:
- LIST option - Normal completion. The member list is available and the next member in the list is returned in the member-var parameter.
- FREE option - Normal completion. The member list is freed successfully.
- SAVE option - Normal completion. The member list is successfully written to a data set.

**4** Empty member list.

**8** One of the following:
- LIST option - End of member list.
- FREE option - Member list does not exist.
- SAVE option - For a data ID, the LMMLIST service has been invoked with the SAVE option after being invoked with LIST option, but before being invoked with the FREE option.

**10** No data set is associated with the given data ID; that is, LMINIT has not been completed.

**12** One of the following:
- The data set is not open or is not partitioned.
- A parameter value is invalid.
- Member list was created using LMMDISP.

**16** A truncation or translation error occurred in accessing dialog variables.

**20** Severe error; unable to continue.

## Example

This example invokes the LMMLIST service with the LIST option to create a member list of the data set associated with the data ID in variable DDVAR and to return the first member namein the list in variable MEMVAR.

### Command Invocation

In this example, the LMMLIST service LIST option creates a member list of the data set associated with the data ID in variable DDVAR and returns the first member name in the list to variable MEMVAR.

```
SET &MEMVAR =
ISPEXEC LMMLIST DATAID(&DDVAR) OPTION(LIST)         +
                MEMBER(MEMVAR)
```

### Call Invocation

```
MEMVAR = ' ';
CALL ISPLINK ('LMMLIST ',DDVAR,'LIST    ','MEMVAR ' );
OR
```

Set the program variable BUFFER to contain:

```
MEMVAR= ' ';
BUFFER = 'LMMLIST DATAID(&DDVAR) OPTION(LIST)
                 MEMBER(MEMVAR)';
```

Set the program variable BUFLEN to the length of the variable BUFFER. Issue the following command:

```
CALL ISPEXEC (BUFLEN, BUFFER);
```

In this example, the LMMLIST service SAVE option creates a member list, writes it to the ISPF LIST data set, using the data ID stored in IDVAR.

### Command Invocation

```
ISPEXEC LMMLIST DATAID(&IDVAR) STATS(YES) OPTION(SAVE)
```

### Call Invocation

```
CALL ISPLINK ('LMMLIST ',IDVAR,'SAVE    ','        ','YES     ');
OR
```

Set the program variable BUFFER to contain:

```
BUFFER = 'LMMLIST DATAID(&IDVAR) STATS(YES) OPTION(SAVE)';
```

Set the program variable BUFLEN to the length of the variable BUFFER. Issue the following command:

```
CALL ISPEXEC (BUFLEN, BUFFER);
```

## LMMOVE – Move Members of a Data Set

The LMMOVE service moves members of a partitioned data set or an entire sequential data set. Once the data has been moved, the "from" data set or members are deleted. Packing data, replacing members, and automatic truncation are optional. Only fixed and variable record format data sets can be packed.

Completion of the LMINIT service is required before you invoke LMMOVE. See "LMINIT – Generate a Data ID for a Data Set" on page 119 for information that can help prevent some common I/O errors that might occur when using the LMMOVE service. LMMOVE requires that the "to data-id" be closed before invocation. The "from data-id" must also be closed when moving sequential data sets.

**Notes:**

1. FROMID and TODATAID can refer to the same data set but they cannot have the same data-id.
2. LMMOVE does not support the copying of unmovable data sets (data set organization POU or PSU).
3. If the ALIAS option is in effect, LMMOVE automatically processes alias members as follows:
   - Either the main member or any alias member may be selected to move the main member and all of its aliases. This will occur even if some of the members are not displayed in the current member selection list.
   - Alias members are moved for both load and non-load data sets as well as for PDS and PDSE data sets.

   Moving to the same dataset is not supported when aliases are automatically selected, as it would result in the "from" and "to" member names being the same.
4. If the NOALIAS option is in effect, LMMOVE does not move alias members unless one of the following is true:
   - All members of the data set are selected.

- A member pattern is used and both the main member and the alias member are included in that pattern.

If the NOALIAS option is in effect, moving an alias member by itself will result in a new member being created, even if the main member has already been moved.

## Command Invocation Format

```
ISPEXEC LMMOVE FROMID(from-data-id)
               [FROMMEM(from-member-name)]

               TODATAID(to-data-id)

               [TOMEM(to-member-name)]

               [REPLACE]

               [PACK]

               [TRUNC]

               [SCLMSET(Y|N)]

               [ALIAS|NOALIAS]
```

## Call Invocation Format

```
CALL ISPLINK ('LMMOVE  ',from-data-id
                        ,[from-member-name]

                        ,to-data-id

                        ,[to-member-name]

                        ,['REPLACE ']

                        ,['PACK    ']

                        ,['TRUNC   ']

                        ,['YES'|NO'|' ']

                        ,['ALIAS   '|'NOALIAS ']);
```

*OR*

```
CALL ISPEXEC (buf-len,buffer);
```

## Parameters

**from-data-id**
Specifies the data ID name associated with the data set to be moved. The data ID has been generated by the LMINIT service. The maximum length of this parameter is 8 characters.

**from-member-name**
The member name or pattern of the members to be moved. An asterisk (*) indicates that all members are to be moved. If the "from" data set is partitioned, this parameter is required. If it is sequential, this parameter is not allowed. The maximum length of this parameter is 8 characters.

**to-data-id**
Specifies the data ID name associated with the data set being moved to. The data ID has been generated by the LMINIT service. The maximum length of this parameter is 8 characters.

**to-member-name**
The name of the member being moved to the "to" data set. If a name is not specified, the name of the member in the "from" data set is used. If the "from" data set is sequential and the "to" data set is partitioned, this parameter is required. If the "to" data set is sequential, this parameter is not allowed. The maximum length of this parameter is 8 characters.

**REPLACE**
Specifies whether like-named members in the "to" data set are to be replaced. If "replace" is not specified and the members exists in the "to" data set, then the move will not be performed and a return code of 20 is issued.

If a list of members is being moved and one cannot be replaced, processing stops and a message is issued indicating how many members were moved.

**PACK**
Specifies whether data is stored in the "to" data set in packed format. If "pack" is not specified, data is stored as it exists.

**TRUNC**
Specifies that truncation is to occur if the logical record length of the "to" data set is less than the logical record length of the "from" data set. If truncation is not specified and the logical record length of the "to" data set is less than the logical record length of the "from" data set, the move is not performed and a return code of 16 is issued.

**SCLMSET**
ISPF maintains a bit in the PDS directory to indicate whether a member was last modified using SCLM or some function outside of SCLM. The SCLMSET value indicates how to set this bit. YES indicates to set the bit ON. NO indicates the bit should be OFF. If you want to keep the current setting for a certain member, omit the SCLMSET parameter.

**ALIAS | NOALIAS**
With ALIAS in effect, either the main member or any alias member may be selected to move the main member and all of its aliases. This will occur even if a single member is specified or if some of the members are not displayed in the current member selection list.

With NOALIAS in effect, aliases must be moved manually to maintain the correct alias relationship. That is, the main member must be moved first followed by the aliases.

**buf-len**
Specifies a fullword fixed binary integer containing the length of *buffer*.

**buffer**
Specifies a buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC invocation for a command procedure.

## Return Codes

**0** Successful completion.

**4** One of the following:
- "From" data set is empty.

- No member matched the pattern in the "from" data set.

**8** "From" member not found.

**10** No data set is associated with given data ID.

**12** One of the following:

- A like-named member already exists in the "to" data set and the Replace option was not specified.
- One or more members of the 'TO' or 'FROM' data sets are "in use" by you or another user and could not be moved.
- Invalid data set organization.
- Data set attribute invalid for packed data.
- Open error.

**16** A truncation error occurred.

**20** Severe error; unable to continue.

## Example

This example invokes the LMMOVE service to move member MYPROG in the data set associated with the data ID in variable DDVAR to the data set associated with the data ID in variable DDVAR2. If MYPROG already exists, replace it.

### Command Invocation

```
ISPEXEC LMMOVE FROMID(&DDVAR) FROMMEM(MYPROG)        +
               TODATAID(&DDVAR2) REPLACE
```

### Call Invocation

```
CALL ISPLINK('LMMOVE  ',DDVAR,'MYPROG ',DDVAR2,' ','REPLACE ');
OR
```

Set the program variable BUFFER to contain:

```
BUFFER = 'LMMOVE FROMID(&DDVAR) FROMMEM(MYPROG)
                TODATAID(&DDVAR2) REPLACE';
```

Set the program variable BUFLEN to the length of the variable BUFFER. Issue the following command:

```
CALL ISPEXEC (BUFLEN, BUFFER);
```

# LMMREN – Rename a Data Set Member

The LMMREN service updates the directory to rename a member of a partitioned data set. You can use this service with an ISPF library or an MVS partitioned data set.The LMINIT service with either ENQ(SHRW) or ENQ(EXCLU) and the LMOPEN service with OPTION(OUTPUT) must be completed before you can use the LMMREN service.

## Command Invocation Format

```
ISPEXEC LMMREN DATAID(data-id)
               MEMBER(old-member-name)

               NEWNAME(new-member-name)

               [NOENQ]
```

## Call Invocation Format

```
CALL ISPLINK('LMMREN  ',data-id
                       ,old-member-name

                       ,new-member-name)

                       ,['NOENQ']);


OR

CALL ISPEXEC (buf-len, buffer);
```

## Parameters

**data-id**
The data ID associated with the data set that contains the member being renamed. The data ID has been generated by the LMINIT service. The maximum length of this parameter is 8 characters.

**old-member-name**
The present name of the member. The maximum length of this parameter is 8 characters.

**new-member-name**
The new member name, which must follow TSO data set naming conventions. The maximum length of this parameter is 8 characters.

**NOENQ**
An optional parameter that specifies that PDF should not issue its standard ENQ during the processing of this service. This standard ENQ consists of a major name of SPFEDIT and a minor name of the data set name and member. PDF by default will issue the ENQ unless NOENQ is specified.

**buf-len**
A fullword fixed binary integer containing the length of the buffer parameter.

**buffer**
A buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC invocation for a command invocation.

## Return Codes

The following return codes are possible:

**0** Normal completion.

**4** Directory already contains the specified new name.

**8** Member not found.

**10** No data set is associated with the given data ID; that is, LMINIT has not been completed.

**12** One of the following:
- The data set is not open or is not open for output.
- The parameter value is invalid.
- The data set organization is invalid.

**20** Severe error; unable to continue.

## Example

This example invokes the LMMREN service to rename member MYPROG in the data set associated with the data ID in variable DDVAR to MYPROGA.

### Command Invocation

```
ISPEXEC LMMREN DATAID(&DDVAR)      +
               MEMBER(MYPROG) NEWNAME(MYPROGA)
               [NOENQ]
```

### Call Invocation

```
CALL ISPLINK('LMMREN  ',DDVAR,'MYPROG ','MYPROGA ')
                       ,['NOENQ']);
```

*OR*

Set the program variable BUFFER to contain:

```
BUFFER = 'LMMREN DATAID(&DDVAR)
                MEMBER(MYPROG) NEWNAME(MYPROGA)';
```

Set the program variable BUFLEN to the length of the variable BUFFER. Issue the following command:

```
CALL ISPEXEC (BUFLEN, BUFFER);
```

---

# LMMREP – Replace a Member of a Data Set

The LMMREP service updates the directory to replace a member of a partitioned data set. The specified member is added if it does not currently exist. This service can be used with an ISPF library or an MVS partitioned data set. The LMINIT service with either ENQ(SHRW) or ENQ(EXCLU), the LMOPEN service with OPTION(OUTPUT), and the LMPUT service must be completed before you can use the LMMREP service.

## Command Invocation Format

```
ISPEXEC LMMREP DATAID(data-id)
               MEMBER(member-name)

               [STATS(YES|NO)]

               [NOENQ]
```

## Call Invocation Format

```
CALL ISPLINK('LMMREP  ',data-id
                       ,member-name

                       ,['YES     '|'NO      '])

                       ,['NOENQ   ']);
```

*OR*

```
CALL ISPEXEC (buf-len, buffer);
```

## Parameters

**data-id**
The data ID associated with the data set that contains a member that is being replaced. The data ID has been generated by the LMINIT service. The maximum length of this parameter is 8 characters.

**member-name**
The name of the member to be replaced. The maximum length of this parameter is 8 characters.

**YES | NO**
Whether the user data area in the directory should be updated so that the statistics of the member are stored in the same format used by Edit.

If you type YES and the data set specified is partitioned and the records are not unformatted (RECFM=U), the directory is updated with the member statistics. At least a valid creation date (ZLCDATE) and the date of the most recent change (ZLMDATE) must be provided in the member statistics.

If you specify NO, the default value, the statistics are not updated.

The following dialog variables are used to pass statistical information from the dialog invoking the LMMREP service:

**ZLVERS** Version number; a number from 1 to 99.

**ZLMOD** Change level; a number from 0 to 99.

**ZLCDATE** Creation date; a character value shown in your national format.

**ZLMDATE** Last change date; a character value shown in your national format.

**ZLMTIME** Last change time; a character value in the format hh:mm. ZLMTIME can also be specified as an 8-character field in the format hh:mm:ss. If the 6th character is not a colon, or if the 7th and 8th characters (ss) are not in the range '00' to '59', only the hour and minute specifications are used. The seconds value is set to the current time.

**ZLMSEC** *Seconds* value of the last change time. This is a 2-character field.

**Note:** If the ZLMTIME variable does not contain a *seconds* value and ZLMSEC is not set, the *seconds* value is set to `00`. If both ZLMTIME and ZLMSEC specify a *seconds* value, the value in ZLMSEC is used.

**ZLCNORC** Current number of records; a number from 0 to 65 535.

**ZLINORC** Beginning number of records; a number from 0 to 65 535.

**ZLMNORC** Number of changed records; a number from 0 to 65 535.

**ZLUSER** User ID of the last user to change the given member; the user ID can have a maximum length of 7 characters.

**NOENQ**
An optional parameter that specifies that PDF should not issue its standard ENQ during the processing of this service. This standard ENQ consists of a major name of SPFEDIT and a minor name of the data set name and member. PDF by default will issue the ENQ unless NOENQ is specified.

**buf-len**
A fullword fixed binary integer containing the length of the buffer parameter.

**buffer**
A buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC invocation for a command invocation.

## Return Codes

The following return codes are possible:

**0** Normal completion.

**8** Member is added; it did not previously exist.

**10** No data set is associated with the given data ID; that is, LMINIT has not been completed.

**12** One of the following:
- The data set is not open or is not open for output.
- The parameter value is invalid.
- The data set organization is invalid.
- Some member statistics have invalid values.

**14** No record has been written for the member to be replaced.

**16** Truncation or translation error in accessing dialog variables.

**20** Severe error; unable to continue.

## Example

This example invokes the LMMREP service to update the directory of the data set associated with the data ID in variable DDVAR to replace member MYPROG.

### Command Invocation

```
ISPEXEC LMMREP DATAID(&DDVAR) MEMBER(MYPROG)
```

### Call Invocation

```
CALL ISPLINK('LMMREP  ',DDVAR,'MYPROG ');
OR
```

Set the program variable BUFFER to contain:

```
BUFFER = 'LMMREP DATAID(&DDVAR) MEMBER(MYPROG)';
```

Set the program variable BUFLEN to the length of the variable BUFFER. Issue the following command:

```
CALL ISPEXEC (BUFLEN, BUFFER);
```

# LMMSTATS – Set and Store, or Delete ISPF Statistics

The LMMSTATS service sets and stores, or deletes ISPF statistics for members of a partitioned data set. This service can be used with ISPF libraries or an MVS partitioned data set. Any and all statistics can be set, or all statistics can be deleted. If no statistics exist, then LMMSTATS will calculate those not specified by keyword. Only fixed- and variable-record format data sets are supported. Completion of the LMINIT service is required before you invoke LMMSTATS. The data set must not be opened for output.

## Command Invocation Format

```
ISPEXEC LMMSTATS DATAID(data-id)
                 MEMBER(member-name)
                 [VERSION(version-number)]
                 [MODLEVEL(mod-level)]
                 [CREATED(create-date)]
                 [MODDATE(last-modified-date)]
                 [MODTIME(last-modified-time)]
                 [CURSIZE(current-size)]
                 [INITSIZE(initial-size)]
                 [MODRECS(records-modified)]
                 [USER(user-id)]
                 [DELETE]
                 [CREATED4(4-char-year-create-date)]
                 [MODDATE4(4-char-year-last-modified-date)]
                 [SCLM(On|Off|Asis)]
```

## Call Invocation Format

```
CALL ISPLINK ('LMMSTATS',data-id
                        ,member-name
                        ,[version-number]
                        ,[mod-level]
                        ,[create-date]
                        ,[last-modified-date]
                        ,[last-modified-time]
                        ,[current-size]
                        ,[initial-size]
                        ,[records-modified]
                        ,[user-id]
                        ,['DELETE  ']
                        ,[4-char-year-create-date]
                        ,[4-char-year-last-modified-date]
                        ,[On|Off|Asis]);
```

*OR*

```
CALL ISPEXEC (buf-len,buffer);
```

## Parameters

**data-id**
The data ID associated with the data set containing the members whose statistics are being modified or deleted. The data ID has been generated by the LMINIT service. The maximum length of this parameter is 8 characters.

**member-name**
Member name, or pattern representing the members whose statistics are to be modified or deleted. A pattern may be specified to indicate a subset of members all members. The maximum length of this parameter is 8 characters.

**version-number**
The number to be assigned as the version number. This parameter must be an integer between 1 and 99, inclusive.

**mod-level**
The number of modifications or changes to the member. This parameter must be an integer between 0 and 99, inclusive.

**create-date**
The date the member was created. The format of the date is dependent on the language in which ISPF is installed. The English format is YY/MM/DD.

**last-modified-date**
The date the member was last modified. The format of this parameter is the same as the create-date parameter.

**last-modified-time**
The time the member was last modified. This parameter should be specified as a character field and must be specified with 5 characters (for example - hh:mm). This parameter may also be specified as an 8-character field in the format hh:mm:ss. If the 6th character is not a colon, of if the 7th and 8th characters (ss) are not in the range '00' to '59', only the hour and minute specifications are used.

**current-size**
The current number of data records in the member. This parameter must be an integer between 0 and 65 535, inclusive.

**initial-size**
The original number of data records in the member when it was created. This parameter must be an integer between 0 and 65 535, inclusive.

**records-modified**
The number of data records modified in a member since it was created. This parameter must be an integer between 0 and 65 535, inclusive.

**user-id**
The user ID of the user that last modified the data. The maximum length of this parameter is 7 characters.

**DELETE**
PDF statistics are removed for the specified members.

**4-char-year-create-date**
The date that the member was created, in 4-character year format. The format of the date depends on the language in which ISPF and ISPF/PDF are invoked. The English format is YYYY/MM/DD.

**4-char-year-last-modified-date**
The date that the member was last changed, in 4-character year format. The format of the date depends on the language in which ISPF and ISPF/PDF are invoked. The English format is YYYY/MM/DD.

**SCLM**
The SCLM setting is a bit that ISPF uses to determine what type of edit the file last had performed upon it.

**On** The last edit of this file was under SCLM control.

**Off** The last edit of this file was under control of something other than SCLM.

**Asis** This LMMSTATS operation is transferring the current setting of this file as it already is.

**buf-len**
A fullword fixed binary integer containing the length of the buffer parameter.

**buffer**
A buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC invocation for a command procedure.

## Return Codes

The following return codes are possible:

**0** Normal completion.

**4** One of the following:
- Data set is empty.
- No members matched the pattern.

**8** Member not found.

**10** No data set is associated with the given data ID; that is, LMINIT has not been completed.

**12** One of the following:
- Invalid parameter value.
- Data set is not partitioned.
- Data ID represents a concatenation of data sets.
- Data set is opened for output.

**20** Severe error; unable to continue.

## Example

This example invokes LMMSTATS to set to 20 the version number of member MYPROG in the data set associated with the data ID stored in DDVAR.

### Command Invocation

```
ISPEXEC LMMSTATS DATAID(&DDVAR) MEMBER(MYPROG) VERSION(20)
```

### Call Invocation

```
CALL ISPLINK ('LMMSTATS',DDVAR,'MYPROG ',20);
```

*OR*

Set the program variable BUFFER to contain:

```
BUFFER = 'LMMSTATS DATAID(&DDVAR) MEMBER(MYPROG) VERSION(20)';
```

Set the program variable BUFLEN to the length of the variable BUFFER. Issue the following command:

```
CALL ISPEXEC (BUFLEN, BUFFER);
```

# LMOPEN – Open a Data Set

The LMOPEN service opens the data set associated with a given data ID so the data set can be either read from, using LMGET, or written to, using LMPUT. The LMINIT service must be completed before LMOPEN can be used.

For each LMOPEN invocation, you should invoke a matching LMCLOSE service. The LMCLOSE service closes the data set to further processing until LMOPEN is invoked again for that data set's data ID. Therefore, you should invoke the LMCLOSE service when processing is completed for that data set. Otherwise, unwanted data can be read from or written to the data set.

**Note:** Some library access services do not require that LMOPEN be executed before invocation (for example, LMCOPY and LMMOVE). Refer to the service description to determine whether LMOPEN should be invoked.

It is the responsibility of the dialog developer to ensure that a data set is opened for output only once. ISPF does not protect against this situation. From the time LMOPEN for output is invoked until LMCLOSE is invoked, there are certain restrictions on what can be done. Do not invoke the EDIT, DISPLAY, or TBDISPL services. Displaying any panel at all may allow the user to edit the already opened data set or invoke a dialog that opens the same data set for output.

## Command Invocation Format

```
ISPEXEC LMOPEN DATAID(data-id)
               [OPTION(INPUT|OUTPUT)]
               [LRECL(lrecl-var)]
               [RECFM(recfm-var)]
               [ORG(org-var)]
```

## Call Invocation Format

```
CALL ISPLINK ('LMOPEN  ',data-id
                        ,['INPUT   '|'OUTPUT  ']
                        ,[lrecl-var]
                        ,[recfm-var]
                        ,[org-var]);
OR
CALL ISPEXEC (buf-len, buffer);
```

## Parameters

**data-id**
The data ID associated with the data set to be opened. The data ID has been generated by the LMINIT service. The maximum length of this parameter is 8 characters.

**INPUT | OUTPUT**
Whether the data set is to be opened for reading or writing. INPUT is the default.

INPUT specifies that the dialog invoking the service uses the LMMFIND and LMGET services to read from the data set. The enqueue value for the LMINIT service can be SHR, EXCLU, or SHRW.

OUTPUT specifies that the dialog invoking the service uses LMPUT and either LMMADD or LMMREP to write to the data set, or uses LMMDEL or LMMREN to change the data set. The enqueue value for the LMINIT service can be EXCLU, SHRW, or MOD. If the data set is allocated SHRW, ISPF uses the RESERVE macro to reserve the DASD volume to the user when the data set is opened for output. This DASD volume remains reserved to the user until the LMCLOSE service is performed. The data ID must represent a single data set, not a concatenation of data sets.

**lrecl-var**
The name of a character variable into which the actual data record length or, if the record format is of variable length, the maximum data record length, is to be stored. This is an output parameter. The maximum length of this parameter is 8 characters.

**recfm-var**
The name of a character variable into which the record format code is to be stored. This is an output parameter. This variable must contain at least four characters. The maximum length of this parameter is 8 characters.

**org-var**
The name of a character variable into which the organization of the data is stored. This is an output parameter. The variable contains “PO” if the data set is partitioned and “PS” if it is physical sequential. The maximum length of this parameter is 8 characters.

**buf-len**
A fullword fixed binary integer containing the length of the buffer parameter.

**buffer**
A buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC invocation for a command invocation.

## Return Codes

The following return codes are possible:

**0** Normal completion.

**8** Data set could not be opened.

**10** No data set is associated with the given data ID; that is, LMINIT has not been completed.

**12** One of the following:
- The parameter value is invalid.
- Data set is already open.

- Cannot open concatenated data sets for output.
- Cannot open a data set allocated SHR for output.

**16** Truncation or translation error in accessing dialog variables.

**20** Severe error; unable to continue.

## Example

This example invokes the LMOPEN service to open the data set associated with the data ID in variable DDVAR for reading. The record length is to be returned in variable DLVAR, the record format in RFVAR, and the data set organization in ORGVAR.

### Command Invocation

```
ISPEXEC LMOPEN DATAID(&DDVAR) OPTION(INPUT)      +
               LRECL(DLVAR)   RECFM(RFVAR)       +
               ORG(ORGVAR)
```

### Call Invocation

```
CALL ISPLINK('LMOPEN  ',DDVAR,'INPUT   ',
                      'DLVAR ','RFVAR ',
                      'ORGVAR ');
```

*OR*

Set the program variable BUFFER to contain:

```
BUFFER = 'LMOPEN DATAID(&DDVAR) OPTION(INPUT)
                LRECL(DLVAR)   RECFM(RFVAR)
                ORG(ORGVAR)';
```

Set the program variable BUFLEN to the length of the variable BUFFER. Issue the following command:

```
CALL ISPEXEC (BUFLEN, BUFFER);
```

# LMPRINT – Print a Partitioned or Sequential Data Set

The LMPRINT service prints to the ISPF list data set an entire sequential or partitioned data set, certain specified members of a partitioned data set, or an index listing for a partitioned data set. The INDEX parameter can be used with fixed, variable, or undefined record formats. If the INDEX parameter is not used, the data set to be printed must be fixed or variable record format. Completion of the LMINIT service is required before you invoke LMPRINT.

## Command Invocation Format

```
ISPEXEC LMPRINT DATAID(data-id)

                [MEMBER(member-name)]

                [INDEX]

                [FORMAT(YES|NO)]
```

## Call Invocation Format

```
CALL ISPLINK ('LMPRINT ',data-id

                         ,[member-name]

                         ,['INDEX   ']
```

```
                                ,['YES     '|'NO      ']);

OR

CALL ISPEXEC (buf-len,buffer);
```

## Parameters

**data-id**
The data ID associated with the data set to be printed. The data ID has been generated by the LMINIT service. The maximum length of this parameter is 8 characters.

**member-name**
The member name or pattern of the members to be printed. An asterisk(*) indicates that all members should be printed. If an index print is requested, member-name must not be specified. The maximum length of this parameter is 8 characters. For more information about patterns and pattern matching, see the *ISPF User's Guide Volume I*.

**INDEX**
Indicates that only the index will be printed.

**YES | NO**
Indicates if the output is to be formatted. The default is YES.

**buf-len**
A fullword fixed binary integer containing the length of the buffer parameter.

**buffer**
A buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC invocation for a command procedure.

## Return Codes

The following return codes are possible:

**0** Normal completion.

**4** One of the following:
- Data set is empty or contains an empty member.
- No members matched the pattern.

**8** Member not found.

**10** No data set associated with given data ID.

**12** One of the following:
- Invalid data set organization; must be partitioned or sequential.
- Invalid parameter.

**20** Severe error; unable to continue.

## Example

This example invokes the LMPRINT service to print the sequential data set associated with the data ID in variable DDVAR, with no formatting of output.

### Command Invocation

```
ISPEXEC LMPRINT DATAID(&DDVAR) FORMAT(NO)
```

### Call Invocation

```
CALL ISPLINK('LMPRINT ',DDVAR,' ',' ','NO      ');
```

*OR*

Set the program variable BUFFER to contain:

```
BUFFER = 'LMPRINT DATAID(&DDVAR) FORMAT(NO)';
```

Set the program variable BUFLEN to the length of the variable BUFFER. Issue the following command:

```
CALL ISPEXEC (BUFLEN, BUFFER);
```

## LMPUT – Write a Logical Record to a Data Set

The LMPUT service writes one logical record to the data set associated with a given data ID. The first LMPUT writes the first logical record to the data set, and later invocations write succeeding records. The LMINIT service with ENQ(EXCLU), ENQ(SHRW), ENQ(MOD), and the LMOPEN service with the OUTPUT option must be completed before you can use the LMPUT service.

If the data set is an ISPF library or MVS partitioned data set, the LMMADD or LMMREP service must be invoked after the last LMPUT to update the directory and to write the last physical record.

If the data set is sequential, the LMCLOSE service must be invoked after the last LMPUT to write the last physical record and to close the data set.

When MODE(MULTX) is used, the write operation occurs in segments (rather than in single records), with each segment comprising multiple records. Each record is prefixed by a 2-byte binary integer field containing its length. The maximum size of each segment written is 32 000 bytes. LMPUT requires data in the dataloc-var to be in the following format:

```
+------+--------------+------+--------------+------+--------------+
|len   | record       |len   | record       |len   | record       |
+------+--------------+------+--------------+------+--------------+
<----     data-length should be set to this length             --->
<----     if less than, then fewer records are written         --->
<----     if greater than, then data-length is ignored         --->
```

The LMPUT service writes records to a data set as is. That is, the LMPUT service does not pack data before writing it if the data is in unpacked format. In order to pack data before writing it, use Edit with the pack option.

### Command Invocation Format

```
ISPEXEC LMPUT DATAID(data-id)

              MODE(INVAR|MOVE|MULTX)

              DATALOC(dataloc-var)

              DATALEN(data-length)

              [NOBSCAN]
```

## Call Invocation Format

```
CALL ISPLINK ('LMPUT   ',data-id
                        ,'INVAR   '|'MOVE    '|'MULTX   '
                        ,dataloc-var
                        ,data-length
                        ,' '
                        ,['NOBSCAN ']);
```

*OR*

```
CALL ISPEXEC (buf-len, buffer);
```

## Parameters

**data-id**
The data ID associated with the data set into which the record is to be written. The data ID has been generated by the LMINIT service. The maximum length of this parameter is 8 characters.

**INVAR|MOVE|MULTX**
INVAR mode shows that the data-location parameter variable contains the data itself; MOVE mode shows that the data-location parameter contains the address of the data to be written. A command dialog can use INVAR and MULTX modes, but not MOVE mode.

**dataloc-var**
The name of a variable that, on entry to the LMPUT service, contains either the data to be written (INVAR mode) or the fullword binary virtual storage address of the data to be written (MOVE mode).

The value of the variable passed from a program function can be either the data record itself or the address of the data record, but it must be consistent with the INVAR|MOVE specification above. If the variable was passed from a command function written in CLIST or REXX, it must *always* contain the data record. The maximum length of this parameter is 8 characters.

**data-length**
The length in bytes of the logical record to be written. The parameter must be a positive nonzero integral value. If the data-length specification causes a DBCS character string to be divided in the middle, the result may be unpredictable. In MULTX mode, the minimum of the data-length parameter and the length of data written to the variable pool determines how much of the data-loc variable is processed into records. The length field before each record determines the amount of data written, not exceeding the data set record length, to each record.

**NOBSCAN**
The No Backscan option; no truncation of trailing blanks for records of variable length occurs.

**buf-len**
A fullword fixed binary integer containing the length of the buffer parameter.

**buffer**
A buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC invocation for a command invocation.

## Return Codes

The following return codes are possible:

**0** Normal completion.

**10** No data set is associated with the given data ID; that is, LMINIT has not been completed.

**12** One of the following:
- The data set is not open or is not open for output.
- The parameter value is invalid.

**16** Truncation or translation error in accessing dialog variables.

**20** Severe error; unable to continue.

## Example

This example invokes the LMPUT service to write a data record, with a length of 80 bytes, contained in variable DATAVAR into the data set associated with the data ID in variable DDVAR.

### Command Invocation

```
ISPEXEC  LMPUT DATAID(&DDVAR)   MODE(INVAR)     +
               DATALOC(DATAVAR) DATALEN(80)
```

### Call Invocation

```
DATALEN=80;
CALL ISPLINK('LMPUT',DDVAR,'INVAR ','DATAVAR ',DATALEN);
Where DATALEN is a fullword integer variable.
```

*OR*

Set the program variable BUFFER to contain:

```
BUFFER = 'LMPUT DATAID(&DDVAR)   MODE(INVAR)
                DATALOC(DATAVAR) DATALEN(80)';
```

Set the program variable BUFLEN to the length of the variable BUFFER. Issue the following command:

```
CALL ISPEXEC (BUFLEN, BUFFER);
```

**Note:** Null variables must be defined to have a length greater than zero. Programs containing definitions of null variables must specify VDEFINE with the NOBSCAN option. Null variables defined in CLISTs should be initialized with the &STR built-in function. Null variables defined in REXX should be initialized with x = ' ', where x is the name of the variable.

## Example (MULTX)

This REXX example invokes the LMPUT service to process a data set in MULTX mode, writing blocks of data in segments no larger than 32 00 bytes. LMPUT pads records that are too short with blanks and truncates records that are too long for the target data set.

```
/* REXX to write out some data to a VB dataset */
 REC = ''
 DLEN = 0
 DO I = 1 TO 100
   X = 5 * I
   A = 'DATA LINE 'I' '
   DO J = 1 TO X
     A = A || 'D'
   END
   RLEN = LENGTH(A)
   NLEN = DLEN + RLEN + 2
   IF NLEN > 32000 THEN DO
      /* WRITE CURRENT BUFFER BEFORE IT GETS TOO BIG */
      'LMPUT DATAID('TESTFILE') MODE(MULTX) DATALOC(REC) ,
      DATALEN('DLEN')'
      IF RC > 0 THEN I = 1000
      REC = ''
      DLEN = RLEN + 2
   END
   ELSE DLEN = NLEN
   RLEN = D2C(RLEN,2)
   REC = REC || RLEN || A
 END
 /* WRITE LAST BUFFER */
 'LMPUT DATAID('TESTFILE') MODE(MULTX) DATALOC(REC) ,
  DATALEN('DLEN')'
```

## LMQUERY – Give a Dialog Information about a Data Set

The LMQUERY service returns values specified for the LMINIT service parameters that are associated with a given data ID. In this way, LMQUERY provides the dialog with selected information about a data set by showing how the LMINIT parameters were set up when the data ID of that data set was generated.

The service sets the contents of the variables named with the information being requested. Blanks are returned in a given variable if no value applies. For example, if DATASET was not used in LMINIT, DATASET in LMQUERY would have blanks.

### Command Invocation Format

```
ISPEXEC LMQUERY DATAID(data-id)

        [PROJECT(proj-var)]

        [GROUP1(group1-var)]

        [GROUP2(group2-var)]

        [GROUP3(group3-var)]

        [GROUP4(group4-var)]

        [TYPE(type-var)]

        [DATASET(dsn-var)]

        [DDNAME(ddn-var)]

        [VOLUME(serial-var)]

        [ENQ(enq-var)]

        [OPEN(open-var)]
```

```
                    [LRECL(lrecl-var)]
                    [RECFM(recfm-var)]
                    [DSORG(dsorg-var)]
                    [ALIAS(alias-var)]
                    [PASSWORD(password-var)]
|                   [OVOLUME(ovolume-var)]
```

## Call Invocation Format

```
CALL ISPLINK ('LMQUERY ',data-id
                        ,[proj-var]
                        ,[group1-var]
                        ,[group2-var]
                        ,[group3-var]
                        ,[group4-var]
                        ,[type-var]
                        ,[dsn-var]
                        ,[ddn-var]
                        ,[serial-var]
                        ,[enq-var]
                        ,[open-var]
                        ,[lrecl-var]
                        ,[recfm-var]
                        ,[dsorg-var]
                        ,[alias-var]
                        ,[password-var]
|                       ,[ovolume-var]);
OR
CALL ISPEXEC (buf-len, buffer);
```

## Parameters

**data-id** The data ID associated with the data set for which information is being requested. The data ID has been generated by the LMINIT service. The maximum length of this parameter is 8 characters.

**project-var** The name of an 8-character variable into which the value of the PROJECT parameter specified on the LMINIT service will be placed.

**group1-var** The name of an 8-character variable into which the value of the GROUP1 parameter specified on the LMINIT service will be placed.

**group2-var** The name of an 8-character variable into which the value of the GROUP2 parameter specified on the LMINIT service will be placed.

**group3-var** The name of an 8-character variable into which the value of the GROUP3 parameter specified on the LMINIT service will be placed.

**group4-var** The name of an 8-character variable into which the value of the GROUP4 parameter specified on the LMINIT service will be placed.

**type-var** The name of an 8-character variable into which the value of the TYPE parameter specified on the LMINIT service will be placed.

**dataset-var** The name of a 44-character variable into which the value of the DATASET parameter specified on the LMINIT service will be placed.

**ddname-var** The name of an 8-character variable into which the value of the DDNAME to which the data

**dsorg-var** The name of a 2-character variable into which the data set organization (PO or PS) will be placed.

**alias-var** The name of a 1-character variable into which an indicator will be placed to indicate whether the data set name originally specified was an alias name. Values of Y or N will be returned.

**password-var** The name of an 8-character variable into which the value of the PASSWORD parameter specified on the LMINIT service will be placed.

**ovolume-var** The name of an 8-character variable into which the volume on which the data set resides will be placed. This variable will have a value even if no VOLUME parameter was specified on the LMINIT service call.

**buf-len** A fullword fixed binary integer containing the length of the buffer parameter.

**buffer** A buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC invocation for a command invocation.

Blanks are returned in any variable for which there is no applicable value.

## Return Codes

The following return codes are possible:

**0** Normal completion.

**4** No applicable information available for a specified keyword; blanks are returned.

**10** No data set is associated with the given data ID; that is, LMINIT has not been completed.

**16** Truncation or translation error in accessing dialog variables.

**20** Severe error; unable to continue.

## Example

This example invokes the LMQUERY service to provide information about the ISPF library associated with the data ID in variable DDVAR. The data ID is created by using the LMINIT service with an ISPF library name. They use the following variables:

**PRJV** Highest-level qualifier of the libraries.

**GRP1V, GRP2V, GRP3V, and GRP4V**
Second-level qualifiers of the libraries.

**TYPEV** Third-level qualifier of the libraries.

### Command Invocation

```
ISPEXEC LMQUERY DATAID(&DDVAR) PROJECT(PRJV) GROUP1(GRP1V) +
                GROUP2(GRP2V) GROUP3(GRP3V) GROUP4(GRP4V)  +
                TYPE(TYPEV)
```

### Call Invocation

```
CALL ISPLINK('LMQUERY ',DDVAR,'PRJV ','GRP1V ','GRP2V ',
                       'GRP3V ','GRP4V ','TYPEV ');
```

*OR*

Set the program variable BUFFER to contain:

```
BUFFER = 'LMQUERY DATAID(&DDVAR) PROJECT(PRJV) GROUP1(GRP1V)
                 GROUP2(GRP2V) GROUP3(GRP3V) GROUP4(GRP4V)
                 TYPE(TYPEV)';
```

Set the program variable BUFLEN to the length of the variable BUFFER. Issue the following command:

```
CALL ISPEXEC (BUFLEN, BUFFER);
```

## LMRENAME – Rename an ISPF Library

The LMRENAME service renames an ISPF library or an MVS data set with a three-level qualified data set name. The data set name used must be the cataloged name, not an alias data set name.

### Command Invocation Format

```
ISPEXEC LMRENAME PROJECT(project)

                 GROUP(group)

                 TYPE(type)

                 {[NEWPROJ(new-project)]
                  [NEWGROUP(new-group)]
                  [NEWTYPE(new-type)]   }
```

### Call Invocation Format

```
CALL ISPLINK('LMRENAME',project

                       ,group

                       ,type

                       {,[new-project]
                        ,[new-group]
                        ,[new-type]   });
```

*OR*

```
CALL ISPEXEC (buf-len, buffer);
```

### Parameters

**project**
The highest-level qualifier in the specification of an ISPF library or MVS data set with a three-level qualified data set name. The maximum length of this parameter is 8 characters.

**group**
The second-level qualifier in the specification of an ISPF library or MVS data set with a three-level qualified data set name. The maximum length of this parameter is 8 characters.

**type**
The third-level qualifier in the specification of an ISPF library or MVS data set with a three-level qualified data set name. The maximum length of this parameter is 8 characters.

**new-project**
The new highest-level qualifier. If this parameter is not specified, the project parameter value is used. The maximum length of this parameter is 8 characters.

**new-group**
The new second-level qualifier. If this parameter is not specified, the group parameter value is used. The maximum length of this parameter is 8 characters.

**new-type**
The new third-level qualifier. If this parameter is not specified, the type parameter value is used. The maximum length of this parameter is 8 characters.

**Note:** You must specify either new-project, new-group, or new-type.

**buf-len**
A fullword fixed binary integer containing the length of the buffer parameter.

**buffer**
A buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC invocation for a command invocation.

## Return Codes

The following return codes are possible:

**0** Normal completion.

**4** New name already exists.

**8** One of the following:
- Specified data set does not exist.
- Rename or catalog failed.
- Data set name is an alias.

**12** The parameter value is invalid.

**20** Severe error; unable to continue.

## Example

This example invokes the LMRENAME service to rename a data set with the name DEPT877.PRIVATE.ASSEMBLE to DEPT877.MINE.ASSEMBLE.

### Command Invocation

```
ISPEXEC LMRENAME PROJECT(DEPT877)     +
                 GROUP(PRIVATE)       +
                 TYPE(ASSEMBLE)       +
                 NEWGROUP(MINE)
```

### Call Invocation

```
CALL ISPLINK('LMRENAME','DEPT877 ',
                        'PRIVATE ',
                        'ASSEMBLE',' ',
                        'MINE ');
```

*OR*

Set the program variable BUFFER to contain:

```
BUFFER = 'LMRENAME PROJECT(DEPT877)
                   GROUP(PRIVATE)
                   TYPE(ASSEMBLE)
                   NEWGROUP(MINE)';
```

Set the program variable BUFLEN to the length of the variable BUFFER. Issue the following command:

```
CALL ISPEXEC (BUFLEN, BUFFER);
```

## LOG – Write a Message to the Log Data Set

The LOG service causes a message to be written to the ISPF log data set.

The log data set, if allocated, is normally processed when you exit ISPF. A LOG command is available to allow you to process the log data set without exiting ISPF.

### Command Invocation Format

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

### Call Invocation Format

```
CALL  ISPEXEC (buf-len, buffer);
```

*OR*

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

### Parameters

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

**buf-len**
Specifies a fullword fixed binary integer containing the length of "buffer."

**buffer**
Specifies a buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC call for a command procedure.

### Return Codes

The following return codes are possible:

**0** Normal completion.

**12** The message-id contains invalid syntax or was not found.

**20** Severe error.

### Example 1

In a CLIST, dialog variable TERMSG contains a message-id. Write this message in the ISPF log file.

```
ISPEXEC LOG MSG(&TERMSG )
```

### Example 2

In a PL/I program, program variable TERMSG contains a message-id. The variable TERMSG has been made accessible to ISPF by a previous VDEFINE operation. Write this message in the ISPF log file. Set the program variable BUFFER to contain:

```
LOG MSG(&TERMSG)
```

Set program variable BUFLEN to the length of the variable BUFFER. Issue the following command:

```
CALL ISPEXEC (BUFLEN, BUFFER);
```

or alternately

```
CALL ISPLINK ('LOG     ',TERMSG);
```

### Example 3

Write message ABCX013 in the ISPF log file.

```
ISPEXEC LOG MSG(ABCX013)
```

Set the program variable BUFFER to contain:

```
LOG MSG(ABCX013)
```

Set program variable BUFLEN to the length of the variable BUFFER. Issue the following command:

```
CALL ISPEXEC (BUFLEN, BUFFER);
```

or alternately

```
CALL ISPLINK ('LOG     ','ABCX013 ');
```

## MEMLIST – Member List Dialog Service

The MEMLIST service enables you to access the Library Utility member list from within a dialog.

When you invoke the MEMLIST service, a member list is displayed with either a 1-character or 9-character line command area. You can perform any of the Library Utility functions, such as Edit, Browse, View, Print, Delete, and Rename, from within the member list. If the line command field is 9 characters, you can also invoke TSO commands against the selected member.

The MEMLIST service is given a data-id that has been associated with a partitioned data set or concatenation of partitioned data sets by the LMINIT service. The data-id must be freed by the LMFREE service.

## Command Invocation Format

```
ISPEXEC  MEMLIST DATAID(data-id)
           [MEMBER(pattern)]
           [CONFIRM(YES|NO)]
           [PANEL(panel-name)]
           [FIELD(1|9)]
```

## Call Invocation Format

```
CALL  ISPEXEC (buf-len, buffer);

OR

CALL  ISPLINK   ('MEMLIST ', data-id,

                     ,[pattern]

                     ,['YES'|'NO']

                     ,[panel-name]

                     ,['1'|'9']);
```

## Parameters

**data-id**
Specifies the variable in which the data ID that uniquely identifies the partitioned data set is stored.

**pattern**
A character string that is used to specify which members are to be displayed. If no pattern is specified, the entire list is displayed.

**YES | NO**
Specifies whether the Confirm Delete panel appears when you try to delete a member from the data set you specified. YES is the default.

If YES is specified, ISPF displays the Confirm Delete panel.

If NO is specified, ISPF does not display the Confirm Delete panel. The member is deleted with no additional action on your part.

**panel-name**
The name of the panel to use for displaying the member list. This can be a customized panel that you provide. Refer to *ISPF Planning and Customizing* for more information about developing a customized panel. If this parameter is omitted, the default panel is ISRUDMM if FIELD is set to 1, and ISRUDSM if FIELD is set to 9.

**1 | 9**
Indicates the length of the line command field on the member list display. The default is 9.

**buf-len**
Specifies a fullword fixed binary integer containing the length of "buffer."

**buffer**
Specifies a buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC call for a command procedure.

## Return Codes

The following return codes are possible:

**0** Normal completion.

**8** The requested data set was empty or no members matched the specified pattern.

**10** No data set is associated with the given data ID. LMINIT has not been completed.

**12** Indicates one of the following conditions:

- Data set not partitioned.
- Parameter value not valid.
- Invocation syntax not valid.

**16** A truncation or translation error occurred in accessing dialog variables.

**20** Severe error.

## Example

The following example shows an invocation of MEMLIST that displays the member list of a partitioned data set with the Delete Data Set Confirmation panel. The variable ID contains a data-id generated by the LMINIT service.

**Command Invocation**

```
ISPEXEC MEMLIST DATAID(&ID) CONFIRM(YES)
```

**Call Invocation**

```
CALL ISPLINK ('MEMLIST ',ID,'YES     ');
```

or alternately

Set the program variable BUFFER to contain:

```
BUFFER='MEMLIST DATAID(&ID) CONFIRM(YES)';
```

Set program variable BUFLEN to the length of the variable BUFFER. Issue the following:

```
CALL ISPEXEC (BUFLEN, BUFFER);
```

# PQUERY – Obtain Panel Information

The PQUERY service returns information for a specified area on a specific panel. The type, DYNAMIC or GRAPHIC, size, and position characteristics associated with the area are returned in variables.

## Command Invocation Format

```
ISPEXEC PQUERY PANEL(panel-name) AREANAME(area-name)
                                 [AREATYPE(area-type-name)]
                                 [WIDTH(area-width-name)]
                                 [DEPTH(area-depth-name)]
                                 [ROW(row-number-name)]
                                 [COLUMN(column-number-name)]
```

## Call Invocation Format

```
CALL  ISPEXEC (buf-len, buffer);

OR

CALL  ISPLINK   ('PQUERY  ', panel-name, area-name
                                        [,area-type-name]
                                        [,area-width-name]
                                        [,area-depth-name]
                                        [,row-number-name]
                                        [,column-number-name]);
```

## Parameters

**panel-name**
Specifies the name of the panel for which information is desired.

**area-name**
Specifies the name of an area within the panel whose attributes are to be returned.

**area-type-name**
Specifies the name of a variable in which the area type is to be stored. Either DYNAMIC or GRAPHIC is returned left-justified and padded with blanks. Nulls are returned if the return code is nonzero.

**area-width-name**
Specifies the name of a variable in which the number of columns in the area is to be stored. For a call, the variable should be defined as a fullword fixed integer.

**area-depth-name**
Specifies the name of a variable in which the number of rows comprising the area is to be stored. For areas that are not extendable (EXTEND(OFF)), this is the number of rows of the rectangular area in the panel definition. For extendable areas (EXTEND(ON)), this is the number of rows in the area after the panel body has been automatically extended to the depth of the physical screen on which the PQUERY service request is being issued. When issuing a PQUERY service request in the batch environment, the screen depth is specified as the value of the BATSCRD parameter on the ISPSTART call. For a call, the variable should be defined as a fullword fixed integer.

**row-number-name**
Specifies the name of a variable in which the number of the row of the top left position of the area is to be stored. For a call, the variable should be defined as a fullword fixed integer.

**column-number-name**
Specifies the name of a variable in which the number of the column of the top left position of the area is to be stored. For a call, the variable should be defined as a fullword fixed integer.

**buf-len**
Specifies a fullword fixed binary integer containing the length of “buffer.”

**buffer**
Specifies a buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC call for a command procedure.

If the panel uses a variable for the WIDTH keyword value on the BODY header, such as )BODY WIDTH(&WID), that variable must be set before invoking the PQUERY service.

## Return Codes

The following return codes are possible:

**0** Normal completion

**8** The panel does not contain the specified area.

**12** The specified panel cannot be found.

**16** Not all are values returned because insufficient space was provided.

**20** Severe error.

## Example

For the area named AREA1 on panel XYZ, return the number of columns in variable PQCOLS and the area type in variable ATYPE.

```
ISPEXEC PQUERY PANEL(XYZ) AREANAME(AREA1)
AREATYPE(ATYPE) WIDTH(PQCOLS)
```

Set the program variable BUFFER to contain:

```
PQUERY PANEL(XYZ) AREANAME(AREA1) AREATYPE(ATYPE) WIDTH(PQCOLS)
```

Set program variable BUFLEN to the length of the variable BUFFER. Issue the following command:

```
CALL ISPEXEC (BUFLEN, BUFFER);
```

or alternately

```
CALL ISPLINK ('PQUERY  ','XYZ ','AREA1 ',
              'ATYPE ','PQCOLS ');
```

# QBASELIB – Query Base Library Information

The QBASELIB service enables an ISPF dialog to obtain the current Library information for a specified DDNAME. For a specified ddname, the data set names allocated to that ddname are returned in a dialog variable.

## Command Invocation Format

```
ISPEXEC QBASELIB dd-name [ID(id-var)]
```

## Call Invocation Format

```
CALL ISPEXEC (buf-len, buffer);
```

*OR*

```
CALL ISPLINK ('QBASELIB',dd-name,[,id-var)]
```

## Parameters

**dd-name**
Specifies the ddname that is being queried. The value can be ISPPLIB, ISPMLIB, ISPSLIB, ISPTLIB, ISPLLIB, ISPTABL, ISPFILE, or any valid base DDNAME.

**id-var**
Optional parameter that specifies the name of a dialog variable which is to contain ″ID″ information. It is set to the data set name or names of the ddname that was specified in the service call. All data set names returned are fully qualified. Multiple data set names are separated by a comma. TSO has a maximum of 255 data set names allowed in the data set list. A data set name list is bounded by parenthesis when the QBASELIB service is requested through ISPLINK. The variable is not modified if the ddname specified is not allocated. It is the responsibility of the dialog developer to initialize this variable.

**Note:** Id-var should be initialized to blanks before every QBASELIB call.

**buf-len**
Specifies a fullword fixed binary integer containing the length of ″buffer.″

**buffer**
Specifies a buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC call for a command procedure.

## Return Codes

The following return codes are possible:

**0** A DDNAME for the specified ddname exists and the requested information has been successfully returned.

**4** The specified dd-name is not defined.

**16** A dialog variable translation or truncation error has occurred.

**20** A severe error has occurred.

## Example

A base library for messages (ISPMLIB) is defined. Query the ″ID″ information and return the ″ID″ information in the variable ***IDV***.

**Command Invocation**

```
ISPEXEC QBASELIB ISPMLIB ID(IDV)
```

**Call Invocation**

Set the program variable BUFFER to contain:

```
QBASELIB ISPMLIB ID(IDV)
```

Set program variable BUFLEN to the length of the variable BUFFER. Issue the following:

```
CALL ISPEXEC (BUFLEN, BUFFER);
```

or alternately

```
CALL ISPLINK ('QBASELIB','ISPMLIB ','IDV ');
```

# QLIBDEF – Query LIBDEF Definition Information

The QLIBDEF service allows an ISPF dialog to obtain the current LIBDEF definition information. This information can be saved by the dialog and used later to restore any LIBDEF definitions that may have been overlaid. For each LIBDEF lib-type, the ID parameter and the ″type″ of ID is returned. The absence of an active LIBDEF definition for a specific lib-type is indicated by the return code.

## Command Invocation Format

```
ISPEXEC QLIBDEF lib-type [TYPE(type-var)] [ID(id-var)]
```

## Call Invocation Format

```
CALL ISPEXEC (buf-len, buffer);
```

*OR*

```
CALL ISPLINK ('QLIBDEF ',lib-type,[,type-var][,id-var)]
```

## Parameters

**lib-type**
Specifies the LIBDEF lib-type definition that is being queried. The value may be ISPPLIB, ISPMLIB, ISPSLIB, ISPTLIB, ISPLLIB, ISPILIB, ISPTABL, ISPFILE, or a generic name. The values that may be specified on a LIBDEF service may be specified on a QLIBDEF service.

**type-var**
Optional parameter that specifies the name of a dialog variable which is to contain the ″type″ of LIBDEF definition. The possible values returned are DATASET, EXCLDATA, LIBRARY or EXCLLIBR. The variable is not modified if there is no LIBDEF. It is the responsibility of the dialog developer to initialize this variable.

**Note:** Type-var should be initialized to blanks before every QLIBDEF call.

**id-var**
Optional parameter that specifies the name of a dialog variable which is to contain ″ID″ information. It is set to the ddname or data set name or names that were specified on the last active LIBDEF service. All data set names returned are fully qualified, even if the original LIBDEF request did not specify fully qualified names. Multiple data set names are separated by a comma. The LIBDEF service has a maximum of 15 data set names allowed in the data set list. A data set name list is bounded by parenthesis when the QLIBDEF service is requested through ISPLINK. The variable is not modified if there is no LIBDEF in effect. It is the responsibility of the dialog developer to initialize this variable.

**Note:** Id-var should be initialized to blanks before every QLIBDEF call.

**buf-len**
Specifies a fullword fixed binary integer containing the length of ″buffer.″

**buffer**
Specifies a buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC call for a command procedure.

Although not mandatory, it is suggested that the service interface (ISPLINK or ISPEXEC) used by the QLIBDEF be the same as that used on the LIBDEF service to restore the definition. This eliminates the need to adjust the syntax of the information returned by QLIBDEF.

## Return Codes

The following return codes are possible:

**0** A LIBDEF definition for the specified lib-type exists and the requested information, if any, has been successfully returned.

**4** The specified lib-type does not have an active LIBDEF definition.

**12** An invalid lib-type value of ISPPROF has been specified.

**16** A dialog variable translation or truncation error has occurred.

**20** A severe error has occurred.

## Example

A panel library, ISPPLIB has been defined by the LIBDEF service. Query the type of LIBDEF definition and the LIBDEF ″ID″ information and return the type of LIBDEF definition in the variable, TYPEV, and the LIBDEF ″ID″ information in the variable, IDV.

```
ISPEXEC QLIBDEF ISPPLIB TYPE(TYPV) ID(IDV)
```

Set the program variable BUFFER to contain:

```
QLIBDEF ISPPLIB TYPE(TYPV) ID(IDV)
```

Set program variable BUFLEN to the length of the variable BUFFER. Issue the following:

```
CALL ISPEXEC (BUFLEN, BUFFER);
```

or alternately

```
CALL ISPLINK ('QLIBDEF ','ISPPLIB ','TYPEV ','IDV ');
```

# QTABOPEN – Query Open ISPF Tables

The QTABOPEN service allows an ISPF dialog to obtain a list of currently open ISPF tables. The TBSTATS or TBQUERY service can then be used to obtain more detailed information about each table.

## Command Invocation Format

```
ISPEXEC QTABOPEN  LIST(list-var)
```

## Call Invocation Format

```
CALL ISPLINK  ('QTABOPEN ',list-var);
```

## Parameters

*list-var*
Specifies the prefix to be used to construct the names of ISPF variables which contain the list of open tables. Each variable name is contructed by appending a sequence number to the prefix. The total number of variables created is returned in a variable contructed by appending '0' (zero) to the prefix.

## Return codes

The following return codes are possible:

**0** Normal completion.

**4** List incomplete. There was insufficient space to contruct a valid variable name.

**12** Prefix too long. List-var must be 7 characters or less.

**20** Severe error.

## Example

In a CLIST, report the number of open tables:

```
ISPEXEC QTABOPEN LIST(myvar)
IF &LASTCC = 0 THEN DO
  WRITE THE NUMBER OF TABLES OPEN ARE MYVAR0
```

---

# QUERYENQ – Query System ENQ Data

The QUERYENQ service allows an ISPF dialog to obtain a list of all system enqueues, or all system enqueues that match the specified criteria.

## Command Invocation Format

```
ISPEXEC QUERYENQ   TABLE(table-name)
                   QNAME(qname)
                   RNAME(rname)
                   REQ(pattern)
                   WAIT
                   LIMIT(limit)
                   SAVE(list-id)
                   XSYS
```

## Call Invocation Format

```
CALL ISPLINK ('QUERYENQ'   ,table-name
                           ,qname
                           ,rname
                           ,pattern
                           ,['WAIT  ']
                           ,limit
                           ,list-id
                           ,['XSYS   ']);
OR
CALL ISPEXEC (buf-len, buffer);
```

## Parameters

**table-name**
A table that must not exist before the service is called. It is returned to the user as an open, non-writable table. It is the caller's responsibility to delete the table

*Table 3. Variables Returned in Each Row of the Table (continued)*

| Name | Size | Description |
|---|---|---|
| ZENDISP | 5 | SHARE or EXCLU |
| ZENHOLD | 4 | OWN or WAIT |
| ZENSCOPE | 7 | SYSTEM or SYSTEMS |
| ZENSTEP | 7 | STEP or blank |
| ZENGLOBL | 6 | GLOBAL or blank |
| ZENSYST | 8 | System name |
| ZENRESV | 7 | RESERVE or blank |

## Return codes

The following return codes are possible:

**0** Table returned or data set written, but XSYS parameter was not specified and the system is running in STAR mode. The data returned may not reflect all ENQs on all systems.

**2** Table returned or data set written.

**4** Table returned but truncated due to limit.

**8** No ENQs satisfy the request.

**10** No ENQs satisfy the request, but XSYS parameter was not specified and the system is running in STAR mode. The data returned may not reflect all ENQs on all systems.

**12** Table creation error, parameter or other termination error. See messages for more detail. This includes services not available due to configuration table restrictions.

**14** The SAVE data set is in use by another user.

**20** Severe error, including TBADD error or data set creation errors.

# REMPOP – Remove a Pop-Up Window

The REMPOP service removes the pop-up window created by an ADDPOP service call. After invoking the REMPOP service, any DISPLAY, TBDISPL or SELECT panel service call will either display a panel in the full panel area of the screen or a higher level pop-up window, if one is active.

## Command Invocation Format

```
ISPEXEC REMPOP  [ ALL ]
```

## Call Invocation Format

```
CALL ISPEXEC  (buf-len, buffer);

CALL ISPLINK  ('REMPOP  ' [,'ALL     ']);
```

## Parameters

**ALL**
Indicates that the dialog manager is to remove all pop-up windows that were created at the current select level. If you do not specify ALL, only one pop-up window is removed.

**buf-len**
Specifies a fullword fixed binary integer containing the length of *buffer*.

**buffer**
Specifies a buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC call for a command procedure.

## Return codes

The following return codes are possible:

**0** Normal completion.

**16** A pop-up window does not exist at this select level.

**20** Severe error.

---

# SELECT – Select a Panel or Function

The SELECT service can be used to display a hierarchy of selection panels or invoke a function.

Within a dialog function, a program can invoke another program using standard CALL or LINK conventions. These nested programs are transparent to the dialog manager. However, when the invoked program is a new dialog function, SELECT must be used.

APL2 can be invoked by specifying the APL2 command and its appropriate keywords as the value of the CMD keyword of the SELECT service. In addition, the SELECT keyword and value LANG(APL) should be coded on the SELECT statement if the APL2 function needs to use DM component services. Otherwise, unpredictable results can occur. 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. APL2 limits a user to one active workspace. In split screen mode, if APL2 is active on one screen, it cannot be activated by the SELECT service on the other screen.

## Command Invocation Format

```
ISPEXEC  SELECT { PANEL(panel-name)    [ADDPOP]                }
                {                      [OPT(option)]           }
                {                                              }
                { CMD(command)  [LANG(APL|CREX)]               }
                {               [MODE(LINE|FSCR)]              }
                {               [BARRIER]                      }
                {               [NEST]                         }
                {                                              }
                { PGM(program-name)  [PARM(parameters)]        }
                {                    [MODE(LINE|FSCR)]         }
                { WSCMD(workstation-command)                   }
                {                      [MODAL|MODELESS]        }
                {                      [WSDIR(dir)]            }
                {                      [MAX|MIN]               }
                {                      [VIS|INVIS]             }
```

```
{ WSCMDV(var_name)                              }
{                       [MODAL|MODELESS]        }
{                       [WSDIR(dir)]            }
{                       [MAX|MIN]               }
{                       [VIS|INVIS]             }

 [NEWAPPL[(application-id)] [PASSLIB]]|[NEWPOOL]

 [SUSPEND]

 [SCRNAME(screen_name)]
```

## Call Invocation Format

```
CALL  ISPEXEC (buf-len, buffer);

OR

CALL  ISPLINK ('SELECT  ', length, keywords);
      Note: parameters that can appear in keywords are:

{ PANEL(panel-name)    [ADDPOP]                 }
{                      [OPT(option)]            }
{                                               }
{ CMD(command)  [LANG(APL|CREX)]                }
{               [MODE(LINE|FSCR)]               }
{               [BARRIER]                       }
{               [NEST]                          }
{                                               }
{ PGM(program-name)  [PARM(parameters)]         }
{                    [MODE(LINE|FSCR)]          }
{ WSCMD(workstation-command)                    }
{                      [MODAL|MODELESS]         }
{                      [WSDIR(dir)]             }
{                      [MAX|MIN]                }
{                      [VIS|INVIS]              }
{ WSCMDV(var)                                   }
{                      [MODAL|MODELESS]         }
{                      [WSDIR(dir)]             }
{                      [MAX|MIN]                }
{                      [VIS|INVIS]              }

 [NEWAPPL[(application-id)] [PASSLIB]]|[NEWPOOL]

 [SUSPEND]

 [SCRNAME(screen-name)]
```

## Parameters

**panel-name**
Specifies the name of a selection panel to be displayed.

**option**
Specifies an initial option, which must be a valid option on the menu specified by panel-name. Specifying an option causes direct entry to that option without displaying the menu. The menu is processed in non-display mode, as though the user had entered the option.

**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 CLIST command procedure, any TSO command that is to be invoked as a dialog function, or an APL2 command with appropriate keyword values. If the APL2 workspace is already started, **command** specifies a string to be passed to the APL2 workspace for execution.

CLIST command parameters can be included within the parentheses. You can prefix the CLIST procedure name with a percent sign (%) to:

- Improve performance.
- Prevent ISPF from entering line display mode if you do not specify MODE(FSCR).
- Assure that the CLIST command procedure is invoked if ISPF has access to a program function that has the same name as the CLIST. If you use the percent sign prefix, ISPF searches only for a CLIST with the specified name. However, without the percent sign prefix, ISPF searches first for a program, then for a CLIST procedure.

  TSO commands specified by this parameter are invoked by the ATTACH macro.

**LANG(APL)**
If this is the first LANG(APL) request, this parameter specifies that the command specified by the CMD keyword is to be invoked and an APL2 environment is to be started. If this is not the first request, this parameter specifies that the string specified by the CMD keyword is to be passed to the APL2 workspace and executed. If this is the first LANG(APL) request and a command other than APL2, or equivalent, is specified by the CMD keyword, the result is not predictable.

**LANG(CREX)**
Specify that the command specified in 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 rather than an ISPF module function pool. LANG(CREX) is optional if the compiled REXX has been link-edited to include any of the stubs EAGSTCE, EAGSTCPP, or EAGSTMP.

See *ISPF Dialog Developer's Guide and Reference* for more information about Compiled REXX processing.

**MODE(LINE)**
Specifies that line mode is to be entered when selecting a command procedure or program function. If you do not specify mode when selecting a command procedure, line mode is entered unless you prefix the command with a percent sign (%).

**MODE(FSCR)**
Specifies that line mode is not to be entered when selecting a command, CLIST, or program function.

**BARRIER**
Specifies that no commands from the REXX data stack will be pulled upon completion of a command invoked with the SELECT service.

**NEST**
Specifies that commands invoked with the SELECT service will be nested. This will allow command output trapping and communication through global variables.

**program-name**
Specifies the name of a program that is to be invoked as a dialog function. If the program is coded in PL/I, it must be a MAIN procedure. Dialog developers should avoid the ISP and ISR prefixes (the DM and PDF component codes) in naming dialog functions. Special linkage conventions, intended only for internal ISPF use, are used to invoke programs named "ISPxxxxx" and "ISRxxxxx".

This parameter must specify a name of a load module, load module alias, or an entry point that is accessible by use of the LINK macro.

See the *ISPF Dialog Developer's Guide and Reference* for restrictions that apply to dialogs in various languages.

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

The parameters within the parentheses are passed as a single character string, preceded by a halfword containing the length of the character string, in binary. The length value does not include itself.

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

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

If the value of the PARM field is to be used as an ISPF dialog variable, it must be assigned to a fixed-length character string, because the VDEFINE service cannot handle variable length PL/I strings.

**Note:** If you want to use special characters in your character string you must use a single quotation mark at the beginning and at the end of the string.

Some high-level languages, such as PL/I, have parameter syntax requirements specific to the language. For example, the first character of the PARM field must be a slash ('/'), because PL/I assumes that any value before the slash is a run-time option. Refer to the publications supporting the language for specific requirements.

**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:)`.

**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**
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: embedded blanks, quotation marks, or special characters that might not parse properly with the WSCMD service.

**NEWAPPL**
Specifies that a new application is being invoked.

**application-id**
Specifies a 1- to 4-character code for the new application named in this SELECT service request. The code is to be prefixed to the user's profile, the edit profile, and the command table associated with the application, as follows, where xxxx is the application-id:

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

The names xxxxPROF, xxxxEDIT, and xxxxCMDS represent table (member) names in the profile or table input library.

If the NEWAPPL keyword is specified but the application-id is not specified, the default application-id is ISP, as follows:

```
User Profile     -  ISPPROF
Edit Profile     -  ISPEDIT
Command Table    -  ISPCMDS
```

If the NEWAPPL keyword is not specified, the application-id defaults to the current application-id.

If an application is invoked using SELECT with NEWAPPL and the invoked application has its own command table that is defined to ISPTLIB using LIBDEF, the LIBDEF of ISPTLIB must be done before issuing the SELECT CMD(..) NEWAPPL(..) for the application's command table to be available for use. This is necessary because the command table associated with the APPLID is opened at the time that the SELECT is processed. Failing to do the LIBDEF for ISPTLIB before the SELECT with NEWAPPL will result in the command table which was defined using LIBDEF not being opened and commands not being found. If the application's unique command table is not found, then the ISPF default command table, ISPCMDS, is loaded for that dialog.

The following is an example of how to code a LIBDEF for ddname ISPTLIB with the data set that contains the command table, APPCCMDS, for application APPC.

The application invoking CLIST CCC:

```
      .
      ISPEXEC SELECT CMD(CCC) NEWAPPL(TEMP)
      .

CLIST CCC:
      PROC 0
      ISPEXEC LIBDEF ISPTLIB DATASET ID(....)
      ISPEXEC SELECT CMD(CMDC) NEWAPPL(APPC) PASSLIB
      ISPEXEC LIBDEF ISPTLIB
      EXIT CODE(0)
```

**PASSLIB**

Indicates that the current set of application-level ISPF libraries, if any exist, are to be used by the application being selected. PASSLIB is valid only if NEWAPPL is specified.

The PASSLIB keyword can also be specified when setting the & ZSEL variable in a selection panel or in command table entries containing the SELECT action.

When both NEWAPPL and PASSLIB are specified, the current set of application-level libraries is made available to the selected application. Any changes made to this set of libraries while this application is running are in effect only while this application has control. Once the selected application terminates, the original set of application-level libraries is reactivated.

If LIBDEF has been issued for a user link library when a SELECT specifying NEWAPPL and PASSLIB is issued, the selected program makes available the LIBDEF user link library definition. Any SELECTs subsequently issued by the program employ member search orders dependent upon the LIBDEF user link library definition.

If a SELECT of a program is issued, and a LIBDEF of a user link library has not been made or PASSLIB is not specified, any SELECTs issued by the program rely on the following convention for member search order:

```
JOB PACK AREA
ISPLLIB
STEP LIBRARY
LINK PACK AREA
LINK LIBRARY
```

If NEWAPPL is specified and PASSLIB is not specified, the current set of application-level libraries, if any exist, are not to be used by the application being selected. The deactivation of these libraries takes place *before* the application is selected. The current application-level library definitions are

saved, however, so they can be replaced in the library search sequence when the application being selected terminates.

When NEWAPPL and PASSLIB are not specified, the current set of application-level libraries remains in effect because the selected function does not represent a new application. If the selected function changes any of these library definitions, the changes apply through all select levels of the application of which the selected function is a part.

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

**SUSPEND**
Specifies that all pop-up windows in the logical screen should be temporarily removed from the terminal screen. Panels displayed by the selected dialog will appear in the full logical screen.

The selected dialog can issue ADDPOP and REMPOP services to create its own pop-up windows. A dialog that is invoked with the SUSPEND option cannot display panels in the windows created by the previous dialog.

When the selected dialog ends, any pop-up windows that were removed will be restored.

The terminal screen is not changed at the time of the SELECT service. The pop-up windows are removed or restored at the next panel display.

**screen-name**
Specifies that the logical screen in which the SELECT command is issued will be given the specified ″screen name″. This logical screen will keep the screen name until that select level is exited, then it returns to its previous value. The user may override the screen name assigned with the SCRNAME command.

**length**
Specifies the length of a buffer containing the selection keywords. This parameter must be a fullword fixed binary integer.

**keywords**
Specifies the name of a buffer containing the selection keywords. This is a character string parameter. The selection keywords in the buffer are specified in the same form as they would be coded for the ISPEXEC command. For example:

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

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

**buf-len**
Specifies a fullword fixed binary integer containing the length of “buffer.”

**buffer**
Specifies a buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC call for a command procedure.

If a command or program is invoked by using SELECT, the return code from the command or program is passed to the function that invoked SELECT. If a selected command, not using ISPF display services, could cause a full screen input or

output operation, the developer should refresh the entire screen on the next display. To do this, use the CONTROL DISPLAY REFRESH service. A selected command procedure or program can cause the screen settings to change. ISPF does not check for these changes. It is the user's responsibility to ensure that the screen settings are saved and then restored before returning to ISPF.

The SELECT interface permits parameters to be specified as symbolic variables. Before a scan and syntax check of a statement, variable names and the preceding ampersands are replaced with the value of the corresponding variable. A single scan takes place.

**Note:** If you receive an abend from a SELECT command, a message indicating the abend code is issued. However, the ISPF subtask does not abend. The results of this scenario are the same if you have ISPF TEST mode on or off.

## Return Codes

The following return codes are possible if a panel is specified:

**0** Normal completion. The END command was entered from the selected menu.

**4** Normal completion. The RETURN command was entered or the EXIT option was specified from the selected menu or from some lower-level menu.

**12** The specified panel could not be found.

**16** Truncation error in storing the ZCMD or ZSEL variable.

**20** Severe error.

**Note:** A return code of 0 is returned when the SELECT service has been coded with no other parameters.

The following return codes are possible from a MODAL workstation command:

**20** Parameter or syntax on SELECT service (all SELECTs) is not valid.

**41** A null command was passed to the workstation.

**42** ISPF was not able to start the command at the workstation.

**43** Remote execution of commands was not allowed by the user.

**OTHER**
The return code from the workstation command + 100 if the return code from the workstation command was > 0

## Example 1

In a CLIST, start a hierarchy of selection panels from a dialog function. The first menu in the hierarchy is named QOPTION.

```
ISPEXEC SELECT PANEL(QOPTION)
```

## Example 2

In a PL/I program, start a hierarchy of selection panels from a dialog function. The first menu in the hierarchy is named QOPTION. Set the program variable BUFFER to contain:

```
SELECT PANEL(QOPTION)
```

Set program variable BUFLEN to the length of the variable BUFFER. Issue the following command:

```
CALL ISPEXEC (BUFLEN, BUFFER);
```

### Example 3

In a PL/I program, program variable QOPT contains 'PANEL(QOPTION)' and program variable QOPTL is a fullword variable containing the binary equivalent of 14. Start a hierarchy of selection panels beginning with panel QOPTION.

```
CALL ISPLINK ('SELECT  ',QOPTL,QOPT);
```

### Example 4

In a CLIST, invoke a program-coded dialog function named PROG1, and pass it a parameter string consisting of ABCDEF.

```
ISPEXEC SELECT PGM(PROG1) PARM(ABCDEF)
```

### Example 5

In a PL/I program, invoke a program-coded dialog function named PROG1, and pass it a parameter string consisting of ABCDEF. Set the program variable BUFFER to contain:

```
SELECT PGM(PROG1) PARM(ABCDEF)
```

Set program variable BUFLEN to the length of the variable BUFFER. Issue the following command:

```
CALL ISPEXEC (BUFLEN, BUFFER);
```

### Example 6

In a PL/I program, program variable PROG contains 'PGM(PROG1) PARM(ABCDEF)' and program variable PROGL is a fullword variable containing the binary equivalent of 23. Invoke a program-coded dialog function, named PROG1, and pass it a parameter string consisting of ABCDEF.

```
CALL ISPLINK ('SELECT  ',PROGL,PROG);
```

## SETMSG – Set Next Message

The SETMSG service allows a dialog function to display a message on the next panel that is written by ISPF to the terminal. The next panel does not have to be displayed as a result of action taken by the function routine. In fact, the function routine can have terminated before the next panel is displayed.

The specified message is retrieved from the message library at the time the set message request is issued. Values for all variables defined in the message are substituted at this time and the message is saved in a message area for the application. When the next panel is displayed, the message is retrieved from the save area and displayed on the panel.

If multiple set-message requests have been issued before a panel is displayed, only the last message is displayed. You can use the optional COND parameter to request that the specified message is to be displayed only if there is no prior SETMSG request pending. A message specified on the panel display request is overridden by any outstanding set next message request.

A message that has been set with SETMSG is displayed the next time any full-screen output is sent to the display, regardless of whether that output is a

panel, table display, Browse data, or Edit data. The SETMSG service executed in the batch environment causes the message to be written to the log at the point at which it would normally be sent to the screen for display.

The message is preserved across CONTROL NONDISPL; that is, the message is displayed on the next actual output to the terminal. If the next panel is processed in non-display mode, the message remains pending, to be displayed with any following panel that is processed in display mode.

If the message refers to a help panel, the help panel should not include substitutable variables. Variables in related help panels contain the values current at the time the HELP command is issued, not at the time the SETMSG service is invoked.

## Command Invocation Format

```
ISPEXEC  SETMSG  MSG(message-id) [COND]
                 [MSGLOC(message-field-name)]
```

## Call Invocation Format

```
CALL  ISPEXEC (buf-len, buffer);

OR

CALL  ISPLINK ('SETMSG  ', message-id [,'COND    ']
                                      [,message-field-name])
```

## Parameters

**message-id**
Specifies the identification of the message to be displayed on the next panel.

**COND**
Specifies that the message is to be displayed on the next panel only if no prior SETMSG request is pending.

**message-field-name**
Used to position the message pop-up window. If the application specifies this parameter, the Dialog Manager positions the message pop-up relative to the named field.

If this parameter is omitted and a message is displayed in a message pop-up window, the window is displayed at the bottom of the logical screen or below the active ADDPOP pop-up window if one exists.

For compatibility with later versions, this parameter should be specified only when the message will display in a pop-up window.

**Note:** When running in GUI mode, this parameter is ignored.

**buf-len**
Specifies a fullword fixed binary integer containing the length of "buffer."

**buffer**
Specifies a buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC call for a command procedure.

## Return Codes

The following return codes are possible:

**0** Normal completion.

**4** SETMSG with COND parameter issued and a SETMSG request was pending.

**12** The specified message field name or message not be found.

**20** Severe error.

## Example 1

On the next panel that is displayed, put a message whose ID, ABCX015, is in a dialog variable named TERMSG.

```
ISPEXEC SETMSG MSG(&TERMSG )
```

Set the program variable BUFFER to contain:

```
SETMSG MSG(ABCX015 )
```

Set program variable BUFLEN to the length of the variable BUFFER. Issue the following:

```
CALL ISPEXEC (BUFLEN, BUFFER);
```

or alternately

```
CALL ISPLINK ('SETMSG  ','ABCX015 ');
```

## Example 2

The following SETMSG and DISPLAY request displays message TSTA110 in a message pop-up window that requires a response from the end user before interaction with the underlying panel is possible. The message pop-up window is positioned relative to the field FLD1.

```
PROC 0
ISPEXEC SETMSG MSG(TSTA110) MSGLOC(FLD1)
ISPEXEC DISPLAY PANEL(A)
```

Using the following message definition for TSTA110

```
TSTA110              .WINDOW=RESP
'ENTER NUMERIC DATA'
```

Results in:

## Return Codes

The following return codes are possible:

**0** Normal completion.

**4** One or more commands in the stack could not be found in the active set of command tables.

**8** User-requested termination using the END or RETURN command.

**20** Severe error.

# TBADD – Add a Row to a Table

The TBADD service adds a new row of variables to a table. The new row is added either immediately following the current row, pointed to by the current row pointer (CRP), or is added at a point appropriate for maintaining the table in the sequence specified in a previously processed TBSORT request. The CRP is set to point to the newly inserted row.

The current contents of all dialog variables that correspond to columns in the table, which were specified by the KEYS and NAMES parameters in a TBCREATE, are saved in the row.

Additional variables, those not specified when the table was created, can also be saved in the row. These “extension” variables apply only to this row, not the entire table. The next time the row is updated, the extension variables must be specified again if they are to be rewritten.

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

For tables without keys, no duplicate checking is performed.

To improve performance when you add several rows to a table, you can specify the MULT keyword with the *number-of-rows* parameter. By specifying the estimated number of rows you expect to add to the table, you supply ISPF the information it needs to more efficiently obtain the necessary storage for all rows when processing

the first of these rows (rather than getting storage for one row at a time). The default value for the number-of-rows parameter is one unless the value is modified at ISPF installation.

When successive TBADD service requests with the MULT keyword are executed in a program loop, the first request results in storage being acquired for the multiple number of rows specified. On subsequent TBADD requests in the loop, ISPF checks to see if enough storage remains for the current row being added. If so, ISPF acquires no additional storage. If not, ISPF acquires additional storage as specified by the MULT keyword.

If the first row to be added to the table includes one or more extension variables, ISPF assumes that all rows added by the TBADD service request *might* include extension variables and takes that into account when obtaining the storage for the rows to be added.

If ISPF is unable to obtain all the storage it has estimated is needed for the number of rows specified (or if not specified, the default number of rows), it gets storage for one row at a time and issues a return code of four. ISPF does not issue an informational message when this condition occurs. At any time, if there remain rows to be added to the table and ISPF is unable to get storage for one row, a severe error (return code 20) results.

## Command Invocation Format

```
ISPEXEC  TBADD  table-name

                [SAVE(name-list)]

                [ORDER]

                [MULT(number-of-rows)]
```

## Call Invocation Format

```
CALL  ISPEXEC (buf-len, buffer);

OR

CALL  ISPLINK ('TBADD   ', table-name

                           [,name-list]

                           [,'ORDER   ']

                           [,number-of-rows]);
```

## Parameters

**table-name**
Specifies the name of the table to be updated.

**name-list**
Specifies a list of extension variables, by name, that are to be saved in the row, in addition to the variables specified when the table was created.

**ORDER**
Specifies that the new row is to be added to the table in the order specified in the sort information record. A TBSORT must have been performed for this

table before use of this keyword. For tables with keys, the table is searched to ensure that the new row has a unique key. If a row with the same key already exists, the row is not added. This keyword is ignored if the table has never been sorted. If this keyword is omitted, any existing sort information record is nullified and to restore it, another TBSORT is required.

When a newly inserted row has sort field names equal to the sort field names of an existing row, the insertion is made after the existing row.

**number-of-rows**
Specifies the expected total number of rows to be added to a table during one session. This is a fullword fixed value greater than zero. The default value is one unless changed at ISPF installation. The maximum value that can be specified is 32 767.

**buf-len**
Specifies a fullword fixed binary integer containing the length of "buffer."

**buffer**
Specifies a buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC call for a command procedure.

## Return Codes

The following return codes are possible:

**0** Normal completion.

**4** The number-of-rows parameter was specified but storage was obtained for only a single row.

**8** A row with the same key already exists; CRP set to TOP (zero). Returned only for tables with keys.

**12** Table is not open.

**16** Numeric convert error; see numeric restrictions for TBSORT. Returned only for sorted tables.

**20** Severe error.

## Example 1

Add a row to the table TELBOOK, based on the sort information record, copying to the row values from function pool variables whose names match those of table variables.

```
ISPEXEC TBADD TELBOOK ORDER
```

Set the program variable BUFFER to contain:

```
TBADD TELBOOK ORDER
```

Set program variable BUFLEN to the length of the variable BUFFER. Issue the following:

```
CALL ISPEXEC (BUFLEN, BUFFER);
```

or alternately

```
CALL ISPLINK ('TBADD   ','TELBOOK ',' ','ORDER   ');
```

## Example 2

Add multiple rows to table TELBOOK.

```
ISPEXEC TBADD TELBOOK MULT(&ROWS)
```

where &ROWS is a variable containing the number of rows to be added.

```
ISPEXEC TBADD TELBOOK ORDER MULT(4)
```

where 4 is the number of rows to be added

```
CALL ISPLINK ('TBADD   ','TELBOOK ',' ','ORDER   ',ROWS);
```

where ROWS is a fixed binary variable containing the number of rows to be added.

```
CALL ISPLINK ('TBADD   ','TELBOOK ',' ',' ',8);
```

where 8 indicates the number of rows to be added.

# TBBOTTOM – Set the Row Pointer to Bottom

The TBBOTTOM service sets the current row pointer (CRP) to the last row of a table and retrieves the row unless the NOREAD parameter is specified.

If NOREAD is not specified, all variables in the row, including key, name, and extension variables, if any, are stored in the corresponding dialog variables. A list of extension variable names can also be retrieved.

## Command Invocation Format

```
ISPEXEC  TBBOTTOM  table-name  [SAVENAME(var-name)]
                               [ROWID(rowid-name)]
                               [NOREAD]
                               [POSITION(crp-name)]
```

## Call Invocation Format

```
CALL  ISPEXEC (buf-len, buffer);

OR

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

## Parameters

**table-name**
Specifies the name of the table to be used.

**var-name**
Specifies the name of a variable where a list of extension variable names

contained in the row will be stored. The list is enclosed in parentheses, and the names within the list are separated by a blank.

**rowid-name**
Specifies the name of a variable in which a number that uniquely identifies the row being accessed is to be stored. Later, this identifier can be specified in the ROW parameter of TBSKIP to cause the CRP to be positioned to the row. This identifier is not saved on permanent storage by TBSAVE or TBCLOSE.

**NOREAD**
Specifies that the variables contained in the requested row are not to be read into the variable pool.

**crp-name**
Specifies the name of a variable in which the row number pointed to by the CRP is to be stored. If the CRP is positioned to TOP, the row number returned is zero.

**buf-len**
Specifies a fullword fixed binary integer containing the length of "buffer."

**buffer**
Specifies a buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC call for a command procedure.

## Return Codes

The following return codes are possible:

**0** Normal completion.

**8** Table is empty; CRP set to TOP (zero).

**12** Table is not open.

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

**20** Severe error.

## Example

Move the current row pointer (CRP) of the table TELBOOK to the last row of the table. From this row, store variable values into the respective function pool variables having the same names.

```
ISPEXEC TBBOTTOM TELBOOK
```

Set the program variable BUFFER to contain:

```
TBBOTTOM TELBOOK
```

Set program variable BUFLEN to the length of the variable BUFFER. Issue the following:

```
CALL ISPEXEC (BUFLEN, BUFFER);
```

or alternately

```
CALL ISPLINK ('TBBOTTOM','TELBOOK ');
```

# TBCLOSE – Close and Save a Table

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

If the table was opened in WRITE mode, TBCLOSE copies the table from virtual storage to the table output library. In this case, the table output library must be allocated to a ddname of ISPTABL or defined by a LIBDEF service request before invoking this service. When storing a table in an output library, the user can give it a new name. The table name used in the output library must not be an alias name.

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

Table output can be directed to a table output library other than the library specified on the table output ISPTABL DD statement. The library to be used must be allocated before table services receives control. Thus, an application can update a specific table library. This is particularly useful for applications that need to maintain a common set of tables containing their data.

The output table library, specified by the ISPTABL DD statement, is the default output library. Therefore, dialogs previously written for SPF that use table services continue to function in the same manner in ISPF as they did when running under SPF.

A TBCLOSE request for a shared table causes the use count in the table for that logical screen to be decremented by one. If the use count for all logical screens is zero, the TBCLOSE service is performed. If the count is not zero, a TBSAVE service is performed. This leaves the table available for continued processing in any screen that still has a use count greater than zero.

Issuing a TBCLOSE with the LIBRARY parameter for a table is not related to closing the data set allocated to that ddname. However, if the LIBDEF service with the DATASET keyword is used to define the alternate library, the data set may be closed and freed by deleting the corresponding LIBDEF specification.

## Command Invocation Format

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

## Call Invocation Format

```
CALL  ISPEXEC (buf-len, buffer);

OR

CALL  ISPLINK ('TBCLOSE ', table-name [,'NEWCOPY '|'REPLCOPY']
                                      [,alt-name]
                                      [,percentage]
                                      [,library][reserved position];
```

## Parameters

**table-name**
Specifies the name of the table to be closed.

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

**REPLCOPY**
Specifies that the table is to be rewritten in place in the output library. If the existing member is smaller than the table that replaces it, or if a member of the same name does not exist in the library, the complete table is written at the end of the output library.

A comparison is made between the virtual storage size of the table and the external size in the table output library. If there is insufficient storage to write the table in place, it is written at the end of the table output library.

**alt-name**
Specifies an alternate name for the table. The table is stored in the output library with the alternate name. If another table already exists in the output library with that name, it is replaced. If the table being saved exists in the output library with the original name, that copy remains unchanged.

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

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

The default value for this parameter is zero.

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

**library**
Specifies the name of a DD statement or LIBDEF lib-type that defines the output library in which the table is to be closed. If specified, a generic (non-ISPF) DD name must be used. If this parameter is omitted, the default is ISPTABL.

**buf-len**
Specifies a fullword fixed binary integer containing the length of "buffer."

**buffer**
Specifies a buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC call for a command procedure.

## Return Codes

The following return codes are possible:

**0** Normal completion.

**12** Table is not open.

**16** Alternate table output library was not allocated.

**20** Severe error.

## Example

Close the table TELBOOK.

```
ISPEXEC TBCLOSE TELBOOK
```

Set the program variable BUFFER to contain:

```
TBCLOSE TELBOOK
```

Set program variable BUFLEN to the length of the variable BUFFER. Issue the following command:

```
CALL ISPEXEC (BUFLEN, BUFFER);
```

or alternately

```
CALL ISPLINK ('TBCLOSE ','TELBOOK ');
```

# TBCREATE – Create a New Table

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

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

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

## Command Invocation Format

```
ISPEXEC  TBCREATE  table-name  [KEYS(key-name-list)]
                                [NAMES(name-list)]
                                [WRITE|NOWRITE]
                                [REPLACE]
                                [LIBRARY(library)]
                                [SHARE]
```

## Call Invocation Format

```
CALL  ISPEXEC (buf-len, buffer);

OR

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

```
                                        [,'REPLACE ']

                                        [,library]

                                        [,'SHARE   ']);
```

## Parameters

**table-name**
Specifies the name of the table to be created. The name can be from one to eight alphanumeric characters in length and should begin with an alphabetic character.

**key-name-list**
Specifies the variables, by name, that are to be used as keys for accessing the table. See "name-list" on page 6 for the specification of name lists. If this parameter is omitted, the table will not be accessible by keys.

**name-list**
Specifies the non-key variables, by name, to be stored in each row of the table.

If key-name-list and name-list are omitted, the table can contain only extension variables that must be specified when a row is written to the table.

**WRITE**
Specifies that the table is permanent, to be written to disk by the TBSAVE or TBCLOSE service. The disk copy is not actually created until the TBSAVE or TBCLOSE service is invoked.

The WRITE/NOWRITE usage of a shared table must be consistent on all TBCREATE and TBOPEN requests. That is, all requests for a given shared table that result in concurrent use of that table must specify the same WRITE or NOWRITE attribute.

**NOWRITE**
Specifies that the table is for temporary use only. When processing is complete, a temporary table should be deleted by the TBEND or TBCLOSE service.

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

A table currently existing in virtual storage in shared mode cannot be replaced. If this is attempted, a return code of 8 results. Further, a shared table cannot be replaced by a non-shared table, and vice versa.

**library**
Specifies the name of a DD statement or LIBDEF lib-type that defines the input library. If specified, a generic (non-ISPF) DD name must be used. If this parameter is omitted, the default input library name is ISPTLIB.

**SHARE**
Specifies that the created table can be shared between all logical screens while the user is in split-screen mode. A table can be "created" by one screen only.

That is, once one screen has issued a TBCREATE SHARE for a given table, another screen is not permitted to issue a TBCREATE for the same table.

A successful TBCREATE or TBOPEN request causes the use count in the table to be incremented by one. The use count determines the action taken by subsequent TBEND and TBCLOSE requests.

**buf-len**
Specifies a fullword fixed binary integer containing the length of "buffer."

**buffer**
Specifies a buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC call for a command procedure.

## Return Codes

The following return codes are possible:

**0** Normal completion.

**4** Normal completion — a duplicate table exists but REPLACE was specified.

**8** Either the table already exists and REPLACE was not specified, or REPLACE was specified and the table is in SHARE mode.

**12** Table in use; ENQ failed.

**16** WRITE mode specified and alternate table input library not allocated. TBCREATE checks the input library to determine if a duplicate table exists. See return code 8.

**20** Severe error.

## Example 1

In a CLIST, create a permanent table, TELBOOK, to contain the variable TABKEY and other variables, the names of which are specified in **dialog** variable TABVARS. The key field is TABKEY.

```
ISPEXEC TBCREATE TELBOOK KEYS(TABKEY) NAMES(&TABVARS )
```

## Example 2

In a PL/I program, create a permanent table, TELBOOK, to contain the variable TABKEY and other variables, the names of which are specified in **program** variable TABVARS. The variable TABVARS has been made accessible to ISPF by a previous VDEFINE operation. The key field is TABKEY. Set the program variable BUFFER to contain:

```
TBCREATE TELBOOK KEYS(TABKEY) NAMES(&TABVARS)
```

Set program variable BUFLEN to the length of the variable BUFFER. Issue the following command:

```
CALL ISPEXEC (BUFLEN, BUFFER);
```

or alternately

```
CALL ISPLINK ('TBCREATE','TELBOOK ','TABKEY  ',TABVARS);
```

## Example 3

In a PL/I program, create a permanent non-keyed table, NKTBL, where FNAME, LNAME, PHONE, and LOC are the non-key table variables.

```
CALL ISPLINK ('TBCREATE','NKTBL ',' ',
            '(FNAME LNAME PHONE LOC)');
```

# TBDELETE – Delete a Row from a Table

The TBDELETE service deletes a row from a table.

For tables with keys, the table is searched for the row to be deleted. The current contents of the key variables, dialog variables that correspond to keys in the table, are used as the search argument. If the table has no keys, the row is determined by the current position of the CRP.

For tables without keys, the row pointed to by the current row pointer (CRP) is deleted.

The CRP is always updated to point to the row before the one that was deleted.

## Command Invocation Format

```
ISPEXEC  TBDELETE  table-name
```

## Call Invocation Format

```
CALL  ISPEXEC (buf-len, buffer);
```

*OR*

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

## Parameters

**table-name**
Specifies the name of the table from which the row is to be deleted.

**buf-len**
Specifies a fullword fixed binary integer containing the length of "buffer."

**buffer**
Specifies a buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC call for a command procedure.

## Return Codes

The following return codes are possible:

**0** Normal completion.

**8** Keyed tables: The row specified by the value in key variables does not exist; CRP set to TOP (zero). Non-keyed tables: CRP was at TOP (zero) and remains at TOP.

**12** Table is not open.

**20** Severe error.

## Example

Delete a row of the table TELBOOK.

```
ISPEXEC TBDELETE TELBOOK
```

Set the program variable BUFFER to contain:

```
TBDELETE TELBOOK
```

Set program variable BUFLEN to the length of the variable BUFFER. Issue the following:

```
CALL ISPEXEC (BUFLEN, BUFFER);
```

or alternately

```
CALL ISPLINK ('TBDELETE','TELBOOK ');
```

## TBDISPL – Display Table Information

The TBDISPL service combines information from a panel definition with information stored in an ISPF table. It displays all or certain rows from the table, allowing the application user to scroll the information up and down.

When only certain rows from a table are to be displayed, the TBSARG service is used to define the selection criteria before issuing TBDISPL. Only search arguments established by TBSARG that specify a forward scan through the table (for example, TBSARG specifying the keyword NEXT, either explicitly or implicitly) should be used. In this case, ROWS(SCAN) must be specified on the )MODEL statement in the panel definition.

TBDISPL can produce a display of a table based on a search argument that specifies a backward scan; that is, PREVIOUS on the TBSARG request and ROWS(SCAN) specified on the )MODEL header statement. This would display the table from bottom to top. Top to bottom is the normal table display. However, because TBDISPL does not support scrolling for the bottom-to-top case, scrolling results are unpredictable.

The format of the display is specified by a panel definition, which TBDISPL reads from the panel library. The panel definition specifies the fixed (non-scrollable) portion and the scrollable portion of the display. The fixed portion contains the command field and commonly the scroll amount field. It can also include other input fields as well as text, output fields, dynamic areas and a graphic area.

The scrollable portion is defined by up to eight “model” lines. They indicate which table fields are to be displayed.

Each line of scrollable data can have one or more input (unprotected) fields, as well as text and output (protected) fields. The user can modify the input fields in the scrollable or fixed portions.

Before TBDISPL is invoked, the table to be displayed must be open, such as TBOPEN, and the current row pointer (CRP) positioned to the row at which the display is to begin, such as TBTOP (automatic following TBOPEN), TBBOTTOM, or TBSKIP. When CRP is pointing to the top of the table, it has a value of 0. It is treated as though the CRP were pointing to the first row. Do not attempt to use TBDISPL to display a command table currently in use. This might produce unpredictable results.

The scrollable portion of the display is formed by replicating the model lines from the panel definition enough times to fill the screen. Each of these replications is known as a model set. Table rows are then read to fill in the appropriate fields in the model set replications. Each table row corresponds to a model set.

The table that is displayed in a panel's scrollable area can be built dynamically by the application. This is useful for applications involving large amounts of data that users might wish to access to varying extents. The application can provide a relatively small table as a starter, then expand the table as users scroll beyond the top or bottom table row.

When the user enters data into a model set, the corresponding table row is said to be selected for processing. The user can select several rows. The data must be modified to select the model set. If you simply overtype the existing model set with the same data, the model set is not considered to be selected.

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

## TBDISPL Operation

TBDISPL allows the user to scroll the data up and down and enter information in the input fields in the scrollable or fixed portions.

TBDISPL operation depends on whether a )REINIT or )PROC section is included in the panel definition. When a )REINIT or )PROC section is included, and if the user makes no modification to the screen and presses the Enter key, TBDISPL returns control to the dialog function. On the other hand, if neither a )REINIT nor a )PROC section is included and if the user makes no modification to the screen and presses the Enter key, TBDISPL treats this as a "no operation" and control does not return to the dialog function. This is for compatibility with the previous version of the product.

During a display of a panel using TBDISPL, any of the following user actions will result in control returning to the dialog function:

- Typing no input and pressing the Enter key, assuming that a )REINIT or )PROC section exists in the panel definition
- Typing data into the fixed or scrollable portion of the display and pressing the Enter key
- Typing data into the fixed or scrollable portion of the display and entering the UP or DOWN command
- Entering the END or RETURN command
- Scrolling UP or DOWN with scroll return to function defined and not enough table rows to handle the scroll request.

## Operational Results From User Actions

The following user actions will *not* result in control returning to the dialog function:

- Typing no input and pressing the Enter key (assuming that neither a )REINIT nor a )PROC section exists in the panel definition).
- Typing no input and entering the UP or DOWN command. This is true if scroll return to function is not defined, but there are enough rows to satisfy the scroll request.
- Entering a system command other than UP, DOWN, END, or RETURN. For example, HELP, SPLIT, or CURSOR.
- Entering an application command that SELECTs another dialog.

After display of a panel using TBDISPL, and before control returns to the dialog function, the following occurs:

1. The contents of all input fields in the fixed portion are stored in the dialog variable specified in the panel definition.
2. If there were no selected rows to process, the CRP is set to TOP (zero). If scroll return to function is defined and rows are needed to satisfy the scroll request, the scroll return system variables are set in the function pool.
3. If there were any selected rows, the CRP is positioned to the first of these, and the row is retrieved from the table. The values of all variables from that row are stored into the corresponding dialog variables. All input fields in the selected model set on the display are then stored in the corresponding dialog variables. The input fields can or cannot correspond to variables in the table.

   Variable ZTDSELS contains the number of rows that were selected. The value of ZTDSELS can be checked in the )PROC section of the panel definition, or it can be checked by the dialog function.
4. The row number that corresponds to the first model set currently displayed on the screen is stored in the system variable ZTDTOP. If, in a dialog, you want to reposition the scrollable data as the user last saw it, you *must* reposition the CRP to the row number stored in ZTDTOP before reinvoking the TBDISPL service with the panel name specified. This is not necessary if the panel name is not specified.

### ZTDTOP and ZTDSELS Variables

ZTDTOP and ZTDSELS are variables in the function pool. A command procedure can access them directly. A program can access them through use of the VDEFINE or VCOPY service. If a program function uses the VCOPY service to access the variable, the value will be in character string format. It will not be in fixed binary format.

If the application user selected more than one row in a single interaction, the variable ZTDSELS is 2 or greater, which indicates that selected rows remain to be processed. These rows are called pending selected rows. A call to TBDISPL is required to position the CRP to each pending selected row, retrieve the row from the table, and store input fields from the corresponding model set. After the CRP is positioned to each selected row, the function can process the row, for example, by issuing a TBPUT request to update the table. For these calls, neither the panel-name nor the message-id should be specified. The processing sequence for each of these calls is as described above, except that the next selected row is processed.

Whenever selected rows remain to be processed, the dialog can choose to ignore them by calling TBDISPL with a specified (nonblank) panel name. This clears out any remaining information about previous calls. If the dialog wants to display another screen before processing pending selected rows from the first display, it must invoke the CONTROL service to save and restore the display environment.

**Note:** Table display service system variables, ZTD*, are not saved as part of the CONTROL DISPLAY SAVE/RESTORE information. The values of these variables may be saved by the dialog developer and restored before resuming processing of the initial table display.

## Command Invocation Format

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

```
                                    [CURSOR(field-name)]
                                    [CSRROW(table-row-number)]
                                    [CSRPOS(cursor-position)]
                                    [AUTOSEL(YES|NO)]
                                    [POSITION(crp-name)]
                                    [ROWID(rowid-name)]
                                    [MSGLOC(message-field-name)]
```

## Call Invocation Format

```
CALL  ISPEXEC (buf-len, buffer);

OR

CALL  ISPLINK ('TBDISPL ', table-name [,panel-name]
                                      [,message-id]
                                      [,field-name]
                                      [,table-row-number]
                                      [,cursor-position]
                                      ['YES     '|'NO      ']
                                      [,crp-name]
                                      [,rowid-name]
                                      [,message-field-name]);
```

## Parameters

**table-name**
Specifies the name of the table to be displayed.

**panel-name**
Specifies the name of the panel to be displayed.

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

**field-name**
Specifies the name of the field where the cursor is to be placed on the display. Any setting of the .CURSOR control variable done in the panel definition takes precedence over this parameter.

**table-row-number**
Specifies the table row number (CRP number) corresponding to the model set on the display where the cursor is to be placed. For a call, this parameter must be a fullword fixed binary number.

Specifying the CSRROW parameter without specifying AUTOSEL(NO) results in the row being retrieved, even if the user did not explicitly select the row. This is called auto-selection.

If the specified row does not have a corresponding model set in the logical table display (the logical table display includes model sets not displayed because of split-screen, PFSHOW, or floating command line), the cursor is placed at the command field. No auto-selection is performed.

Any setting of the .CSRROW control variable done in the panel definition takes precedence over this parameter.

**cursor-position**
Specifies the position within the field where the cursor is to be placed. This position applies regardless of whether the initial cursor placement was specified in the CURSOR calling sequence parameter, the .CURSOR control variable in the )INIT or )REINIT section of the panel, or is the result of default cursor placement. If cursor-position is not specified or is not within the field, the default is 1.

Any setting of the .CSRPOS control variable done in the panel definition takes precedence over this parameter.

**AUTOSEL( YES | NO)**
YES specifies that if the CSRROW(table-row-number) parameter is specified or if .CSRROW is set within the )INIT or )REINIT section, the row is to be retrieved, even if the user did not explicitly select the row. This is known as auto-selection.

NO specifies that even if the CSRROW(table-row-number) parameter is specified or if .CSRROW is set within the )INIT or )REINIT section, the row is to be retrieved only if the user explicitly selects the row by entering data into the corresponding model set.

If the CSRROW parameter or the .CSRROW control variable is not specified, the AUTOSEL parameter is ignored.

Any setting of the .AUTOSEL control variable done in the panel definition takes precedence over this parameter.

**crp-name**
Specifies the name of a variable in which the row number pointed to by the CRP is to be stored. If the CRP is positioned to TOP, the row number returned is zero.

**rowid-name**
Specifies the name of a variable in which a number that uniquely identifies the row being accessed is to be stored. Later, this identifier can be specified in the ROW parameter of TBSKIP to cause the CRP to be positioned to the row. This identifier is not saved on permanent storage by TBSAVE or TBCLOSE.

**message-field-name**
Used to position the message pop-up window. If the application specifies this parameter, the Dialog Manager positions the message pop-up relative to the named field.

If this parameter is omitted and a message is displayed in a message pop-up window, the window is displayed at the bottom of the logical screen or below the active ADDPOP pop-up window if one exists.

For compatibility with later versions, this parameter should be specified only when the message will display in a pop-up window.

**Note:** When running in GUI mode, this parameter is ignored.

**buf-len**
Specifies a fullword fixed binary integer containing the length of "buffer."

**buffer**
Specifies a buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC call for a command procedure.

## Parameter Processing

The panel-name and message-id parameters are optional. Their processing is as follows:

- If panel-name is specified and message-id is not specified, the panel definition is retrieved, rows from the table are read, starting at the CRP, to fill the screen, and the screen is displayed without a message. Any information from previous TBDISPL calls, such as pending scroll requests or pending selected rows, is cleared.
- If panel-name and message-id are both specified, the panel definition is retrieved, rows from the table are read to fill the screen, and the screen is displayed with the specified message.
- If panel-name is not specified and message-id is specified, the current table display is overlaid with a message, without rebuilding the screen or rereading the table.
- If neither panel-name nor message-id is specified, the processing depends on whether there are selected rows remaining to be processed. If no selected rows remain to be processed, the following occurs: If the application user's last action was to:
  - Press the Enter key, then rows from the table are again read to fill the screen and the screen is redisplayed.
  - Enter a scroll command, then the scroll function is now honored by reading and displaying the appropriate rows from the table.
  - Enter an END or RETURN command, then the CRP is set to TOP (zero) and control returns to the function issuing the TBDISPL with a return code of 8. If this occurs more than once in immediate succession, a return code of 20 is issued, since the application can be in a loop.

  If there are selected rows remaining to be processed, the CRP is positioned to the first of these, the row is retrieved from the table, and input fields from the selected model set are stored.

Use the CONTROL service to save and restore the environment when a TBDISPL series, in which panel-name is not specified, is to be interrupted by another TBDISPL, DISPLAY, BROWSE, or EDIT operation.

The CURSOR and CSRROW parameters are optional. Their processing is as follows:

- If the CURSOR parameter is not specified but the CSRROW parameter is specified, the cursor is placed on the first field in the specified row.
- If the CURSOR parameter is specified, but the CSRROW parameter is not specified or is specified with a value of zero, the current value of the CRP determines the row location, and the cursor is placed in this row on the field specified by the CURSOR parameter. A value of zero in the CRP places the cursor on the command line.
- If neither the CURSOR nor the CSRROW parameter is specified, the cursor is placed at the command field.

- If both the CURSOR and CSRROW parameters are specified, the cursor is placed at the field specified by the CURSOR parameter within the model set corresponding to the table row specified by the CSRROW parameter.
- Whenever the CSRROW parameter is specified without specifying AUTOSEL(NO), the row is retrieved, even if the user did not modify that row. This allows the dialog developer to force the user to correct an error on that row before going on to process other rows.
- Any setting of the .CURSOR and the .CSRROW control variables done in the panel definition takes precedence over the CURSOR and CSRROW parameters.

## Return Codes

The following return codes are possible:

**0** If the panel definition contains neither a )REINIT nor a )PROC section, the Enter key was pressed, or a scroll command was entered. Any of the following occurred:

- One row was selected in the scrollable part of the display. The CRP is set to point to that table row and the row is retrieved. The input fields from the selected model set on the display are then stored in the function pool.
- The user entered information into the fixed portion of the display.
- All of the following:
  - A scroll return to function has been specified (ZTDRET defined to UP, DOWN, or VERTICAL).
  - More rows are needed to fill a scroll request.
  - No selected rows remain to be processed.

If the panel definition contains a )REINIT or )PROC section, there is the additional possibility that the user entered no information and just pressed the Enter key.

**4** The Enter key was pressed or a scroll command was entered. The first or both of the following occurred:

- Two or more rows in the scrollable part of the display were selected. The CRP is set to the first selected row and the row is retrieved. The input fields from the selected model set on the display are then stored in the function pool.
- The user entered information into the fixed portion of the display.
- If scroll return to function has been specified, and two or more rows are selected for processing, TBDISPL returns a return code 4 until all selected rows are processed. You process the request for more rows to be added to the table only after all selected rows have been processed; that is, only when ZTDSELS has a value of 0.

For subsequent TBDISPL requests with no panel name and no message-id, return code 4 is issued for each request until one selected row remains to be accessed. For this last row, a return code of zero is issued by TBDISPL, still specified with no panel name and no message-id. The variable ZTDSELS will have a value of one.

**8** The END or RETURN command was entered. For panels created by the conversion utility, CANCEL and EXIT commands also give return code 8. If CANCEL and EXIT is requested from a panel displayed using TBDSPL service calls and the panel was defined with dialog tag language (DTL),

the dialog manager returns the command in ZVERB and sets a return code of 8 from the display screen. The CRP is set to the first of any selected rows in the scrollable part of the display. The input fields from the selected model set on the display are then stored in the function pool.

If no rows were selected, the CRP is at the top (zero).

To process all selected rows when END or RETURN was entered, continue to issue TBDISPL requests with no panel name or message-id specified until ZTDSELS is one.

If you enter the END command on a table display panel, a subsequent redisplay will result in a return code of 8.

The user might have entered information into the fixed portion of the display.

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

**16** Truncation or translation error in storing defined variables.

**20** Severe error.

## Example

Display the table TELBOOK using panel definition TPANEL2 to format the display.

```
ISPEXEC TBDISPL TELBOOK PANEL(TPANEL2)
```

Set the program variable BUFFER to contain:

```
TBDISPL TELBOOK PANEL(TPANEL2)
```

Set program variable BUFLEN to the length of the variable BUFFER. Issue the following:

```
CALL ISPEXEC (BUFLEN, BUFFER);
```

or alternately

```
CALL ISPLINK ('TBDISPL ','TELBOOK ','TPANEL2 ');
```

## System Variables Related to TBDISPL

If a program function uses the VCOPY service to access a variable, the value will be in character string format. It will not be in fixed binary format.

System variables used with TBDISPL processing are:

**ZTDMARK** Specifies an alternate bottom-of-data marker. ZTDMARK is created by the dialog and can reside in any variable pool. It is an input variable, whose length can be equal to or less than the screen width. If ZTDMARK exists, its value is used as the marker. If ZTDMARK does not exist, the default marker of “BOTTOM OF DATA” with asterisks on each side is used.

For example, the following assignment could be made in the )INIT section of a table display panel:

```
&ZTDMARK = '----> End of Data <----'
```

ZTDMARK can be blank. That is, an assignment such as

```
&ZTDMARK = ' '
```

is valid. In this case a bottom-of-data marker would not appear on the screen.

**ZTDMSG** Specifies the ID of a message to be used as an alternate top-row-displayed indicator. ZTDMSG is created by the dialog and can reside in any variable pool. It is an input variable whose length is 8.

If ZTDMSG exists, TBDISPL invokes the GETMSG service to get the short message and long message text. If the short message exists and is nonblank, it is used as the top-row-displayed indicator. If the short message does not exist, the long message text is used as the top-row-displayed indicator. In both cases, the current values of any variables in the message are placed in the message and the text is placed right-justified on the top line of the display.

If ZTDMSG does not exist, the long form of message ISPZZ100 is used.

The text used for the top-row-displayed indicator is summarized in the *ISPF Dialog Developer's Guide and Reference*.

A message ID whose short and long message text is blank (' ') or the null variable (&Z) can be assigned to ZTDMSG. In this case, the table display would not have a top-row-displayed indicator.

No top row is displayed if the user attempts to:

- Display an empty table
- Scroll past the bottom row
- Specify no rows matching the TBSARG criteria and ROWS(SCAN) is specified on the )MODEL statement of the panel definition.

In this case, message ISPZZ101 is used for the top-row-displayed indicator. This has no short message text, and the long message text is '&Z'.

**ZTDROWS** Created by TBDISPL to indicate the number of rows in the table most recently displayed. It resides in the function pool. It is an output variable whose length is 6. Unless it has been defined otherwise by a program function, ZTDROWS is 6 characters long and will have leading zeros, if necessary.

**ZTDSELS** Created by TBDISPL to indicate the number of selected rows. It includes the current selected row, if one exists, and any pending selected rows. ZTDSELS resides in the function pool. It is an output variable whose length is 4. Unless it has been defined otherwise by a program function, ZTDSELS is 4 characters long and will have leading zeros if necessary.

**ZTDTOP** Created by TBDISPL to indicate the table row number of the top row displayed. ZTDTOP resides in the function pool. It is an output variable whose length is 6. Unless it has been defined otherwise by a program function, ZTDTOP is 6 characters long and will have leading zeros if necessary.

System variables ZTDRET, ZTDADD, ZTDSCRP, ZTDLTOP, ZTDLROWS, ZTDSRID, ZTDAMT, and ZTDSIZE are used exclusively when dynamically building the table being displayed.

Table display service system variables, ZTD*, are not saved as part of the CONTROL DISPLAY SAVE/RESTORE information. The values of these variables may be saved by the dialog developer and restored before resuming the processing of the initial table display. Also, the ZVERB is not saved.

## Panel Control Variables Related to TBDISPL

Control variables used with TBDISPL processing are as follows:

**.AUTOSEL** The variable version of the AUTOSEL parameter. It can be assigned "YES", "NO", or a blank in the )INIT or )REINIT sections. Any assignment made to .AUTOSEL in the )PROC section is ignored.

- YES indicates that auto-selection should be performed if the CSRROW parameter is specified and the user does not explicitly select the row.
- NO indicates that auto-selection should *not* be performed.
- Specifying a blank value is the same as specifying YES, with one exception: if there are no input fields in the model lines, auto-selection will occur only if YES is explicitly specified.

Any setting of this variable takes precedence over the AUTOSEL parameter.

**.CSRROW** The variable version of the CSRROW parameter. It can be assigned the table row number (CRP number) corresponding to the model set on the display where the cursor is to be placed. Any setting of this variable takes precedence over the CSRROW parameter.

## Parameter Variables Related to TBDISPL

Variable names can be specified as TBDISPL parameters, as follows:

**POSITION(crp-name)**
Specifies the name of the variable in which TBDISPL is to store the row number (CRP number) of the current selected row. If there are no selected rows, the CRP will be at the top and the row number returned is zero.

**ROWID(rowid-name)**
Specifies the name of the variable in which TBDISPL is to store the rowid of the current selected row.

The difference between a CRP number and a rowid is as follows:

- A CRP number is an ordinal number; that is, the first row has a CRP number of 00000001, the second row has a CRP number of 00000002, and so on. CRP numbers are associated with "slots" in a table, rather than particular rows. If a new row is inserted after the first row, that new row now has a CRP number of 00000002. What had been row 00000002 is now row 00000003, what had been row 00000003 is now row 00000004, and so on.
- A rowid is a nominal value that uniquely identifies a row. This value stays with the row, even if the row has other rows inserted before it. Note, however, that this identifier is not saved on permanent storage by the TBSAVE or TBCLOSE service.

## Using TBDISPL with Other Services

Consider the following items when using TBDISPL with other services:

1. CONTROL DISPLAY LOCK

This service specifies that the next display output is to leave the user's keyboard locked as the panel is displayed, and ISPF is to simulate an ENTER. This facility can be used to generate crude animation or display an "in process" message during a long-running operation.

Table displays done in conjunction with this service should display panels that have a )REINIT or )PROC section. Otherwise, the simulated ENTER is treated as a no-operation, as described under "TBDISPL Summary" on page 229.

2. CONTROL DISPLAY SAVE and CONTROL DISPLAY RESTORE

   If the dialog wants to invoke a display service (BROWSE, EDIT, DISPLAY, another TBDISPL) before processing pending selected rows, it must invoke the CONTROL DISPLAY service to save and restore the current TBDISPL series environment.

   The dialog should invoke CONTROL DISPLAY SAVE before the non-TBDISPL series display service and CONTROL DISPLAY RESTORE after the non-TBDISPL series display service. For example:

   | Service | Description |
   |---|---|
   | **TBOPEN TAB1** | Open the table |
   | **TBDISPL TAB1 PANEL(PAN1)** | Display the table and panel |
   | **CONTROL DISPLAY SAVE** | Save control information about PAN1 |
   | **DISPLAY PANEL(PAN2)** | Display a second panel |
   | **DISPLAY PANEL(PAN3)** | Display a third panel |
   | **CONTROL DISPLAY RESTORE** | Restore control information about PAN1 |
   | **TBDISPL TAB1** | Reinvoke TBDISPL to process the next selection or redisplay the table and panel |
   | **CONTROL DISPLAY SAVE** | Again save control information about PAN1 |
   | **DISPLAY PANEL(PAN2)** | Display the second panel again |
   | **DISPLAY PANEL(PAN3)** | Display the third panel again |
   | **CONTROL DISPLAY RESTORE** | Again restore control information about PAN1 |
   | **TBDISPL TAB1** | Reinvoke TBDISPL to process the next selection or redisplay the table and panel |

3. BROWSE, EDIT, and DISPLAY

   See item 2.

4. Command Tables

   Do not attempt to use TBDISPL to display a command table currently in use. The results would not be predictable.

5. TBSARG

   When only certain rows from a table are to be displayed, the TBSARG service must be invoked before issuing TBDISPL to establish a search criteria. The search criteria should specify a forward scan through the table. In this case, ROWS(SCAN) must be specified on the )MODEL statement in the panel definition.

6. TBSORT

The TBSORT service can be used freely with the TBDISPL service, even during a TBDISPL series. Note, however, that the pending selected rows will be processed in their original order; that is, in the order they would have been processed had the dialog not invoked the TBSORT service.

## Techniques for Using the TBDISPL Service

The following techniques can be applied in programs and command procedures using the TBDISPL service.

1. Displaying Only Certain Rows

   When only certain rows from a table are to be displayed, the TBSARG service must be invoked before issuing TBDISPL to establish a search criteria. The search criteria should specify a forward scan through the table. In this case, ROWS(SCAN) must be specified on the )MODEL statement in the panel definition.

2. Displaying Table Extension Variables

   As TBDISPL creates the scrollable portion of the display, it reads rows from the table and fills in fields in the model sets with their current values. If a field in a model line is an “extension” variable in the table and does not exist in all rows, TBDISPL repeats its value in model sets to which it does not apply. To prevent this, use the CLEAR(var-name, var-name, ...) keyword on the )MODEL statement. This keyword sets to blank the specified variables before each row is read from the table to fill the scrollable portion.

3. Clearing Already-Processed Select Fields

   As the TBDISPL service is reinvoked to process pending selected rows, the dialog may set to blank the select field for successfully processed rows. This is useful in case there is a redisplay with an error message. The already processed select fields will be blank and the not-yet-processed select fields will still have the user-entered data in them.

   Having the following statements in the )REINIT section of the panel definition could achieve this:

   ```
   If (.msg=' ')
     &Select=' '
     Refresh(Select)
   ```

   where “Select” represents the name of any field in the panel that the dialog wants to clear. The three statements shown above could be on one line. For example:

   ```
   If (.msg=' ')  &Select=' '   Refresh(Select)
   ```

4. Using Auto-Selection

   Consider the following situation:
   - The user has entered invalid data in the select field.
   - The panel is redisplayed with an error message.
   - The user does not change the invalid data but performs some action that results in control returning to the dialog function.

   The model set with the invalid data was not user-selected. If the dialog wants to ensure that the user corrects the invalid data, it should use auto-selection in this situation. That is, the CSRROW parameter or control variable should be specified, and the AUTOSEL parameter or control variable should be blank or YES. This will result in the specified row being selected even if the user did not explicitly select it by modifying the corresponding model set on the display.

The auto-selection feature is normally used when the cursor is placed at invalid data in the scrollable portion and there is an error message displayed. It is not used when the cursor is placed in the scrollable portion for informational purposes.

If the auto-selected row is not displayed on the logical screen because of split screen, PFSHOW, or a floating command line, the cursor is placed at the command field. The dialog should ensure that the user is aware of the auto-selected row by issuing a message when specifying table-row-number.

5. Controlling the Top Row Displayed

   As discussed previously, the user can issue the UP or DOWN command to scroll during a TBDISPL display. Scrolling changes the row that is displayed at the top of the scrollable portion. This topic discusses how the dialog function controls the top row displayed.

   In a typical table display dialog, the TBDISPL service is invoked repeatedly in a loop. The first call results in a display (“the first display”). Subsequent calls can produce a display (“subsequent displays”) or can process pending selected rows (“no display”).

   Controlling the Top Row Displayed in a “First Display”

   The TBDISPL service must be invoked with the PANEL parameter specified to obtain a “first display”. In this case, the current row is the top row displayed. For convenience, a table with its CRP at TOP is treated as though the current row was row 1. The dialog can use any of the services that move the CRP, such as TBSKIP or TBTOP, to make the desired table row the current row.

   Controlling the Top Row Displayed in a “Subsequent Display”

   There are three ways to produce a “subsequent display”:

   a. Invoke TBDISPL with the PANEL parameter specified.
   b. Invoke TBDISPL without the PANEL parameter specified, but with the MSG parameter or .MSG control variable specified.
   c. Invoke TBDISPL without the PANEL parameter specified and without the MSG parameter, or .MSG control variable, specified when there are no pending selected rows.

   In the first case, the current row is the top row displayed. The system variable ZTDTOP contains the row number of the top row displayed on the previous TBDISPL display. The following technique can be useful to control the top row displayed:

```
TBTOP  table                     /* Set CRP to TOP        */
TBSKIP table NUMBER(&ZTDTOP)     /* Set CRP to previous   */
                                 /*  top row displayed    */
VGET (ZVERB ZSCROLLN)            /* Retrieve variables    */
Select                           /* Determine Case        */
 When &ZVERB = 'UP' Then         /* - When scroll UP req  */
  TBSKIP table NUMBER(-&ZSCROLLN)/*  skip back toward top */
 When &ZVERB = 'DOWN' Then       /* - When scroll DOWN req*/
  TBSKIP table NUMBER(&ZSCROLLN) /*  skip forward         */
 Otherwise                       /* - Otherwise, not a    */
End                              /*  scroll request       */
                                 /*                       */
TBDISPL table PANEL(panel)       /* Disp the table and pnl*/
```

   In the second case, the top row displayed is the same as that displayed on the previous display. That is, the previous image is “redisplayed” as the user last saw it, except that the specified message is also shown. Certain fields can have been refreshed and the cursor can be in a different place.

   In the third case, any pending scroll request is honored. That is, if the user had entered any data and issued a scroll request on a previous TBDISPL display,

that scroll request is now honored. If no scroll request was pending, the top row displayed is whatever it was on the previous display.

6. Using Variable Model Lines

   Model lines can be specified dynamically through the use of variable model lines. That is, the attribute characters and field names are not specified in the model section. Instead, a variable whose value contains the attribute characters and field names is specified in column one of the model line.

## Rules Applying to Variable Model Lines

Here are some rules that apply to variable model lines:

- The variable must begin in column 1.
- The variable must be the only data on the model line.
- The length of the value of the variable must not be greater than the screen width.
- The variable must be initialized before the panel is displayed. It is not acceptable to initialize the variable in the )INIT section of the panel definition.
- Changes to the variable that occur within the panel or dialog function are not honored until TBDISPL is invoked again with a nonblank panel name.
- A variable whose value is blank is acceptable.
- If the variable contains the character string “OMIT” in uppercase, lowercase, or in mixed case, starting in column one, then that variable model line will not be used.
- There can be from one to eight model lines. Some can be variable model lines and others can be explicitly specified.
- “Z” variables used as name placeholders are acceptable in variable model lines. Be sure to assign an appropriate value to .ZVARS in the )INIT section.

Figure 9 on page 221 is the panel definition for a panel named VARMOD. Figure 10 on page 222 and Figure 11 on page 222 are two possible types of TBDISPL displays using panel VARMOD.

```
)Attr
  | Type(input)  Intens(high) Just(left)  Caps(on)   Pad('')
  $ Type(&type ) Intens(low ) Just(left)  Caps(off)  Padc('_')
  ø Type(&type ) Intens(low ) Just(left)  Caps(on)   Padc('_')
)Body  Expand(//)
%--/-/-- Customer Information --/-/--
%Command ===>_cmdfld / /  +Scroll ===>_amt +
+
+ Show Address? ==>_QAD+(Yes or No)
+ Allow Update? ==>_QUP+(Yes or No)
+
%Select
%Code   Account     Name &TITLE2
)Model
&MDL1
&MDL2
&MDL3
)Init
  &amt=page
  If (&QAD=' ') &QAD=NO
  If (&QUP=' ') &QUP=NO
  If (&QUP='YES') &TYPE='Input'
  If (&QUP='NO')  &TYPE='Output'
  If (&QAD='YES') .ZVARS='(State)'
)Proc
  &QAD = Trans(Trunc(&QAD,1) Y,YES N,NO ' ',NO *,*)
  Ver(&QAD,List,YES,NO)
  &QUP = Trans(Trunc(&QUP,1) Y,YES N,NO ' ',NO *,*)
  Ver(&QUP,List,YES,NO)

  If (&QAD='YES')
    &TITLE2='and Address'
    &MDL1='|SCODE+øAccount+   $Name                              +'
    &MDL2='                   $Address                         +
$City              + øZ +'
    &MDL3='%================================+
============================================='
  If (&QAD='NO')
    &TITLE2=' '
    &MDL1='|SCODE+øAccount+   $Name                              +'
    &MDL2='OMIT'
    &MDL3='OMIT'
)End
```

*Figure 9. Variable Model Lines – Panel Definition*

Before panel VARMOD is displayed, the dialog function must initialize the variable model lines as follows:

```
&MDL1='|SCODE+øAccount+   $Name                              +'
&MDL2='OMIT'
&MDL3='OMIT'
```

This panel is designed to be displayed in a loop. That is, the TBDISPL service is invoked repeatedly to display the table and panel until the user enters the END or RETURN command.

When the panel is displayed, the user can set the “Show Address?” field (QAD) to YES or NO. If this field is NO (the default), only one model line is used, which shows the customer’s account number and name. If this field is YES, three model lines are used. The first remains unchanged; the second is the customer’s street address, city, and state; and the third contains divider lines. Also, the variable

&TITLE2, which appears in the )BODY section, is set to a nonblank value. This is used as part of the column heading for the scrollable portion.

```
----------------------------- Customer Information ----------------- ROW 1 OF 8
Command ===>                                                  Scroll ===>
 Show Address? ==> NO  (Yes or No)
 Allow Update? ==> NO  (Yes or No)

Select
Code   Account     Name
''  KC10001     Rogers, Kelly
''  KC10002     Holloway, Rich
''  KC10003     Holmes, Karen
''  KC10004     Jones, Ann
''  KC10005     Donavan, Harold
''  KC10006     Bentley, Chris
''  KC10007     Seabold, Matthew
''  KC10007     Fitzgerald, Therese
****************************** BOTTOM OF DATA ********************************
```

*Figure 10. Variable Model Lines – Display 1*

```
----------------------------- Customer Information ----------------- ROW 1 OF 8
Command ===>                                                  Scroll ===>
 Show Address? ==> NO  (Yes or No)
 Allow Update? ==> NO  (Yes or No)

Select
Code   Account     Name and Address
''  KC10001     Rogers, Kelly
                   253 Main St                    Junction City         KS
 ================================================================================
''  KC10002     Holloway, Rich
                   2810 Curtis Lane               Long Beach            CA
================================================================================
''  KC10003     Holmes, Karen
                   3600 Chestnut St               Hyannis               MA
================================================================================
''  KC10004     Jones, Ann
                   212 Fallon Ave                 North Hudson          NY
================================================================================
''  KC10005     Donavan, Harold
                   180 Berthold Ave               Baton Rouge           LA
```

*Figure 11. (Part 1 of 2). Variable Model Lines – Display 1*

```
================================================================================
 ''  KC10006     Bentley, Chris
                    South Mountain Pass             Ashland               NH
================================================================================
 ''  KC10007     Seabold, Matthew
                    42 Dragonica Way                Newark                DE
================================================================================
 ''  KC10008     Fitzgerald, Therese
                    67 Waimea Blvd                  Naalehu               HI
================================================================================
****************************** BOTTOM OF DATA ***********************************
```

*Figure 12. (Part 2 of 2). Variable Model Lines – Display 2*

Panel definition VARMOD has a number of features besides variable model lines:

- It is in mixed case to improve readability.
- The TYPE attribute of the fields ACCOUNT and NAME, as well as ADDRESS, CITY, and STATE, when they are shown, is a variable. When the user sets the "Allow Update?" field (QUP) to NO (the default), the customer information fields (ACCOUNT, NAME, ...) become output fields. That is, they are protected and cannot be updated.

  When the "Allow Update?" field is set to YES, the customer information fields become input fields. The user could then update the displayed information and the dialog function would update the table.
- The title line makes use of the expand character defined on the )BODY statement. This is a convenient way to center the title text. The command line also uses the expand character.
- Many of the lines in the executable sections, here the )INIT and )PROC sections, have more than one statement in them. This saves space and improves readability.
- The first two assignments of &MDL2 and &MDL3 make use of the continuation character "+". This is convenient to use when assigning long strings to a variable.

## An Example of Using the TBDISPL and TBPUT Services

This topic describes the use of the TBDISPL and TBPUT services in a dialog that displays rows of a table for possible modification by a user.

This dialog invokes the TBDISPL service to display a table named TAB1 with a panel named PAN1. The )BODY section of the panel definition corresponds to the fixed (non-scrollable) portion of the display. The )MODEL section of the panel definition corresponds to the scrollable portion of the display. This is where the table rows are displayed. The "model lines" in the )MODEL section are replicated enough times to fill the screen. Each of these replications is known as a model set, and corresponds to a row of the table. The fields in the model sets correspond to table columns.

Changes the user wishes to make in TAB1 are entered on the display directly into fields in the model sets. When the user enters data into a model set, the corresponding table row is said to be selected for processing.

After the user selects one or more rows, the TBDISPL service locates the first selected row and retrieves it. To retrieve a row means to position the CRP to that row, read it, and then store the row values into the function pool. Next, values from the changed model set are stored in the function pool.

The dialog function then invokes the TBPUT service to write the updated function pool variables to the table row. A user can also enter data, such as function commands, into the fixed portion of the display.

The user ends the dialog by entering the END or RETURN command.

This example does not illustrate:

- Logic to insert or delete rows in the table
- Verification of user-entered data by the dialog function or by the )PROC section in the panel definition
- Controlling cursor placement on the display
- Controlling which is the top row displayed.

The function can be started by a user at a terminal by 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.

What follows is first a listing of the complete function, followed by each statement repeated, with supporting text and figures.

## Command Procedure Function

```
1.  TBOPEN TAB1 WRITE
2.  Set &RC = 0
3.  Do while &RC < 8
4.  TBDISPL TAB1 PANEL(PAN1)
5.  Set &RC = return code
6.  Process fixed portion input
7.  Do while &PROCFLAG = ON
8.  Process scrollable portion input TBPUT TAB1
9.  If &ZTDSELS > 1 Then
10. TBDISPL TAB1
11. Else
12. Set &PROCFLAG = OFF
13. End
14. End
15. TBCLOSE TAB1
```

## Description of Function Steps

1. TBOPEN TAB1 WRITE

   Open the table. Read table TAB1 into virtual storage for update. The contents of table TAB1 are shown in Figure 13.

```
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
```

*Figure 13. Five Rows in Table TAB1*

2. Set &RC = 0

   Create a variable that will hold the return code from the TBDISPL service. In this example, the variable is called "RC". Initialize it to zero so that it will enter the loop in step 3.

3. Do while &RC < 8

   Start the main loop. This will keep invoking TBDISPL to display the table until the user enters the END or RETURN command.

4. TBDISPL TAB1 PANEL(PAN1)

   Display information from table TAB1 on panel PAN1. The current row, which is the row the CRP is pointing to, will be the top row displayed. If the CRP is at the top (CRP number zero), then the first row of the table will be the first row displayed. The display, as it appears at the terminal, is shown in Figure 14. Format of the display is controlled by a panel definition named PAN1, shown in Figure 15 on page 226. TBDISPL, besides displaying the table, allows the user to scroll up and down the scrollable data in the display.

```
----------------------------- Employee List ------------------------ ROW 1 OF 5
Command ===>                                              Scroll ===> PAGE

   Notes ==>
Make changes to any information except Employee Serial:

------ Employee Name --------         --- Phone ---          Employee
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
Kelvey       Ann           A          914   555-4156          503774
****************************** BOTTOM OF DATA ********************************
```

*Figure 14. Table TAB1 as Displayed Using Panel PAN1*

```
)Attr
  _Type(Input) Intens(Low)
  # Type(Input) Intens(Low) Caps(off)
)Body
%----------------------------  Employee List  ---------------------------------
%Command ===>_CMDFLD                                        %Scroll ===>_amt +
%
+   Notes ==>#NOTES
+Make changes to any information except Employee Serial:
+
+------ Employee Name -------          --- Phone ---        Employee
+Last         First        MI          Area  Number          Serial
+
)Model
_LNAME     _FNAME    _I       _PHA _PHNUM      _EMPSER
)Init
  &AMT  = PAGE
)Proc
  VPUT (Notes) Profile
)End
```

*Figure 15. Table Display Panel Definition PAN1*

Control will be returned to the dialog function when the user does one of the following:

- Presses the Enter key. The user may or may not have typed data into the fixed or scrollable portion of the screen.

  An exception to this condition occurs if all of the following were true:
  - The user typed no data into the fixed portion of the screen.
  - The user typed no data into the scrollable portion of the screen.
  - The user pressed the Enter key.
  - Panel PAN1 had neither a )REINIT nor a )PROC section. PAN1 does in fact have a )PROC section.

  In this case, control would not be returned to the dialog function.
- Enters the END or RETURN command. This may have been done by the user pressing a function key or by typing the command into the command field and pressing the Enter key. Panel PAN1, which is shown in Figure 15, has a command field named CMDFLD. The user may or may not have typed other data into the fixed or scrollable portion of the screen.
- Enters the UP or DOWN scroll command when data has been typed into the fixed or scrollable portion of the screen.

Control will not be returned to the dialog function when the user does one of the following:

- Presses the Enter key when no data has been typed into the fixed or scrollable portion of the screen and the panel definition contains neither a )REINIT nor a )PROC section.
- Enters the UP or DOWN scroll command when no data has been typed into the fixed or scrollable portion of the screen.
- Enters a system command other than UP, DOWN, END, or RETURN. For example, HELP, SPLIT, or CURSOR.
- Enters an application command that selects another dialog.

When a model set in the scrollable part of the display has been changed, the corresponding table row is said to be a selected row. TBDISPL retrieves the selected row. To retrieve a row means to position the CRP to that row, read it,

and then store the row values into the function pool. Next, values from the changed model set are stored in the function pool. If there are no selected rows, then the CRP is set to zero.

5. Set &RC = return code

   Save the return code from TBDISPL in variable RC. This variable controls the loop starting in step 4. The following return codes are possible:

   **0** There were zero or one selected rows
   **4** There were two or more selected rows
   **8** The user entered the END command. Any number of rows, including zero, may have been selected.

   It is possible that TBDISPL will issue severe error return codes of 12 or 20. Because CONTROL ERRORS CANCEL, the default value, is in effect, ISPF will cancel the dialog function.

6. Process fixed portion input

   Process the data the user typed into the fixed portion of the display. On a table display panel definition, the )BODY section defines the fixed portion of the display and the )MODEL section defines the scrollable portion of the display. Panel PAN1, shown in Figure 15, has three input fields in the )BODY section:

   **CMDFLD** The command field
   **AMT** The scroll amount field
   **NOTES** A "notepad" field

   Users can enter ISPF system commands such as END, RETURN, UP, DOWN, HELP, and SPLIT in the CMDFLD field. Or, they can enter an application command that SELECTs another dialog, if there is such a command defined in the active command table. Users can also enter function commands. These are commands that are handled by the dialog function. CANCEL is an example of a function command. The function could check if CMDFLD had the value CANCEL. If so, a TBEND could be issued. In this example, there would also have to be logic to leave the TBDISPL loop after the TBEND is issued.

   The second input field, AMT, is the scroll amount field. Changes to this field are always handled by ISPF. The TBDISPL service does not consider changes to this field as "input to the fixed portion of the screen".

   The third input field, NOTES, could be used as a small on-screen notepad. The )PROC section of PAN1 uses the VPUT service to put this variable into the profile pool. In this field, the user could write short notes that are to be remembered from one session to the next.

   This example shows the processing of the fixed portion input as step 6. It is done before the processing of the scrollable portion input. This would be natural for handling a CANCEL command. However, if for example, the dialog function also handled a SAVE command, which would result in a TBSAVE, the dialog writer may want that processing to occur after the scrollable portion input processing.

   The processing of the fixed portion input can be placed:

   a. Before the processing of all selected rows (step 6)
   b. After the processing of all selected rows (between steps 13 and 14)
   c. Before the processing of each selected row (between steps 7 and 8)
   d. After the processing of each selected row (between steps 8 and 9)

7. Set &PROCFLAG = ON

   Create a variable that indicates there are selected rows. In this example, the variable is called "PROCFLAG". Initialize this flag to ON so it will enter the loop in step 8.

8. Process scrollable portion input

TBPUT TAB1

Process the scrollable portion input. Here, the current selected row is processed. In this example, the TBPUT service is invoked to update the row. The function pool values of variables corresponding to table columns are written to the table row.

If the processing of the scrollable portion input includes invoking any service that resulted in a display, such as BROWSE, EDIT, DISPLAY, or another TBDISPL, then the CONTROL service must be invoked to save and then restore the table display control information, such as pending selected rows. Example:

```
TBDISPL TAB1 PANEL(PAN1)
                          Display table TAB1 with panel PAN1,
                          assuming you select several rows
CONTROL DISPLAY SAVE      Save "control" information
DISPLAY PANEL(PAN2)       Display panel PAN2
CONTROL DISPLAY RESTORE
                          Restore the "control" information
TBDISPL TAB1              Invoke TBDISPL to get the next selected row
CONTROL DISPLAY SAVE      Save "control" information
DISPLAY PANEL(PAN2)       Display panel PAN2
CONTROL DISPLAY RESTORE
                          Restore the "control" information
```

If non-ISPF displays are processed, instead of using CONTROL DISPLAY SAVE and CONTROL DISPLAY RESTORE, use CONTROL DISPLAY REFRESH either before or after the non-ISPF display is done.

9. If &ZTDSELS > 1 Then

   Determine if there are any pending selected rows. If ZTDSELS is zero, there were no selected rows and this step would not have been reached (see Step 7). If ZTDSELS is one, then there was one selected row. This is the current row and there are no pending selected rows. If ZTDSELS is more than one, then there is the current selected row and at least one pending selected row.

10. TBDISPL TAB1

    Reinvoke TBDISPL without the PANEL or MSG parameter to get the next selected row. That is, the CRP will be positioned to the next selected row to retrieve that row, and the function pool values of variables corresponding to fields in the scrollable portion will be updated to reflect changes made to the corresponding model set on the display.

11. Else

    Since ZTDSELS is not greater than one (Step 9) but is greater than zero (Step 7), then ZTDSELS must equal one. This means that there are no pending selected rows.

12. Set &PROCFLAG = OFF

    Force control to leave the loop started in Step 7. All selected rows have been processed.

13. End

    End the selected row processing loop.

14. End

    End the main loop, which displays table TAB1 with panel PAN1.

15. TBCLOSE TAB1

    Close table TAB1. Write the updated version of TAB1 onto disk, and delete the virtual storage copy.

## TBDISPL Summary

1. Floating command line

   If the command line for a table display panel has been moved to the bottom position, and if no alternate placement has been specified for the long message line, the line directly above the repositioned command line is reserved (left blank) for the display of long messages. Otherwise, if a user entered erroneous data on that line, a long message could overlay that data.

   ISPF adjusts display scrolling to account for the line reserved for long messages.

2. TBDISPL does not modify the table

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

3. Displaying an empty table

   It is acceptable to invoke TBDISPL to display a table with no rows. The scrollable portion will consist only of the bottom-of-data marker. In previous versions, this resulted in a severe error, return code = 20, message = ISPT051.

4. CSRROW and auto-selection

   Specifying the CSRROW parameter or control variable without setting the AUTOSEL parameter or control variable to “NO” results in the row being selected, even if the user did not explicitly select the row. This is called auto-selection.

5. Dual defaults for CAPS and JUST

   In the )BODY section of a table display panel, input and output fields default to CAPS(ON) and JUST(LEFT). In the )MODEL section, they default to CAPS(OFF) and JUST(ASIS). These dual defaults exist to allow both new capability in this version and compatibility with previous versions of the product.

6. Effect of having a )REINIT or )PROC section

   TBDISPL behavior is affected by whether a )REINIT or )PROC section is included in the panel definition. When a )REINIT or )PROC section is included, and the user makes no modification to the screen and presses the Enter key, TBDISPL returns control to the dialog function. On the other hand, if neither a )REINIT nor a )PROC section is included, and the user makes no modification to the screen and presses the Enter key, TBDISPL treats this as a “no operation”, and control does not return to the dialog function. This is to allow both new capability in this version and compatibility with previous versions of the product.

7. Search arguments in conjunction with TBDISPL

   Only search arguments specifying a forward scan through the table should be used in conjunction with TBDISPL. Otherwise, TBDISPL does not support scrolling through the display.

8. TBDISPL parameters and their categories:

```
          Required    Optional
 Service  Parameter   Parameters                 Categories

 TBDISPL  table-name                            in  name
                    [PANEL(panel-name)]         in  name
                    [MSG(message-id)]           in  name
                    [CURSOR(field-name)]        in  name
                    [CSRROW(table-row-number)]  in  number
                    [CSRPOS(cursor-position)]   in  number
```

```
                    [AUTOSEL(YES|NO)]          in  key
                    [POSITION(crp-name)]       out number
                    [ROWID(rowid-name)]        out number
```

**in** Indicates that the parameter is used to pass information from the dialog to ISPF.

**out** Indicates that the parameter is used to enable ISPF to pass information to the dialog. ISPF will create a variable with the indicated name.

**key** Indicates it is a keyword parameter.

**name** Indicates the value specified in the parameter is a name.

**number** Indicates the value specified in the parameter is a number.

9. The following items can appear in the )BODY section of a table display panel definition:
   - Text
   - Variables within text, such as "&XYZ"
   - Input fields
   - Output fields
   - Dynamic areas that are not scrollable or extendable
   - A graphic area that is not extendable.
10. The following items cannot appear in the )BODY section of a table display panel definition:
   - Dynamic areas that are scrollable or extendable
   - More than one graphic area. This is true for any panel
   - A graphic area that is extendable. Graphic areas are never scrollable.
11. The following items can appear in the )MODEL section of a table display panel definition:
   - Text
   - Variable model lines
   - Input fields
   - Output fields.
12. The following items cannot appear in the )MODEL section of a table display panel definition:
   - Variables within text
   - Dynamic areas
   - Graphic areas.
13. During TBDISPL display, the following user actions return control to the dialog function:
   - Pressing the Enter key. See item 6 in "TBDISPL Summary" on page 229 for an exception.
   - Entering the END or RETURN command
   - Entering the UP or DOWN scroll command when data has been typed into the fixed or scrollable portion of the screen
   - Entering the UP or DOWN scroll command when using dynamic table expansion and more rows are needed to satisfy the scroll request.
14. During TBDISPL display, the following user actions do not return control to the dialog function:
   - Pressing the Enter key when no data has been typed into the fixed or scrollable portion of the screen and the panel definition has neither a )REINIT nor a )PROC section
   - Entering the UP or DOWN scroll command *without* typing data into the fixed or scrollable portion of the screen. Also, control does not return to the dialog function in either of the following two cases:
     – Dynamic table expansion is not defined
     – Dynamic table expansion is defined and the table already contains enough rows to satisfy the scroll.

- Entering a system command other than UP, DOWN, END, or RETURN. For example, HELP, SPLIT, PRINT, or CURSOR.
- Entering an application command that selects another dialog.

15. The following return codes are possible from TBDISPL:
    **0** There were zero or one selected rows. A scroll may be pending.
    **4** There were two or more selected rows.
    **8** The END or RETURN command was entered. Any number of rows, including zero, may have been selected.
    **12** The specified panel or message could not be found or the specified table was not open.
    **20** Severe error.
16. Levels of commands:

    **System commands**
    Provided by ISPF and always available to a user, unless explicitly overridden by an application. For example: END, UP, HELP, PRINT.

    **Application commands**
    Available to a user throughout operation of an application. For example: a command defined in the active command table that SELECTs another dialog.

    **Function commands**
    Meaningful only while operating a particular function within an application. For example, the dialog function can be designed so that TBSORT is invoked when the user enters "SORT" in the command field.
17. Commands can be entered by:
    - Typing information into the command field and pressing the Enter key
    - Pressing a function key
    - Selecting an ATTENTION FIELD using the light pen or cursor select key.
18. TBDISPL does not rebuild the display until all selected rows have been successfully processed. Therefore, the CRPs of the displayed table will not match those of the actual table if the order or structure of the table is changed within a TBDISPL series. This can affect correct cursor row placement for a redisplay with message while in the series.

    It is recommended that any verification of selected rows be done for all selected rows before performing operations that change the order or structure of the table. This requires that selected row IDs be saved until all selected rows have been retrieved and validated. This affects only the cursor placement as just described. The value passes back in the name specified with the POSITION keyword contains the CRP of the row in the actual table.

## TBEND – Close a Table without Saving

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

A TBEND request for a shared table causes the use count in the table for that logical screen to be decremented by one. If the use count for all logical screens is zero, the TBEND service is performed. Otherwise, no action occurs, and the table is available for continued processing in any screen that still has a use count greater than zero.

### Command Invocation Format

```
ISPEXEC  TBEND  table-name
```

## Call Invocation Format

```
CALL  ISPEXEC (buf-len, buffer);

OR

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

## Parameters

**table-name**
Specifies the name of the table to be ended.

**buf-len**
Specifies a fullword fixed binary integer containing the length of "buffer."

**buffer**
Specifies a buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC call for a command procedure.

## Return Codes

The following return codes are possible:

**0** Normal completion.

**12** Table is not open.

**20** Severe error.

## Example

Delete the virtual storage copy table TELBOOK; do not change any permanent copy in the table library.

```
ISPEXEC TBEND TELBOOK
```

Set the program variable BUFFER to contain:

```
TBEND TELBOOK
```

Set program variable BUFLEN to the length of the variable BUFFER. Issue the following:

```
CALL ISPEXEC (BUFLEN, BUFFER);
```

or alternately

```
CALL ISPLINK ('TBEND   ','TELBOOK ');
```

# TBERASE – Erase a Table

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

The table must *not* be open in WRITE mode when this service is invoked.

## Command Invocation Format

```
ISPEXEC  TBERASE  table-name [LIBRARY(library)]
```

## Call Invocation Format

```
CALL  ISPEXEC (buf-len, buffer);

OR

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

## Parameters

**table-name**
Specifies the name of the table to be erased.

**library**
Specifies the name of a DD statement or LIBDEF lib-type that defines the library in which the table exists. If this parameter is omitted, the default is ISPTABL.

**buf-len**
Specifies a fullword fixed binary integer containing the length of "buffer."

**buffer**
Specifies a buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC call for a command procedure.

## Return Codes

The following return codes are possible:

| | |
|---|---|
| **0** | Normal completion. |
| **8** | Table does not exist in the output library. |
| **12** | Table in use; ENQ failed. |
| **16** | Table output library not allocated. |
| **20** | Severe error. |

## Example

Erase the table TELBOOK from the table library.

```
ISPEXEC TBERASE TELBOOK
```

Set the program variable BUFFER to contain:

```
TBERASE TELBOOK
```

Set program variable BUFLEN to the length of the variable BUFFER. Issue the following:

```
CALL ISPEXEC (BUFLEN, BUFFER);
```

or alternately

```
CALL ISPLINK ('TBERASE ','TELBOOK ');
```

# TBEXIST – Determine Whether a Row Exists in a Table

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

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

This service is not valid for non-keyed tables and causes the current row pointer (CRP) to be set to the top.

## Command Invocation Format

```
ISPEXEC  TBEXIST  table-name
```

## Call Invocation Format

```
CALL  ISPEXEC (buf-len, buffer);
```

*OR*

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

## Parameters

**table-name**
Specifies the name of the table to be searched.

**buf-len**
Specifies a fullword fixed binary integer containing the length of "buffer."

**buffer**
Specifies a buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC call for a command procedure.

## Return Codes

The following return codes are possible:

**0** Normal completion; the CRP is positioned to the specified row.

**8** Keyed tables: the specified row does not exist; the CRP is set to TOP (zero).

Non-keyed tables: service not possible; the CRP is set to TOP.

**12** Table is not open.

**20** Severe error.

## Example

In the keyed table TELBOOK, test for the existence of a row having a specific key value.

```
ISPEXEC TBEXIST TELBOOK
```

If return code = 0, the row exists.

Set the program variable BUFFER to contain:

```
TBEXIST TELBOOK
```

Set program variable BUFLEN to the length of the variable BUFFER. Issue the following:

```
CALL ISPEXEC (BUFLEN, BUFFER);
```

If return code = 0, the row exists.

or alternately

```
CALL ISPLINK ('TBEXIST ','TELBOOK ');
```

If return code = 0, the row exists.

# TBGET – Retrieve a Row from a Table

The TBGET service accesses a row in a table. If the NOREAD parameter is not specified, the row values are read into the function pool.

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

For tables without keys, the row pointed to by the current row pointer (CRP) is fetched. You can use the TBSCAN, TBSKIP, TBBOTTOM, and TBTOP services to position the CRP.

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

All variables in the row, including key and name variables, if any, are stored into the corresponding dialog variables. A list of extension variable names can also be retrieved.

## Command Invocation Format

```
ISPEXEC  TBGET  table-name  [SAVENAME(var-name)]
                            [ROWID(rowid-name)]
                            [NOREAD]
                            [POSITION(crp-name)]
```

## Call Invocation Format

```
CALL  ISPEXEC (buf-len, buffer);

OR

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

## Parameters

**table-name**
Specifies the name of the table to be read.

**var-name**
Specifies the name of a variable into which a list of extension variable names

contained in the row will be stored. The list is enclosed in parentheses, and the names within the list are separated by a blank.

**rowid-name**
Specifies the name of a variable in which a number that uniquely identifies the row being accessed is to be stored. Later, this identifier can be specified in the ROW parameter of TBSKIP to cause the CRP to be positioned to the row. This identifier is not saved on permanent storage by TBSAVE or TBCLOSE.

**NOREAD**
Specifies that the variables contained in the requested row are not to be read into the variable pool.

**crp-name**
Specifies the name of a variable in which the row number pointed to by the CRP is to be stored. If the CRP is positioned to TOP, the row number returned is zero.

**buf-len**
Specifies a fullword fixed binary integer containing the length of "buffer."

**buffer**
Specifies a buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC call for a command procedure.

## Return Codes

The following return codes are possible:

**0** Normal completion.

**8** Keyed tables: The row specified by the value in the key variables does not exist in any row after the current row pointer, the CRP is set to TOP (ZERO).

Non-keyed tables: the CRP was at TOP and remains at TOP.

**12** Table is not open.

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

**20** Severe error.

## Example

In the keyed table TELBOOK, from a row having a specific key value, copy variable values into the respective function pool variables having the same names.

```
ISPEXEC TBGET TELBOOK
```

Set the program variable BUFFER to contain:

```
TBGET TELBOOK
```

Set program variable BUFLEN to the length of the variable BUFFER. Issue the following command:

```
CALL ISPEXEC (BUFLEN, BUFFER);
```

or alternately

```
CALL ISPLINK ('TBGET   ','TELBOOK ');
```

# TBMOD – Modify a Row in a Table

The TBMOD service unconditionally updates a row in a table.

For tables with keys, the table is searched for the row to be updated. The current contents of the key variables, dialog variables that correspond to keys in the table, are used as the search argument. If a match is found, the row is updated. If a match is not found, a TBADD is performed, adding the row to the end of the table (or it is added at an appropriate point for maintaining the table) in the sequence specified in a previously processed TBSORT request.

For tables without keys, TBMOD is equivalent to TBADD. The following processing takes place: any new row is added either immediately following the current row, pointed to by the current row pointer (CRP), or it is added at a point appropriate for maintaining the table in the sequence specified in a previously processed TBSORT request.

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

The current contents of all dialog variables that correspond to columns in the table, keys and names, are saved in the row.

Additional variables, not specified when the table was created, can also be saved in the row. These “extension” variables apply only to this row, not to the entire table. Whenever the row is updated, the extension variables must be specified again if they are to be rewritten.

When the TBMOD service uses the TBADD service to add rows to a table, the default value for number-of-rows parameter of the MULT keyword for TBADD can affect TBMOD execution. See the description of the TBADD service for information.

## Command Invocation Format

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

## Call Invocation Format

```
CALL  ISPEXEC (buf-len, buffer);

OR

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

## Parameters

**table-name**
Specifies the name of the table to be updated.

**name-list**
Specifies a list of extension variables, by name, that are to be saved in the row, in addition to the variables specified when the table was created.

**ORDER**
Specifies that any new row is to be added or inserted in the order specified in

the sort information record. A TBSORT must have been performed for this table before use of this keyword. For tables with keys, the row is updated and then reordered if necessary. If a match is not found or the table does not have keys, the row is added at a point appropriate for maintaining the table in the sequence specified by the sort information record. This keyword is ignored if the table has never been sorted. If this keyword is omitted, any existing sort information record is nullified.

When a newly inserted row has sort field-names equal to the sort field-names of an existing row, the insertion is made after the existing row.

**buf-len**
Specifies a fullword fixed binary integer containing the length of "buffer."

**buffer**
Specifies a buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC call for a command procedure.

## Return Codes

The following return codes are possible:

**0** Normal completion. Keyed tables: Existing row updated. Non-keyed tables: New row added to table.

**8** Keys did not match; new row added to the table. Returned only for tables with keys.

**12** Table is not open.

**16** Numeric conversion error; see numeric restrictions for TBSORT. Returned only for sorted tables.

**20** Severe error.

## Example

Update or add a row of variables in the table TELBOOK using values from variables in the function variable pool.

```
ISPEXEC TBMOD TELBOOK
```

Set the program variable BUFFER to contain:

```
TBMOD TELBOOK
```

Set program variable BUFLEN to the length of the variable BUFFER. Issue the following:

```
CALL ISPEXEC (BUFLEN, BUFFER);
```

or alternately

```
CALL ISPLINK ('TBMOD   ','TELBOOK ');
```

# TBOPEN – Open a Table

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

An ENQ is issued to ensure that no other user is currently accessing the table. The ENQ applies only to the specified table in the table (member) in the table input

library, not the entire library. For the WRITE option, an exclusive ENQ remains in effect until the table is closed. For the NOWRITE option, a shared ENQ remains in effect only during the time that the table is read into storage.

## Command Invocation Format

```
ISPEXEC  TBOPEN  table-name  [WRITE|NOWRITE]
                             [LIBRARY(library)]
                             [SHARE]
```

## Call Invocation Format

```
CALL  ISPEXEC (buf-len, buffer);

OR

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

## Parameters

**table-name**
Specifies the name of the table to be opened.

**WRITE**
Specifies that the table is being accessed for update. The updated table can subsequently be saved on disk by use of the TBSAVE or TBCLOSE service. This option is the default.

The WRITE/NOWRITE usage of a shared table must be consistent on all TBOPEN and TBCREATE requests. That is, all requests for a given shared table that result in concurrent use of that table must specify the same WRITE or NOWRITE attribute.

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

**library**
Specifies the name of a DD statement or LIBDEF lib-type that defines the input library. If specified, a generic (non-ISPF) DD name must be used. If this parameter is omitted, the default is ISPTLIB.

**SHARE**
Specifies that the table in virtual storage can be shared between logical screens while the user is in split-screen mode. The TBOPEN request from the first logical screen reads the table into virtual storage and opens it. Subsequent TBOPEN requests from other logical screens use the same table (and same CRP) that is in virtual storage.

A successful TBOPEN or TBCREATE request causes the use count in the table to be incremented by one. The use count determines the action taken by subsequent TBEND and TBCLOSE requests.

**buf-len**
Specifies a fullword fixed binary integer containing the length of “buffer.”

**buffer**
Specifies a buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC call for a command procedure.

## Return Codes

The following return codes are possible:

**0** Normal completion.

**8** Table does not exist.

**12** ENQ failed; table was in use by another user or the current user.

**16** Table input library was not allocated.

**20** Severe error.

## Example

Access (open) the table TELBOOK for updating.

```
ISPEXEC TBOPEN TELBOOK WRITE
```

Set the program variable BUFFER to contain:

```
TBOPEN TELBOOK WRITE
```

Set program variable BUFLEN to the length of the variable BUFFER. Issue the following command:

```
CALL ISPEXEC (BUFLEN, BUFFER);
```

or alternately

```
CALL ISPLINK ('TBOPEN  ','TELBOOK ','WRITE   ');
```

# TBPUT – Update a Row in a Table

The TBPUT service updates the current row of a table.

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

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

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

Additional variables not specified when the table was created, can also be saved in the row. These “extension” variables apply only to the row, not to the entire table. Whenever the row is updated, the extension variables must be specified again if they are to be rewritten.

## Command Invocation Format

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

## Call Invocation Format

```
CALL  ISPEXEC (buf-len, buffer);

OR

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

## Parameters

**table-name**
Specifies the name of the table to be updated.

**name-list**
Specifies a list of extension variables, by name, that are to be saved in the row, in addition to the variables specified when the table was created.

**ORDER**
Specifies that, if necessary, the updated row is to be moved in the table to a point that preserves the order specified in the sort information record. A TBSORT must have been performed for this table before use of this keyword. This keyword is ignored if the table has never been sorted. If this keyword is omitted, any existing sort information record is nullified.

When a newly repositioned row has sort field-names equal to the sort field-names of an existing row, the row is inserted after the existing row.

**buf-len**
Specifies a fullword fixed binary integer containing the length of "buffer."

**buffer**
Specifies a buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC call for a command procedure.

## Return Codes

The following return codes are possible:

**0** Normal completion.

**8** Keyed tables: The key does not match that of the current row; CRP set to TOP (zero).

Non-keyed tables: CRP was at TOP and remains at TOP.

**12** Table is not open.

**16** For sorted tables: numeric conversion error; see numeric restrictions for TBSORT.

**20** Severe error.

## Example

Update a row, in the table TELBOOK, using values from variables in the function variable pool.

```
ISPEXEC TBPUT TELBOOK
```

Set the program variable BUFFER to contain:

```
TBPUT TELBOOK
```

Set program variable BUFLEN to the length of the variable BUFFER. Issue the following command:

```
CALL ISPEXEC (BUFLEN, BUFFER);
```

or alternately

```
CALL ISPLINK ('TBPUT   ','TELBOOK ');
```

# TBQUERY – Obtain Table Information

The TBQUERY service returns information about a specified table, which must have been opened (TBOPEN) by the current user before invoking this service. The following information can be obtained:

- The number of key fields and their names
- The number of all other columns and their names
- The number of rows and the current row position
- The sort arguments in the last invocation of TBSORT
- Details of the last invocation of TBSARG

All the parameters except for table-name are optional. If all of the optional parameters are omitted, TBQUERY simply validates the existence of an open table.

## Command Invocation Format

```
ISPEXEC  TBQUERY  table-name  [KEYS(key-name)]
                              [NAMES(var-name)]
                              [ROWNUM(rownum-name)]
                              [KEYNUM(keynum-name)]
                              [NAMENUM(namenum-name)]
                              [POSITION(crp-name)]
                              [SORTFLDS(srt-name)]
                              [SARGLIST(lst-name)]
                              [SARGCOND(cond-name)]
                              [SARGDIR(dir-name)]
```

## Call Invocation Format

```
CALL  ISPEXEC (buf-len, buffer);

OR

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

```
                                        [,srt-name]

                                        [,lst-name]

                                        [,cond-name]

                                        [,dir-name] );
```

## Parameters

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

**key-name**
Specifies the name of a variable into which a list of key variable names contained in the table will be stored. A list that is not null will be enclosed in parentheses, and the names within the list will be separated by a blank. If no key variables are defined for the table, the key-name variable is set to null.

**var-name**
Specifies the name of a variable into which a list of variable names in the table, specified with the NAMES keyword when the table was created, will be stored. The list will be enclosed in parentheses, and the names within a list that is not null will be separated by a blank. If no name variables are defined for the table, the var-name variable is set to null.

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

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

**namenum-name**
Specifies the name of a variable in which the number of variables in the table specified with the NAMES keyword when the table was created will be stored.

**crp-name**
Specifies the name of a variable in which the row number pointed to by the CRP is to be stored. If the CRP is positioned to TOP, the row number returned is zero.

**srt-name**
Returns the sort arguments as they were presented to TBSORT. If no sort is active for the table then the srt-name variable is set to null.

**lst-name**
Returns the name-list that was last presented to the TBSARG service ARGLIST parameter. If ARGLIST is not currently active then lst-name variable is set to null.

**cond-name**
Returns the list of name-cond pairs that was last presented to the TBSARG service NAMECOND parameter.

**dir-name**
Returns the current direction of the search (NEXT or PREVIOUS) established for TBSCAN.

**buf-len**
Specifies a fullword fixed binary integer containing the length of “buffer”.

**buffer**
Specifies a buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC call for a command procedure.

## Return Codes

The following return codes are possible:

**0** Normal completion.

**12** Table is not open.

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

**20** Severe error.

## Example

For the keyed table TELBOOK:
- In dialog variable QKEYS, store the names of key variables.
- In dialog variable QNAMES, store the names of non-key variables.
- In dialog variable QROWS, store the number of rows.

```
ISPEXEC TBQUERY TELBOOK KEYS(QKEYS) NAMES(QNAMES) ROWNUM(QROWS)
```

Set the program variable BUFFER to contain:

```
TBQUERY TELBOOK KEYS(QKEYS) NAMES(QNAMES) ROWNUM(QROWS)
```

Set program variable BUFLEN to the length of the variable BUFFER. Issue the following:

```
CALL ISPEXEC (BUFLEN, BUFFER);
```

or alternately

```
CALL ISPLINK ('TBQUERY ','TELBOOK ',
              'QKEYS ','QNAMES ','QROWS ');
```

# TBSARG – Define a Search Argument

The TBSARG service establishes a search argument for scanning a table by using the TBSCAN or TBDISPL services. When TBSARG is used in conjunction with TBDISPL, the panel definition referred to by the TBDISPL request must contain a specification of ROWS(SCAN) on the )MODEL statement in the panel definition.

The direction of the scan, forward or backward, can be specified. The conditions that terminate the subsequent scan can also be specified.

The search argument is specified in dialog variables that correspond to columns in the table, including key variables. A value of null for one of the dialog variables means that the corresponding table variable is not to be examined during the search. However, the variable will be examined if the NOBSCAN parameter was specified when the variable was defined using the VDEFINE service.

Generally, TBSARG is used before TBSCAN or TBDISPL operations to establish search arguments for these operations. To set up a search argument, set table variables in the function pool to nulls by using TBVCLEAR. Next, set a value in each variable in the function pool that is to be part of the search argument. Then,

issue TBSARG to establish this variables as the search argument to be used in subsequently requested TBSCAN or TBDISPL operations.

Use the NAMECOND list to establish search argument conditions. For any table variable that was given a value in the function pool, but is not specified in the NAMECOND list, the default is EQ.

Only extension variables can be included in the search argument. They are included by specifying their names in the name-list parameter. The values of these variables become part of the search argument. A null value in an extension variable is a valid search argument.

A search argument of the form AAA* means that only the characters up to the asterisk (*) are compared. This is called a generic search argument. A generic search argument is specified by placing an asterisk in the last nonblank position of the argument. Asterisks embedded in the argument are treated as data. For example, to perform a generic search for a row value of DATA*12, the generic search argument is:

```
DATA*12*
```

The first asterisk is part of the search argument. The second asterisk designates the argument to be a generic search argument.

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

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

You can use either a DBCS or a MIX (DBCS and EBCDIC) format string as a search argument. If either is used as a generic search argument, it 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 format.

The position of the current row pointer (CRP) is not affected by the TBSARG service.

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

Comparisons between the row values and the argument list are always done on a character basis. That is, the values are considered character data, even if they represent numbers.

## Command Invocation Format

```
ISPEXEC  TBSARG  table-name  [ARGLIST(name-list)]
                             [NEXT|PREVIOUS]
                             [NAMECOND(name-cond-pairs)]
```

## Call Invocation Format

```
CALL  ISPEXEC (buf-len, buffer);

OR

CALL  ISPLINK ('TBSARG  ', table-name
                             [,name-list]
                             [,'NEXT    '|'PREVIOUS']
                             [,name-cond-pairs]);
```

## Parameters

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

**name-list**
Specifies a list of extension variables, by name, whose values are to be used as part of the search argument. See "Invoking the ISPF Services" on page 2 for specification of name lists.

**NEXT**
Specifies that the scan is to proceed from the row following the current row to the bottom of the table. This is the default.

**PREVIOUS**
Specifies that the scan is to proceed from the row preceding the current row to the top of the table. To scan the bottom row, CRP must be positioned to TOP.

**name-cond-pairs**
Specifies a list of names and conditions for determining the search argument conditions for scanning a table. There must be one condition specified for every name specified in the list. This list is used to associate a particular operator (condition) with a previously established scan argument. This parameter does not affect how the search arguments are established.

The name-cond-pairs syntax is as follows:

```
(name1,condition1,name2,condition2  ...)
```

Each "name" must be the name of a key field, name field, or name of an extension variable for the table. If the specified name does not exist, a severe error is encountered.

The "condition" specifies the scan condition for the "name" (column) to which it is paired. The search arguments are specified in dialog variables that correspond to columns in the table, and this determines the columns that take place in the search.

The valid condition-values are EQ, NE, LE, LT, GE, and GT. If some or all condition-value-pairs are not specified, the default is EQ for those columns

participating in the search. Each argument and its associated operator are treated as separate entities, not as subfields of a single argument. The condition-values LE, LT, GE, and GT might have a date indicator immediately following. The date indicator is Y*n*, where Y indicates that the variable name associated with the condition-value is a date, and *n* is an integer from 1 to 7 indicating the offset within the variable value where the year begins. The year should be a 2-digit year, because a century value is inserted in front of the 2-digit year for compare purposes. The following meanings are associated with the condition-values:

**EQ** Specifies that the search is for an equal condition between the argument value and the row value. This is the default.

**NE** Specifies that the search is for a row value not equal to the argument value.

**LE** Specifies that the search is for a row value less than or equal to the argument value.

**LT** Specifies that the search is for a row value less than the argument value.

**GE** Specifies that the search is for a row value greater than or equal to the argument value.

**GT** Specifies that the search is for a row value greater than the argument value.

**Yn** Can be used with LE, LT, GE, and GT. It must immediately follow one of the four allowed condition-values. The Y indicates that the paired variable name is a date variable that needs a century value added to a 2-digit year so that dates can be compared correctly. The *n* is a number from 1 to 7 that gives the offset within the variable value where the year is located.

**buf-len**
Specifies a fullword fixed binary integer containing the length of "buffer."

**buffer**
Specifies a buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC call for a command procedure.

## Return Codes

The following return codes are possible:

**0** Normal completion.

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

**12** Table is not open.

**20** Severe error.

## Examples

Establish a search argument to be used by a TBSCAN operation of the table TELBOOK. Assume that LNAME and ZIPCODE are columns in table TELBOOK. Specify a scan direction of forward and terminate the scan when the row value for the LNAME column is equal to "JOHNSON" and the ZIPCODE column is greater than 08007.

- Invoke TBVCLEAR for table TELBOOK
- Set variable LNAME to JOHNSON
- Set variable ZIPCODE to 08007
- Issue the following request:

```
ISPEXEC TBSARG TELBOOK NEXT NAMECOND(LNAME,EQ,ZIPCODE,GT)
```

Set the program variable BUFFER to contain:

```
TBSARG TELBOOK NEXT NAMECOND(LNAME,EQ,ZIPCODE,GT)
```

Set program variable BUFLEN to the length of the variable BUFFER. Issue the following:

```
CALL ISPEXEC (BUFLEN, BUFFER);
```

or alternately

```
CALL ISPLINK ('TBSARG  ','TELBOOK ',' ','NEXT    ',
              '(LNAME,EQ,ZIPCODE,GT)');
```

Establish a search argument to be used by a TBSCAN operation of the table DATETBL. Assume DATE1 to be a name variable in table DATETBL and that the dates are in a yy/mm/dd format. Specify a scan direction of forward and terminate the scan when the row value of DATE1 is greater than 99/01/31.

- Invoke TBVCLEAR for table DATETBL
- Set variable DATE1 to 99/01/31
- Issue the following TBSARG request:

```
ISPEXEC TBSARG DATETBL NEXT NAMECOND(DATE1,GTY1)
```

# TBSAVE – Save a Table

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

When storing a table in an output file, the user can give it a new name. The table name used in the output library must not be an alias name.

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

Table output can be directed to a table output library other than the library specified on the table output ISPTABL DD statement or LIBDEF service request. The library to be used must be allocated before table services receives control. Thus, an application can update a specific table library. This is particularly useful for applications that need to maintain a common set of tables containing their data.

## Command Invocation Format

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

## Call Invocation Format

```
CALL  ISPEXEC (buf-len, buffer);

OR

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

## Parameters

**table-name**
Specifies the name of the table to be saved.

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

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

A comparison is made between the virtual storage size of the table and the external size in the table output library. If there is insufficient storage to write the table in-place, it is written at the end of the table output library.

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

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

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

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

The default value for this parameter is zero.

**library**
Specifies the name of a DD statement or LIBDEF lib-type that defines the output library in which table-name is to be saved. If specified, a generic (non-ISPF) DD name must be used. If this parameter is omitted, the default is ISPTABL.

**buf-len**
Specifies a fullword fixed binary integer containing the length of "buffer."

**buffer**
Specifies a buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC call for a command procedure.

## Return Codes

The following return codes are possible:

**0** Normal completion.

**12** Table is not open.

**16** Alternate table output library was not allocated.

**20** Severe error.

## Example

Write a table, TELBOOK, previously opened and currently in virtual storage, to the table library. Retain the copy in virtual storage for further processing. Do not close the table.

```
ISPEXEC TBSAVE TELBOOK
```

Set the program variable BUFFER to contain:

```
TBSAVE TELBOOK
```

Set program variable BUFLEN to the length of the variable BUFFER. Issue the following:

```
CALL ISPEXEC (BUFLEN, BUFFER);
```

or alternately

```
CALL ISPLINK ('TBSAVE  ','TELBOOK ');
```

# TBSCAN – Search a Table

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

The search can be either in a forward or a backward direction. The forward direction starts with the row after the current row and continues to the end of the table. The backward direction starts with the row before the CRP and continues to the top of the table. If a match is found, the CRP is set to that row. The row is retrieved unless the NOREAD parameter is specified. All variables in the row, including keys and extension variable, if any, are stored in the corresponding variables in the function pool. A list of extension variable names can also be retrieved.

Use of the name-list parameter is optional. If specified, it overrides the search argument set by the TBSARG service for this search only. The values of all variables specified in the name-list parameter become part of the search argument. Key, name, and extension variables can be specified.

A value of the form AAA* means that only the characters This is called a generic search argument. A generic search argument is specified by placing an asterisk in

the last nonblank position of the argument. Asterisks embedded in the argument are treated as data. For example, to perform a generic search for a row value of DATA*12, the generic search argument is:

```
DATA*12*
```

The first asterisk is part of the search argument. The second asterisk designates the argument as a generic search argument. In a CLIST, the following technique can be used to set a variable to a literal value that ends with an asterisk:

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

A null value in a variable is a valid search argument.

You can use either a DBCS or a MIX (DBCS and EBCDIC combined) format string as a search argument. If either is used as a generic search argument, it must be specified as follows:

- DBCS format string

  ```
  DBDBDBDB**
  ```

  where DBDBDBDB represents a DBCS string and ** is a single DBCS character representing the asterisk (*).
- MIX (DBCS and EBCDIC combined) format string

  ```
  eeee[DBDBDBDBDB]*
  ```

  where eeee represents an EBCDIC string, DBDBDBDB represents a DBCS string, [ and ] represent shift-out and shift-in characters, and * is an asterisk in single-byte format.

Comparisons between the row values and the argument list are always done on a character basis. That is, the values are considered character data, even if they represent numbers.

## Command Invocation Format

```
ISPEXEC  TBSCAN  table-name  [ARGLIST(name-list)]
                            [SAVENAME(var-name)]
                            [ROWID(rowid-name)]
                            [NEXT|PREVIOUS]
                            [NOREAD]
                            [POSITION(crp-name)]
                            [CONDLIST(condition-value-list)]
```

## Call Invocation Format

```
CALL  ISPEXEC (buf-len, buffer);

OR

CALL  ISPLINK ('TBSCAN  ', table-name [,name-list]
                                      [,var-name]
                                      [,rowid-name]
                                      ['NEXT    '|'PREVIOUS']
```

```
[,'NOREAD  ']

[,crp-name]

[,condition-value-list]);
```

## Parameters

**table-name**
Specifies the name of the table to be searched.

**name-list**
Specifies a list of key, name, or extension variables, by name, whose values are to be used as the search argument. Use of the name-list parameter is optional. If specified, it overrides the search argument set by the TBSARG service for this search only. If the name-list parameter is omitted, a search argument must have been established by a previous TBSARG command. Otherwise, a severe error occurs. See "Invoking the ISPF Services" on page 2 for specification of name lists.

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

**rowid-name**
Specifies the name of a variable in which a number that uniquely identifies the row being accessed is to be stored. Later, this identifier can be specified in the ROW parameter of TBSKIP to cause the CRP to be positioned to the row. This identifier is not saved on permanent storage by TBSAVE or TBCLOSE.

**NEXT**
Specifies that the scan is to proceed from the row following the current row to the bottom of the table. This is the default.

**PREVIOUS**
Specifies that the scan is to proceed from the row preceding the current row to the top of the table. To scan the bottom row, CRP must be positioned to TOP.

**NOREAD**
Specifies that the variables contained in the requested row not be read into the variable pool.

**crp-name**
Specifies the name of a variable in which the row number pointed to by the CRP is to be stored. If the CRP is positioned to TOP, the row number returned to zero.

**condition-value-list**
Specifies a list of values for determining when the scan should end. Each condition-value relates to a search argument for a column or extension variable in the table as specified in the ARGLIST parameter. This parameter is ignored if no ARGLIST parameter is specified. The operators specified in the condition-list correspond one-to-one with the names in the ARGLIST. If there are extra operators, a severe error condition is encountered.

The name-list and condition-value-list syntax is:

```
ARGLIST(name1,name2, ....)
```

CONDLIST(condition1, condition2, ...) The valid condition-values are EQ, NE, LE, LT, GE, and GT. If there are fewer condition-values than search arguments the default is EQ for those columns. Each argument and its associated operator are treated as separate entities, and not as subfields of a single argument.

The condition-values LE, LT, GE, and GT can have a date indicator immediately following them. The date indicator is Y*n*, where `Y` indicates that the variable name associated with the condition-value is a date, and *n* is an integer between 1 and 7 indicating the offset within the variable value where the year begins. The year should be a 2-digit value because a century value is inserted in front of the 2-digit year for comparison purposes.

The following meanings are associated with the condition-values:

**EQ** Specifies that the scan is to end when an equal condition exists between the argument value and the row value. This is the default.

**NE** Specifies that the scan is to end when the row value is not equal to the argument value.

**LE** Specifies that the scan is to end when the row value is less than or equal to the argument value.

**LT** Specifies that the scan is to end when the row value is less than the argument value.

**GE** Specifies that the scan is to end when the row value is greater than or equal to the argument value.

**GT** Specifies that the scan is to end when the row value is greater than the argument value.

**Yn** Can be used with LE, LT, GE, and GT. It must immediately follow one of the four allowed condition-values. The `Y` indicates that the paired variable name is a date variable that needs a century value added to a 2-digit year so that dates can be compared correctly. The *n* is a number from 1 to 7 that gives the offset within the variable value where the year is located.

**buf-len**
Specifies a fullword fixed binary integer containing the length of "buffer."

**buffer**
Specifies a buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC call for a command procedure.

## Return Codes

The following return codes are possible:

**0** Normal completion.

**8** Row does not exist, no match was found; CRP is set to TOP (zero). The rowid remains unchanged.

**12** Table is not open.

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

**20** Severe error.

## Example 1

For the table TELBOOK:

Move table TELBOOK's CRP to the row that fulfills the search argument as specified in a preceding TBSARG operation. For an example of TBSARG, see the example in the TBSARG description in this chapter. Copy values from variables in that row to function pool variables whose names match those of the table variables.

```
ISPEXEC TBSCAN TELBOOK
```

Set the program variable BUFFER to contain:

```
TBSCAN TELBOOK
```

Set program variable BUFLEN to the length of the variable BUFFER. Issue the following command:

```
CALL ISPEXEC (BUFLEN, BUFFER);
```

or alternately

```
CALL ISPLINK ('TBSCAN  ','TELBOOK ');
```

## Example 2

For the table TELBOOK:

Use the TBSCAN service to position the CRP of table TELBOOK to the row containing the name JOHNSON in variable LNAME, and the zip code 08007 in variable ZIPCODE. Copy values of the variables in that row to function pool variables whose names match those of the table variables.

- Set function pool variable LNAME to JOHNSON.
- Set function pool variable ZIPCODE to 08007.
- Issue the following request:

```
ISPEXEC TBSCAN TELBOOK ARGLIST(LNAME,ZIPCODE)
```

Set the program variable BUFFER to contain:

```
TBSCAN TELBOOK ARGLIST(LNAME,ZIPCODE)
```

Set program variable BUFLEN to the length of the variable BUFFER. Issue the following:

```
CALL ISPEXEC (BUFLEN, BUFFER);
```

or alternately

```
CALL ISPLINK ('TBSCAN  ','TELBOOK ','(LNAME,ZIPCODE)');
```

If the return code is 0, the row was found and values were copied from the row variables to function pool variables.

## Example 3

Establish a search argument to be used by a TBSCAN operation of the table DATETBL. Assume DATE1 to be a name variable in table DATETBL and that the dates are in a yy/mm/dd format. Specify a scan direction of forward and terminate the scan when the row value of DATE1 is greater than 99/01/31.

- Invoke TBVCLEAR for table DATETBL
- Set variable DATE1 to 99/01/31

- Issue the following TBSCAN request:

```
ISPEXEC TBSCAN DATETBL NEXT NAMECOND(DATE1,GTY1)
```

# TBSKIP – Move the Row Pointer

The TBSKIP service moves the current row pointer (CRP) of a table forward or backward by a specified number of rows and retrieves the row to which it is pointing unless the NOREAD parameter is specified.

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

## Command Invocation Format

```
ISPEXEC  TBSKIP  table-name  [NUMBER(number)]
                             [SAVENAME(var-name)]
                             [ROWID(rowid-name)]
                             [ROW(rowid)]
                             [NOREAD]
                             [POSITION(crp-name)]
```

## Call Invocation Format

```
CALL  ISPEXEC (buf-len, buffer);

OR

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

## Parameters

**table-name**
Specifies the name of the table to be used.

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

For a call, this parameter must be a fullword fixed binary number.

A default skip of +1 exists if the ROW and NUMBER parameters are both omitted. When the ROW parameter is specified, no default skip of +1 is assumed if the NUMBER parameter is omitted.

**var-name**
Specifies the name of a variable into which a list of extension variable names contained in the row is stored. The list is enclosed in parentheses, and the names within the list are separated by a blank.

**rowid-name**
Specifies the name of a variable in which a number that uniquely identifies the row being accessed is to be stored. Later, this identifier can be specified in the ROW parameter to cause the CRP to be positioned to the row. This identifier is not saved on permanent storage by TBSAVE or TBCLOSE.

**rowid**
Specifies the numeric value that uniquely identifies the row to be accessed. This value is obtained by using the ROWID(rowid-name) parameter.

A default skip of +1 exists if the ROW and NUMBER parameters are both omitted. When the ROW parameter is specified, no default skip of +1 is assumed if the NUMBER parameter is omitted.

**NOREAD**
Specifies that the variables contained in the requested row not be read into the variable pool.

**crp-name**
Specifies the name of a variable in which the row number pointed to by the CRP is to be stored. If the CRP is positioned to TOP, the row number returned is zero.

**buf-len**
Specifies a fullword fixed binary integer containing the length of "buffer."

**buffer**
Specifies a buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC call for a command procedure.

## Return Codes

The following return codes are possible:

**0** Normal completion.

**8** CRP would have gone beyond the number of rows in the table. This includes a table empty condition, with CRP set to TOP (zero). The rowid remains unchanged.

**12** Table is not open.

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

**20** Severe error.

## Example

In the table TELBOOK, move the current row pointer (CRP) to the next row. After the move, copy values from variables in that row to variables in the function variable pool having names that are the same as the names of the variables in the row.

```
ISPEXEC TBSKIP TELBOOK
```

Set the program variable BUFFER to contain:

```
TBSKIP TELBOOK
```

Set program variable BUFLEN to the length of the variable BUFFER. Issue the following command:

```
CALL ISPEXEC (BUFLEN, BUFFER);
```

or alternately

```
CALL ISPLINK ('TBSKIP  ','TELBOOK ');
```

# TBSORT – Sort a Table

The TBSORT service places the rows of an open table in a user-specified order and stores this specified order in a sort information record.

The sort can be done on more than one field and in either an ascending or descending order. TBSORT can be issued for an empty table. When a TBSORT is completed, the CRP is set to zero (top).

The sort can also be done by date without having to change the date variable to a 4-digit year. The PDF configuration table field YEAR2000 is used to determine a century value to be appended to the existing 2-digit year values within the ISPF table. The variable is only modified internally for comparison purposes and no actual change is made to data stored in the ISPF table.

The sort information record is maintained with the table. This record contains the order of the "last-sort" and provides for rows to be added to the table in the proper sequence after a sort has been completed. This is done through the ORDER keyword on the TBADD, TBMOD, and TBPUT services. The sort information record is saved on external storage when a TBSAVE or TBCLOSE operation successfully completes. It is retrieved during TBOPEN processing.

## Command Invocation Format

```
ISPEXEC  TBSORT  table-name  FIELDS(sort-list)
```

## Call Invocation Format

```
CALL  ISPEXEC (buf-len, buffer);
```

*OR*

```
CALL  ISPLINK ('TBSORT  ', table-name, sort-list);
```

## Parameters

**table-name**
Specifies the name of the table to be sorted.

**sort-list**
Specifies sort fields. The syntax is as follows:

```
(field-name1, B|C|N|Yn, A|D,field-name2, B|C|N|Yn, A|D, ...)
```

Each sort field-name must be either a KEY or NAME field. The first (left most) field-name is the primary key (most significant) and the rows are then collated based on the values of the field-names.

The field-name is followed by a sort field type designator. The sort field type designator can have a value of 'C' for a character sort, a value of 'N' for a numeric sort, a value of 'B' for a binary sort, or a value of 'Yn' for a year sort. For English, where sorting is in EBCDIC sequence, specifying either C or B as the sort field type designator causes the same sort order. For other languages, where the character format can be other than EBCDIC, only B is to be specified for a binary sort.

The 'Yn' sort is treated as a character sort where the variable being sorted is a date variable, with *n* being the offset of the beginning of a 2–byte year in the variable. Internally within ISPF, the year is expanded to a 4–byte year using the PDF configuration field **YEAR2000** for calculating the century value.

The collating sequence for character sorts during DBCS and English sessions is in EBCDIC order. This means, for example, that all lowercase letters precede uppercase letters when sorting in an ascending sequence. For other languages, a character sort is done such that both uppercase and lowercase, as well as accented and non-accented versions of a letter, are sorted in the proper order.

The sort field type designator is followed by a sort sequence direction value. The sort sequence direction value can be either 'A' (ascending) or 'D' (descending). The field type designator and the sort sequence direction can be omitted for the last named field only. They default to 'C' (character) and 'A' (ascending), respectively.

In some non-English languages, the comma is used in place of a decimal point. To accommodate various language usages, three numeric representations are supported: period, comma, and French representations.

*Table 4. Decimal Point Representations*

| Convention | Example | Example | Where Used |
|---|---|---|---|
| Period | 1,234.56 | 0.789 | Japan, Mexico, UK, USA |
| Comma | 1.234,56 | 0,789 | Most other countries |
| French | 1234,56 | 0,789 | France, South Africa |

The TBSORT service accommodates these three numeric representations. The convention used is determined by the language of the session, specified by the value of ZLANG in the system profile table. The current English version accepts only the period, treating it as the delimiter of the whole and decimal portion of a number. Sorting is based on the specified language convention.

The following restrictions apply to fields for a "numeric" type sort:

1. The field must be a decimal number and optionally can contain a plus (+) or minus (-) sign. The decimal number can be either a whole number (for example, 234) or a mixed number (for example, 234.56), composed of a whole number followed by a decimal point. A decimal point is not required after a whole number, but is required in a mixed number. (Under the period convention, the decimal point is represented by a period (.); under the comma or French conventions, the decimal point is represented by a comma (,).) No other characters are allowed except leading blanks.

2. No numeric string can exceed 16 characters. This length value includes any plus or minus sign, any blanks, or a decimal point.
3. The largest value that can be sorted is plus or minus 2147483647.
4. The string can have leading blanks following the sign character.

**buf-len**
Specifies a fullword fixed binary integer containing the length of "buffer."

**buffer**
Specifies a buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC call for a command procedure.

## Return Codes

The following return codes are possible:

**0** Normal completion.

**12** Table is not open.

**16** Numeric convert error.

**20** Severe error.

**Notes on Performance:**

1. The performance of TBSORT is not greatly affected by the starting order of the table. However, a sort by year can affect performance because an internal conversion to a 4-digit year must be done for each comparison.
2. A numeric sort affects performance because a conversion of two numbers must be done for each comparison.

## Example 1

Perform a sort on the LASTNAME field for table TELBOOK. Use the defaults of "A" (ascending) and "C" (character).

```
ISPEXEC TBSORT TELBOOK FIELDS(LASTNAME)
```

Set the program variable BUFFER to contain:

```
TBSORT TELBOOK FIELDS(LASTNAME)
```

Set program variable BUFLEN to the length of the variable BUFFER. Issue the following command:

```
CALL ISPEXEC (BUFLEN, BUFFER);
CALL ISPLINK ('TBSORT  ','TELBOOK ','LASTNAME');
```

## Example 2

Perform a sort on table MODSIZES. Sort on the field NAME, a character field, in ascending sequence. Then sort on the field SIZE field, a numeric field, in descending sequence.

```
ISPEXEC TBSORT MODSIZES FIELDS(NAME,C,A,SIZE,N,D)
```

Set the program variable BUFFER to contain:

```
TBSORT MODSIZES FIELDS(NAME,C,A,SIZE,N,D)
```

Set program variable BUFLEN to the length of the variable BUFFER. Issue the following command:

```
CALL ISPEXEC (BUFLEN, BUFFER);
```

or alternately

```
CALL ISPLINK ('TBSORT  ','MODSIZES','(NAME,C,A,SIZE,N,D)');
```

# TBSTATS – Retrieve Table Statistics

The TBSTATS service obtains statistical information for a table and saves the information in variables specified in the service request.

Table statistics are maintained with each physical table member stored on permanent storage. The TBSTATS service provides access to these statistics from a dialog. The TBSTATS service also provides status information regarding the current usage of a specified table.

The statistics for a given table are available whether the table is open or closed. The statistics reflect the table as it exists on the input table file, except when the table is open in the logical screen where the TBSTATS service is issued. The statistics then reflect the version of the table that is currently open.

The existence of a table can be checked by the value in the STATUS1 field. If the table does not exist, no other processing takes place.

The following statistical information is available:
- Date the table was created
- Time the table was created
- Date of last update
- Time of last update
- Last user to update the table
- Number of rows when the table was created
- Current number of rows (zero if the table is empty)
- Number of existing rows that have been updated
- Number of times the table has been updated
- Last table service issued for the table. The table must be open.
- Return code associated with the last table service. The table must be open.
- Whether the table is available for WRITE mode processing
- Whether the table exists in the table input file chain
- Whether the table is open for this logical screen.

For statistical purposes, two table processes have been defined. The “create process” is defined as beginning with the TBCREATE and ending with a TBCLOSE or TBEND. The “update process” is defined as beginning with the TBOPEN and ending with a TBCLOSE or TBEND.

## Command Invocation Format

```
ISPEXEC  TBSTATS  table-name  [CDATE(date-created-name)]

                              [CTIME(time-created-name)]

                              [UDATE(date-updated-name)]

                              [UTIME(time-updated-name)]

                              [USER(user-name)]

                              [ROWCREAT(row-created-name)]

                              [ROWCURR(rownum-name)]

                              [ROWUPD(row-updated-name)]
```

```
                                    [TABLEUPD(table-updated-name)]

                                    [SERVICE(service-name)]

                                    [RETCODE(return-code-name)]

                                    [STATUS1(status1-name)]

                                    [STATUS2(status2-name)]

                                    [STATUS3(status3-name)]

                                    [LIBRARY(library)]
                                    [CDATE4D(date-created-name-4-digit)]
                                    [UDATE4D(date-updated-name-4-digit)]
```

## Call Invocation Format

```
CALL  ISPEXEC (buf-len, buffer);

OR

CALL  ISPLINK ('TBSTATS ', table-name  [,date-created-name]

                                       [,time-created-name]

                                       [,date-updated-name]

                                       [,time-updated-name]



                                       [,user-name]

                                       [,row-created-name]

                                       [,rownum-name]

                                       [,row-updated-name]

                                       [,table-updated-name]

                                       [,service-name]

                                       [,return-code-name]

                                       [,status1-name]

                                       [,status2-name]

                                       [,status3-name]

                                       [,library])
                                       [,date-created-name-4-digit]
                                       [,date-updated-name-4-digit];
```

## Parameters

**table-name**
Specifies the name of the table for which statistical information is to be obtained.

**date-created-name**
Specifies the name of a variable where the date the table was created is to be stored. The date is returned in the form YY/MM/DD.

**time-created-name**
Specifies the name of a variable where the time the table was created is to be stored. The time is returned in the form HH.MM.SS.

**date-updated-name**
Specifies the name of a variable where the date the table was last updated is to be stored. The date is returned in the form YY/MM/DD.

**time-updated-name**
Specifies the name of a variable where the time the table was last updated is to be stored. The time is returned in the form HH.MM.SS.

**user-name**
Specifies the name of a variable where the userid of the user that created or last updated the table is to be stored.

**row-created-name**
Specifies the name of a variable where the number of rows that existed at the end of the "create process" is to be stored.

**rownum-name**
Specifies the name of a variable where the number of rows contained in the table is to be stored.

**row-updated-name**
Specifies the name of a variable where the number of updated rows is to be stored. This is the number of existing rows that have been updated by TBPUT or TBMOD. During the "update process," rows that are added to the table are included in this number. Any row that increments this number, when deleted, will decrement this number.

**table-updated-name**
Specifies the name of a variable where the number of times this table has been updated is to be stored. This is the number of "update processes" that have occurred in which at least one row has been updated.

**service-name**
Specifies the name of a variable where the last table services command issued for this table is to be stored. This value is returned only if the table is currently open for the same logical screen.

**return-code-name**
Specifies the name of a variable where the return code associated with the last table services command issued for this table is to be stored. This value is returned only if the table is currently open to the same logical screen.

**status1-name**
Specifies the name of a variable where the status of the table in the table input library chain is to be stored. Values that can be stored and their meanings are:

1 = table exists in the table input library chain
2 = table does not exist in the table input library chain
3 = table input library is not allocated

.

**status2-name**
Specifies the name of a variable where the status of the table in this logical screen is to be stored. Values that can be stored and their meanings are:

1 = table is not open in this logical screen
2 = table is open in NOWRITE mode in this logical screen
3 = table is open in WRITE mode in this logical screen
4 = table is open in SHARED NOWRITE mode in this logical screen
5 = table is open in SHARED WRITE mode in this logical screen.

**status3-name**
Specifies the name of a variable where the availability of the table to be used in WRITE mode is to be stored. Values that can be stored and their meanings are:

1 = table is available for WRITE mode
2 = table is not available for WRITE mode.

**library**
Specifies the ddname of a FILEDEF command or the lib-type of the LIBDEF service request that defines an optional input file definition and provides control for the table input source. If omitted, the default is ISPTLIB.

**date-created-name-4–digit**
Specifies the name of a variable where the date the table was created is to be stored. The date is returned in the form YYYY/MM/DD.

**date-updated-name-4–digit**
Specifies the name of a variable where the date the table was last updated is to be stored. The date is returned in the form YYYY/MM/DD.

**buf-len**
Specifies a fullword fixed binary integer containing the length of "buffer."

**buffer**
Specifies a buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC call for a command procedure.

## Return Codes

The following return codes are possible:

**0** Normal completion (returned even if the table does not exist).

**16** Variable value has been truncated.

**20** Severe error.

## Example

Determine the date when the table TELBOOK was created and when it was last updated.

```
ISPEXEC TBSTATS TELBOOK CDATE(DATE1) UDATE(DATE2)
```

Set the program variable BUFFER to contain:

```
TBSTATS  TELBOOK  CDATE(DATE1)  UDATE(DATE2)
```

Set program variable BUFLEN to the length of the variable BUFFER. Issue the following:

```
CALL ISPEXEC (BUFLEN, BUFFER);
```

or alternately

```
CALL ISPLINK ('TBSTATS ','TELBOOK ','DATE1 ','DATE2 ');
```

# TBTOP – Set the Row Pointer to the Top

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

## Command Invocation Format

```
ISPEXEC  TBTOP  table-name
```

## Call Invocation Format

```
CALL  ISPEXEC (buf-len, buffer);
```

*OR*

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

## Parameters

**table-name**
Specifies the name of the table to be used.

**buf-len**
Specifies a fullword fixed binary integer containing the length of "buffer."

**buffer**
Specifies a buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC call for a command procedure.

## Return Codes

The following return codes are possible:

**0** Normal completion.

**12** Table is not open.

**20** Severe error.

## Example

For the table TELBOOK, move the current row pointer (CRP) to the row immediately before its first row.

```
ISPEXEC TBTOP TELBOOK
```

Set the program variable BUFFER to contain:

```
TBTOP TELBOOK
```

Set program variable BUFLEN to the length of the variable BUFFER. Issue the following command:

```
CALL ISPEXEC (BUFLEN, BUFFER);
```

or alternately

```
CALL ISPLINK ('TBTOP   ','TELBOOK ');
```

# TBVCLEAR – Clear Table Variables

The TBVCLEAR service sets dialog variables to nulls.

All dialog variables that correspond to columns in the table, specified when the table was created, are cleared.

The contents of the table and the position of the current row pointer (CRP) are not changed by this service.

## Command Invocation Format

```
ISPEXEC  TBVCLEAR  table-name
```

## Call Invocation Format

```
CALL  ISPEXEC (buf-len, buffer);
```

*OR*

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

## Parameters

**table-name**
Specifies the name of the table to be used.

**buf-len**
Specifies a fullword fixed binary integer containing the length of "buffer."

**buffer**
Specifies a buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC call for a command procedure.

## Return Codes

The following return codes are possible:

**0** Normal completion.

**12** Table is not open.

**20** Severe error.

## Example

Clear dialog variables associated with the table TELBOOK.

```
ISPEXEC TBVCLEAR TELBOOK
```

Set the program variable BUFFER to contain:

```
TBVCLEAR TELBOOK
```

Set program variable BUFLEN to the length of the variable BUFFER. Issue the following command:

```
CALL ISPEXEC (BUFLEN, BUFFER);
```

or alternately

```
CALL ISPLINK ('TBVCLEAR','TELBOOK ');
```

# TRANS – Translate CCSID Data

The TRANS dialog service translates data from one Coded Character Set Identifier (CCSID) to another. A maximum variable size of 32 767 bytes of data can be translated. There is no automatic transformation of single-byte to double-byte data or double-byte to single-byte data. This service is available through the ISPEXEC and ISPLINK interfaces. See the *ISPF Dialog Developer's Guide and Reference*.

## Command Invocation Format

```
ISPEXEC  TRANS  FRMCCSID(from-ccsid-number)   TOCCSID(to-ccsid-number)

                FROMVAR(from-variable-name)  [TOVAR(to-variable-name)]

               [LENGTH(data-length)]
```

## Call Invocation Format

```
CALL ISPEXEC (buflen, buffer)
```

*OR*

```
CALL ISPLINK ('TRANS   ',from-ccsid-number,to-ccsid-number,from-variable-name
                                  [,to-variable-name]
                                  [,data-length]);
```

## Parameters

**from-ccsid-number**
Required parameter. The from-ccsid-number is a 5-digit decimal (5 character position) number that specifies the current CCSID of the variable data before translation.

**to-ccsid-number**
Required parameter. The to-ccsid-number is a 5-digit decimal (5 character position) number that specifies the CCSID the variable data will be translated to.

**from-variable-name**
Required parameter. Specifies the name of a dialog variable that contains the source data to be translated. The translated data is returned in this variable if the TOVAR parameter is omitted.

**to-variable-name**
Optional parameter. Specifies the name of a dialog variable that receives the translated data. A truncation error occurs if this variable is not large enough to hold the translated data. Only the translated data is stored in this variable. The translated data is returned in the dialog variable identified by the FROMVAR parameter if this parameter is omitted.

**data-length**
Optional parameter. The length of data in the source variable that is translated. This number must be an integer from 0 to 32 767. A zero value results in this parameter being ignored. For call invocation, this parameter must be a fullword fixed binary number. If this parameter is specified, the smaller of its value and the length of source variable data is used. If this parameter is omitted, the length of the source variable data determines the amount of data that is translated. Only the translated data is stored in the receiving variable.

## Return codes

**0** Service completed successfully.

**4** Translate tables do not support the requested "to ... from" combination. For a list of extended code page translate tables provided by ISPF, see the *ISPF Dialog Developer's Guide and Reference*.

**8** From variable not found.

**16** Variable services indicated a translation error or truncation occurred storing the translated data.

**20** Severe error.

# VCOPY – Create a Copy of a Variable

This service is used only with CALL ISPLINK or CALL ISPLNK calls.

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

The variable names can be specified as a single 8-character value, a list enclosed in parentheses, or a name-list structure. In LOCATE mode an array of pointers must be supplied to receive the data address. An array of lengths must be supplied to receive the data lengths.

In locate mode, the VCOPY service automatically allocates storage for the data, and returns the address and length of each variable to the caller.

In move mode, an array of lengths must be supplied on input. Its values map the structured area which must be supplied to receive the data. The caller first allocates storage for the data, and then invokes VCOPY, passing the address and length of the storage area into which the data is to be copied. The length array is then set with the data lengths.

When a variable has been masked and is accessed by VCOPY, the output string will contain the mask characters. When specifying the length to receive these variables on the VCOPY call, the length should be as long as the mask, not the defined variable. See "VMASK – Mask and Edit Processing" on page 294 for a full description of the VMASK service.

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

## Command Invocation Format

```
ISPEXEC  *This service does not apply to APL2 or command
          procedures*
```

## Call Invocation Format

```
CALL  ISPEXEC  *This service cannot be used with this interface*
```

*OR*

```
CALL  ISPLINK ('VCOPY   ', name-list, length-array, value-array
                  [,'LOCATE  '|'MOVE    '] );
```

## Parameters

**name-list**
Specifies an area containing the names of dialog variables to be copied. The standard name-list format is used.

**length-array**
Specifies an array of fullword fields containing the lengths of the data areas for the dialog variable values. The array can consist of a single item. In move mode, each element of the array is set by the caller to the output area size. In move or locate mode, each element of the array is set by the service to the number of bytes of data for the corresponding variable. The length does not include trailing blanks unless the variable is defined to maintain blanks. For example:

- VCOPYing a variable that was defined using VDEFINE with the NOBSCAN option
- VCOPYing a REXX variable that was explicitly set with trailing blanks and then VPUT to the SHARED or PROFILE pool.

**value-array**
In locate mode, specifies the name of an array that contains pointers to fields into which the copied variables are placed. The array can consist of a single item. In move mode, specifies the name of a structure that is mapped by the length array.

**LOCATE**
Specifies locate mode. The address of the copied value is returned to the user invoking the service. This is the default mode.

**MOVE**
Specifies move mode. The copied value is returned to the user invoking the service.

## Return Codes

The following return codes are possible:

**0** Normal completion.

**8** One or more variables do not exist.

**12** Validation failed.

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

**20** Severe error.

## Example

Copy the value in dialog variable QROW to a field named QROWSDATA in this PL/I program module. Perform the copy in move mode, as opposed to locate mode. Variable L8 contains a value of 8.

```
CALL ISPLINK ('VCOPY   ','QROW ',L8,QROWDATA,'MOVE    ');
```

# VDEFINE – Define Function Variables

The VDEFINE service is used only with CALL ISPLINK or CALL ISPLNK calls.

The VDEFINE service is invoked by a program to give ISPF the ability to use dialog variable names to directly access variables within the particular program module. In the call to VDEFINE, the program specifies the format (character string, fixed binary, bit string, hex, float, pack(n), binstr, DBCS, or user-defined) and length of the variables. Stacking of definitions for a particular dialog variable can be achieved by successive calls to VDEFINE for that dialog variable.

When the VDEFINE service is called, ISPF enters the dialog variable names into the function pool for the dialog function currently in control. Dialog variables entered in the function pool by use of the VDEFINE service are called defined variables to distinguish them from implicit variables in the function pool.

A list of dialog variables can be defined with a single call to the VDEFINE service. The program variables that correspond to the dialog variables defined to ISPF by VDEFINE must be in contiguous locations in storage or defined as an array or structure within the program. Also, unless you specify LIST as an option in the options list referred to by the service request, all variables must have the same format and length. The program variable name passed to ISPF must be the name of the first variable as defined in the program, the name of the array, or the name of the structure.

When the LIST option is used, programs can VDEFINE only selected application variables in a dialog application structure. This is accomplished by specifying an asterisk (*) as a placeholder in the name-list and in the corresponding position in the format definition array for those portions of dialog application storage that are not to be considered by VDEFINE. The * place-holder (in the name-list and the format) allows ISPF to determine the address of the dialog application storage of the next true variable name in the name-list. This is determined by the corresponding length in the length array parameter.

Before issuing the VDEFINE service request (with the LIST parameter specified) the function must create two arrays to specify the formats and lengths of the variables to be defined. The first array defines, in sequence, the format (character string, fixed binary, and so forth) of each variable. The second array defines, in sequence, the length (in bytes) of each variable. Variable names in the name-list that you specify on the VDEFINE request must be in the same relative positions as the corresponding format and length definitions in the arrays.

## Command Invocation Format

```
ISPEXEC  *This service does not apply to APL2 or command
          procedures*
```

## Call Invocation Format

```
CALL  ISPEXEC  *This service cannot be used with this interface*

OR

CALL  ISPLINK ('VDEFINE ', name-list, variable, format, length
                                              [,options-list]
                                              [,user-data] [,'LFORMAT']);
```

## Parameters

**name-list**
Specifies the symbolic name or name-list to be used by ISPF when referencing the specified variables.

An asterisk, in conjunction with the USER format keyword described below, specifies that the exit routine, whose address is specified in the user-data parameter, is to be called for variables not found in the function pool.

An asterisk (*) in the name-list, in conjunction with an asterisk in the format parameter described below, specifies that the storage represented by the corresponding physical length in the length parameter is to be skipped when calculating the address of the next name in the name-list. When this facility is used, LIST must be specified in the options-list parameter.

**variable**
Specifies the variable whose storage is to be used. If a name list is passed, this storage contains an array of variables. The number of names in the list determines the dimension of the array.

When LIST is specified for options-list, this parameter is the name of a variable or structure whose storage is used for dialog variables in the name list. This storage is assigned to dialog variables in the order that they appear in the name-list, and according to the length array mapping.

**format**
Specifies the data conversion format.

When LIST is specified for options-list, this parameter is the name of an array of CHAR(8) fields, one for each variable in the name-list. Each element of this array defines the data format of the variable in the corresponding position in the name-list. Entries must be left-justified and padded with blanks. There must be at least as many array elements as there are names in the name-list. You can use an asterisk in the format list to have application storage not be considered by VDEFINE. See the previous discussion under the name-list parameter.

Valid formats are:

**BINSTR — Binary String**
ISPF provides the binary string data format to support dialog applications written in the C language. When a variable defined as BINSTR is updated in the function pool, ISPF pads with binary zeros. This is desirable within C function programs, because the C language uses binary zeros to mark the end of a character string.

In updating this type of variable, ISPF stores only up to "length – 1" amount of significant data and places a null terminator in the last position. Because the updated data contains the binary zero, the length of the variable should be greater than 1.

**BIT**
Bit string, represented by the characters 0 or 1. Within the variable, the data is left-justified and padded on the right with binary zeros. For these variables, a null value is stored as binary zeros and cannot be distinguished from a zero value.

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

No data conversion is performed when fetching and storing a CHAR variable, nor is there any checking for valid characters. In PL/I, a character string to be used as a dialog variable must be declared as fixed length, because VDEFINE cannot distinguish variable-length PL/I strings.

**DBCS**
DBCS string. Within the variable, the data is left-justified and padded on the right with blanks. The variable must not contain shift-out or shift-in characters and it must be even in length.

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

**FIXED**
Fixed binary integer, represented by the characters 0–9.

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

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

**FLOAT — Floating Point**
The floating point data format is used for variables consisting of numeric values stored in characteristic/mantissa form.

Format type FLOAT dialog variables are displayed (and stored in the shared and profile pool) in character representation with the decimal separator.

Floating point numbers are processed as real numbers. A single-precision number is processed as a 32-bit real number and can have 7 or 8 significant digits. A double-precision number is processed as a 64-bit real and can have 13 or 14 significant digits. For single-precision floating point numbers, up to 7 digits is displayed as a real number. Any number greater than 7 digits is represented in exponential notation.

For example, for short floating point numbers,

| VALUE | REPRESENTATION |
|---|---|
| **1234567** | 1234567 |
| **12345678** | $1.234568E^{+07}$ |
| **123.4567** | 123.4567 |
| **123.45678** | 123.4568 |

For double-precision floating point numbers, the limit is 13 digits.

The length that you specify for this type must equal the total number of bytes of program storage that the variable uses. FLOAT variables can have a length of 4 or 8 bytes. A FLOAT variable defined with a length of 4 bytes is considered single precision and with 8 bytes is considered double precision.

The magnitude (M) of a floating point number supported by ISPF is:

$$5.4 \times 10^{-79} \leq M \leq 7.2 \times 10^{+75}$$

ISPF converts floating point numbers between the real number and character formats. Because of this conversion, rounding is not predictable for single precision numbers when the digit being rounded is a 5.

**HEX**
Bit string, represented by the characters 0–9 and A–F. Within the variable, the data is left-justified and padded on the right with binary zeros. For these variables, a null value is stored as binary zeros and cannot be distinguished from a zero value.

**PACK | PACK(n) — Packed Decimal**
The packed decimal data format provides support for COBOL and corresponds to a COBOL COMP-3 data type. Packed decimal variables consist of right-justified numeric values stored such that each decimal digit takes up one-half byte. All bytes contain 2 decimal digits, except for the last byte in the variable. The last byte consists of the least significant decimal digit followed by a representation of the sign. The maximum number of digits in a PACKed variable is 18 as specified by ANSI COBOL standard. This results in the number of digits always being an odd number.

The valid values to represent the sign are the hexadecimal digits C for positive and D for negative. If the sign is any other hexadecimal digit, the value is considered to be unsigned.

The length that you specify for this type must equal the total number of bytes of program storage that the variable uses. PACK variables can have a length of 1–10 bytes.

When you define a variable as having a PACK(n) data format, n specifies the number of digits to appear to the right of the assumed decimal point. For example, the value of a variable when defined is 12345. The assumed decimal position would occur before the 1 if defined as PACK(5), after the 1 if defined as PACK(4), after the 2 if defined as PACK(3), and so on. PACK without (n) specified is equivalent to PACK(0).

Variables defined as PACK or PACK(n) are converted to character representation when retrieved from the function pool. If the variable is defined as PACK(n), and n is greater than zero, the converted number will contain a decimal separator followed by n digits after the assumed decimal point.

When a variable defined as PACK(n) is updated in the function pool, ISPF will pad the variable with zeros or truncate on either end to ensure the variable contains the correct number of digits to the right of the assumed decimal separator.

The value of n must be in the range 0–18.

**USER**
Specifies that the format is to be determined by the user. Any conversion format is allowed. A conversion routine must be specified and is specified by naming it in the user-data parameter, described below.

**length**
Specifies the length of the variable storage, in bytes. This parameter must be a fullword fixed binary integer. The maximum length for a FIXED variable is 4 bytes, for PACK(n) variables is 10 bytes, and for FLOAT variables is 8 bytes. The maximum length for other types of variables is 32 767 bytes.

For character variables in a C program, this length should be one less than the length of the program variable. This allows for the null terminator at the end of the string. Always initialize variables for the length specified in this parameter, unless you are using the BINSTR parameter.

When LIST is specified as an option in the options-list, this parameter is the name of an array of fullword fixed binary integers. Each element of this array defines the data length of the variable in the corresponding position of the name-list. There must be at least as many array elements as there are names in the name-list.

**options-list**
Specifies initialization of the defined storage and/or retention of trailing blanks in variable data. The options-list parameters are COPY, NOBSCAN, and LIST. They are specified in the name-list format.

**Note:** Option-list parameters cannot be specified if the USER format keyword and a name-list of asterisk (*) have been selected.

**COPY**
Specifies that any dialog variable with the same name can be used to initialize the defined storage. The variable pools are searched in the standard function pool, shared pool, profile pool sequence.

**NOBSCAN**
Specifies that any trailing blanks in the variables are to remain in the variables.

**LIST**
Specifies that the variables in the name-list to be defined to ISPF are of varying formats (format array) and lengths (length array).

When the LIST option is used, programs can VDEFINE only selected application variables in a dialog application structure. This is accomplished by specifying an asterisk (*) as a placeholder in the name-list and in the corresponding position in the format definition array for those portions of dialog application storage that are not to be considered by VDEFINE. The asterisk place-holder (in the name-list and the format) allows ISPF to determine the address of the dialog application storage of the next true variable name in the name-list. This is determined by the corresponding length in the length array parameter.

**user-data**
Specifies the storage location that contains the entry point address of the conversion subroutine followed by any other data that should be passed to the routine.

The exit is given control in 31-bit mode if either the VDEFINE dialog service is invoked in 31-bit mode or the high-order bit of the user-exit address is on as specified for the VDEFINE service. The high-order bit contains the AMODE and the remainder of the word contains the address. If bit 0 contains 1, the exit routine is given control in 31-bit addressing mode.

This parameter is specified whenever the USER parameter is specified.

**LFORMAT**
Indicates the specified name-list variables all have the same format.

## Return Codes

The following return codes are possible:

**0** Normal completion.

**8** Variable not found.

**16** Data truncation occurred.

**20** Severe error.

## Example 1

Establish ISPF accessibility, using the name MSGNAME, to a field named ERRMSG in this PL/I module. The field is a character string and is 8 bytes long. Program variable L8 contains a value of 8.

```
CALL ISPLINK ('VDEFINE ','(MSGNAME)',ERRMSG,'CHAR    ',L8);
```

## Example 2

Define three variables (FVAR, CVAR, and DVAR) with data formats of FIXED, CHAR, and DBCS, and with lengths of 4, 5, and 20, respectively.

```
DECLARE

 1 VARS,
   3 FVAR     FIXED BIN(31),
   3 CVAR     CHAR(5),
   3 DVAR     CHAR(20),
 FARR(3) CHAR(8),
 LARR(3) FIXED BIN(31);

FARR(1) = 'FIXED';
FARR(2) = 'CHAR';
FARR(3) = 'DBCS';
LARR(1) = 4;
LARR(2) = 5;
LARR(3) = 20;

CALL ISPLINK ('VDEFINE ','(FVAR CVAR DVAR)',
               VARS,FARR,LARR,'LIST    ');
```

## Example 3

Define two dialog variables, VAR1 and VAR2, contained in a structure. The name of the structure is STRCVARS and contains other data that is not used.

```
DECLARE

      1 STRCVARS,
         3 VAR1 FIXED BIN(31),
         3 FILLER CHAR(4),
         3 VAR2 CHAR(8)
      FARR(3) CHAR(8),
      LARR(3) FIXED BIN(31);

  FARR(1) = 'FIXED   ';
  FARR(2) = '*';
  FARR(3) = 'CHAR    ';
  LARR(1) = 4;
  LARR(2) = 4;
```

```
  LARR(3) = 8;

  CALL ISPLINK('VDEFINE ','(VAR1 * VAR2)',
               STRCVARS,FARR,LARR,'LIST     ');
```

## Example 4

Define three variables (CVAR1, CVAR2, and CVAR3) all with data format of CHAR and lengths 4, 4, and 8 respectively.

```
DECLARE

1 VARS,
  3 CVAR1   CHAR(4),
  3 CVAR2   CHAR(4),
  3 CVAR3   CHAR(8),
FVAR      CHAR(8),
LARR(3)   FIXED BIN(31);

FVAR = 'CHAR';
LARR(1) = 4;
LARR(2) = 4;
LARR(3) = 8;

CALL ISPLINK ('VDEFINE ','(CVAR1 CVAR2 CVAR3)',
VARS,FVAR,LARR,'LIST ',' ','LFORMAT');
```

## VDEFINE Exit Routine

The dialog writer can specify an exit routine to define dialog variables when program variables are non-standard (other than BINSTR, BIT, CHAR, DBCS, FLOAT, FIXED, HEX, PACK, or PACK(N)). Then, when a variable is accessed by any DM component service, the exit routine is invoked to perform any conversion necessary between the program variable's format and the character string format required for a dialog variable.

The dialog writer must specify the following information in the dialog function that VDEFINEs the variables to be formatted by the exit routine.

1. A storage location must be defined that contains the entry point address of the user exit and any other user data that should be passed to the exit routine. For example:

   ```
   DECLARE USERXT EXTERNAL ENTRY;  /*USERXT IS THE NAME OF THE*/
                                   /*EXIT ROUTINE            */
   DECLARE 1 XITINFOR,
             2 XITPTR ENTRY VARIABLE,
             2 USERDATA CHARACTER; /*CONTAIN ANY USER DATA TO */
                                   /*BE PASSED TO THE EXIT    */
                                   /*ROUTINE                  */
   ```

2. The VDEFINE must specify a format of USER and specify the area that contains the address of the exit routine and the user data field. If the defined variable name is '*', all unresolved dialog variable accesses result in the call of the exit routine. Unresolved dialog variables are those that were not implicitly entered or defined in the function pool.

   ```
   ISPLINK ('VDEFINE ', '(VAR )', VAR,
            'USER    ', 4, ' ', XITINFOR)
   ```

ISPF invokes the exit routine using a call (BALR 14,15), and standard OS linkage conventions must be followed. The parameters passed by ISPF to the exit routine are shown on the call. The exit is invoked with:

```
CALL XRTN( UDATA,     /* invoke exit and pass user area */
           SRVCODE,   /* request code                   */
           NAMESTR,   /* name length and name chars     */
           DEFLEN,    /* defined area length           */
           DEFAREA,   /* defined area                  */
           SPFDLEN,   /* ISPF data length              */
           SPFDATAP); /* ISPF data address             */
```

**UDATA**
An area that follows the exit routine address parameter, specified on the VDEFINE statement. This area can contain any additional information the user desires. Its format is CHAR(*).

If more than one variable is defined using the same exit routine, the dialog must ensure that the length and address of the converted data for each variable are returned to ISPF in unique locations. Otherwise, unexpected results can occur if a service, such as TBADD, is called with two or more of these variables.

In the example, UDATA points to an area that contains addresses for SPFDLEN and SPFDATAP to be used for the variable VAR.

**SRVCODE**
Service request-type code, as a fullword fixed value. The allowable values are 0 for a read and 1 for a write. Other values should be accepted without error, in order to allow further extensions. Codes of 2 and 3 are used by the dialog test facility variable query function. Code 2 is a request for the number of variables to be returned in the SPFDLEN field. Code 3 is a request for the names of the variables to be returned in the buffer pointed to by SPFDATAP. The names are entered as contiguous 8-byte tokens.

**NAMESTR**
Name of the dialog variable being requested, preceded by the 1-byte name length.

**DEFLEN**
The length of the area specified to the VDEFINE service. Its format is a fullword fixed value.

**DEFAREA**
The area specified to the VDEFINE service. Its format is CHAR(*).

**SPFDLEN**
For a write request, the length of the SPFDATA area is supplied by ISPF to the exit routine. For a read request, the length of the data is returned to ISPF. It must be supplied by the exit routine. Its format is a fullword fixed value.

**SPFDATAP**
For a write request, the address of the data to be stored is supplied by ISPF to the exit routine. For a read request, the address of the data is returned to ISPF. Its format is a fullword pointer.

## Return Codes

The following return codes are possible and should be set in the exit routine:

**0** Successful operation.

**8** Variable not found on read request.

**Others**
Severe error

## Example of Using the VDEFINE Exit

```
*****************************************************************
* THIS CSECT, NAMED USERXT, IS A SIMPLE EXAMPLE OF A            *
* VDEFINE EXIT.   ITS PURPOSE IS TO ILLUSTRATE HOW TO           *
* USE THE VDEFINE EXIT INTERFACE.   USERXT CONVERTS BINARY      *
* DATA IN A PROGRAM TO CHARACTER DATA USED BY ISPF.             *
* GENERALLY, AN EXIT ROUTINE IS NOT REQUIRED TO DO THIS         *
* CONVERSION, BECAUSE ISPF PROVIDES THE CAPABILITY TO DO        *
* THE CONVERSION.                                               *
*                                                               *
* THIS EXAMPLE ASSUMES THAT ALL VARIABLES PASSED FOR            *
* CONVERSION HAVE BEEN EXPLICITLY DEFINED TO ISPF               *
* (USING THE VDEFINE SERVICE), AND ARE, THEREFORE, IN THE       *
* FUNCTION POOL.  IT DOES NOT TAKE INTO CONSIDERATION THE       *
* CASE OF AN ASTERISK (*) BEING SPECIFIED FOR THE               *
* NAME-LIST PARAMETER OF THE VDEFINE SERVICE.  SEE THE          *
* VDEFINE SERVICE DESCRIPTION FOR MORE INFORMATION.             *
*                                                               *
* USERXT IS INVOKED BY ISPF USING A CALL (BALR 14,15) AS        *
* SHOWN BELOW.  STANDARD OS LINKAGE CONVENTIONS MUST BE         *
* FOLLOWED.  USERXT IS INVOKED AS FOLLOWS:                      *
*    CALL USERXT( UDATA,    /* USER DATA                      */*
*                 SRVCODE,  /* SERVICE REQUEST CODE           */*
*                 NAMESTR,  /* NAME LENGTH AND NAME           */*
*                 DEFLEN,   /* LENGTH OF AREA SPECIFIED TO    */*
*                              VDEFINE                          */*
*                 DEFAREA,  /* AREA SPECIFIED TO VDEFINE      */*
*                 SPFDLEN,  /* ISPF DATA LENGTH               */*
*                 SPFDATAP);/* ISPF DATA ADDRESS              */*
*                                                                *
*                                                                *
*****************************************************************
USERXT    CSECT
          STM   14,12,12(13)  * STANDARD LINKAGE                 *
          BALR  12,0
          USING  *,12
          ST    13,SAVE+4
          LA    15,SAVE
          ST    15,8(13)
          LR    13,15
*****************************************************************
* DETERMINE SERVICE REQUESTED.  A SRVCODE OF 0 IS A READ REQUEST *
* AND A SRVCODE OF 1 IS A WRITE REQUEST.                        *
*****************************************************************
          L     2,4(1)        * OBTAIN SRVCODE PARAMETER         *
          XR    3,3           * GET   0, 0 REPRESENTS A READ     *
          C     3,0(2)        * COMPARE THE SRVCODE TO 0         *
          BE    READ          * BRANCH TO READ IF SRVCODE IS 0   *
          LA    3,1           * GET 1, 1 REPRESENTS A WRITE      *
          C     3,0(2)        * COMPARE THE SRVCODE TO 1         *
          BNE   END           * BRANCH TO THE END IF NOT A WRITE *
```

```
*****************************************************************
* FOR A WRITE REQUEST THE LENGTH OF THE SPFDATA AREA IS SUPPLIED *
* AND THE ADDRESS OF THE DATA TO BE STORED IS SUPPLIED.  THE     *
* DEFAREA WILL BE UPDATED WITH THE CONVERTED DATA.               *
*****************************************************************
WRITE     L     2,20(1)       * OBTAIN SPFDLEN PARAMETER         *
          L     4,0(2)        *                                  *
          ST    4,SPFLEN      * SAVE THE SPFDLEN PARAMETER       *
          S     4,ONE         * DECREMENT BY ONE FOR EXECUTE     *
          L     5,WRKLEN      * OBTAIN LENGTH OF THE WRKAREA     *
          XR    4,5           * COMBINE THE EXECUTE LENGTHS      *
          L     2,24(1)       * OBTAIN SPFDATAP PARAMETER        *
          L     3,0(2)        *                                  *
          XC    WRKAREA,WRKAREA * CLEAR THE WRKAREA              *
          EX    4,PACK        * EXECUTE THE PACK COMMAND         *
          CVB   6,WRKAREA     * CONVERT THE DATA TO BINARY       *
          ST    6,TEMP        * STORE THE CONVERTED DATA         *
          L     2,16(1)       * OBTAIN THE DEFAREA PARAMETER     *
          MVC   0(4,2),TEMP   * STORE CONVERTED DATA IN DEFAREA  *
          B     END
*****************************************************************
* FOR A READ REQUEST THE LENGTH OF THE DATA AND THE ADDRESS OF   *
* THE DATA ARE RETURNED TO ISPF.  THE DATA AND ITS LENGTH ARE    *
* OBTAINED FROM THE DEFAREA AND DEFLEN.                          *
*****************************************************************
READ      XC    WRKAREA,WRKAREA * CLEAR THE WRKAREA              *
          L     5,0(1)          * ADDRESS OF USER DATA           *
          USING UDATA,5         * GET ADDRESSABILITY             *
          L     2,16(1)         * OBTAIN THE DEFAREA PARAMETER   *
          L     6,0(2)          * OBTAIN THE DATA                *
          CVD   6,WRKAREA       * CONVERT THE DATA TO DECIMAL    *
          L     7,CONVADD       * ADDRESS TO STORE CONVERTED DATA*
          UNPK  0(7),WRKAREA    * UNPACK THE DATA                *
          MVZ   14(1,7),0(7)    * MOVE THE ZONE                  *
          L     2,24(1)         * OBTAIN THE SPFDATAP PARAMETER  *
          ST    7,0(2)          * RETURN THE ADDRESS OF THE DATA *
          L     7,CONVLNG       * ADDR TO STORE CONV DATA LENGTH *
          L     4,FIFTEEN       * RETURN THE LENGTH OF THE DATA  *
          ST    4,0(7)          *                                *
          L     2,20(1)         * OBTAIN THE SPFDLEN PARAMETER   *
          ST    7,0(2)          * RETURN THE LENGTH OF THE DATA  *
END       SR    15,15           * SET GOOD RETURN CODE           *
          L     13,SAVE+4       * STANDARD EXIT LINKAGE          *
          L     14,12(13)
          LM    0,12,20(13)
          BR    14
          DS    0H
PACK      PACK  WRKAREA(0),0(0,3)
SAVE      DC    18F'0'          * REGISTER SAVE AREA             *
WRKAREA   DS    D               * CONVERSION WORKAREA            *
SPFLEN    DS    F               * LENGTH OF DATA FROM ISPF       *
TEMP      DS    F               * CONVERSION TEMPORARY AREA      *
ONE       DC    F'1'            * CONSTANT 1                     *
FIFTEEN   DC    F'15'           * LENGTH OF OUTDATA              *
WRKLEN    DC    X'00000070'     * LENGTH OF WRKAREA FOR EXECUTE  *
UDATA     DSECT                 * USER DATA                      *
CONVLNG   DS    AL(4)           * ADDRESS OF CONV DATA LENGTH    *
CONVADD   DS    AL(4)           * ADDRESS OF CONV DATA           *
          END   USERXT
```

## VDELETE – Remove a Definition of Function Variables

The VDELETE service is used only with CALL ISPLINK or CALL ISPLNK calls.

The VDELETE service removes variable names, previously defined by the program module, from the function pool. This service is the opposite of VDEFINE.

## Command Invocation Format

```
ISPEXEC  *This service does not apply to APL2 or
          command procedures*
```

## Call Invocation Format

```
CALL  ISPEXEC  *This service cannot be used with this interface*

OR

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

## Parameters

**name-list**
Specifies the dialog variable names that are to be removed from the function pool, or contains an asterisk.

An asterisk (*) specifies removal from the function pool of all dialog variable names previously defined by the program module, including exit routine definitions.

## Return Codes

The following return codes are possible:

**0** Normal completion.

**8** At least one variable not found.

**20** Severe error.

## Example

Remove ISPF accessibility to a PL/I program variable that was previously established by VDEFINE to be accessible using dialog variable name MSGNAME.

```
CALL ISPLINK ('VDELETE ','MSGNAME ');
```

# VERASE – Remove Variables from Shared or Profile Pool

The VERASE service removes variable names and values from the shared pool, the application profile pool, or both. System variables, variable type 'non-modifiable', cannot be removed by using the VERASE service.

## Command Invocation Format

```
ISPEXEC  VERASE  name-list
                 [ASIS|SHARED|PROFILE|BOTH]
```

## Call Invocation Format

```
CALL  ISPEXEC (buf-len, buffer);

OR

CALL  ISPLINK ('VERASE  ', name-list
       [,'ASIS    '|SHARED  '|'PROFILE '|'BOTH    '] );
```

## Parameters

**name-list**
Specifies the dialog variable names that are to be removed from the shared or application profile pool.

**ASIS**
Specifies that the variables are to be removed from the shared pool or, if not found in the shared pool, they are to be removed from the application profile pool. ASIS is the default value.

**SHARED**
Specifies that the variables are to be removed from the shared pool.

**PROFILE**
Specifies that the variables are to be removed from the application profile pool.

**BOTH**
Specifies that the variables are to be removed from both the shared pool and the application profile pools.

**buf-len**
Specifies a fullword fixed binary integer containing the length of "buffer."

**buffer**
Specifies a buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC call for a command procedure.

## Return Codes

The following return codes are possible:

**0** Normal completion.

**8** At least one variable not found.

**20** Severe error.

**Notes:**

1. ISPF processes the entire name list even if it cannot find one or more of the variable names in the list.
2. With BOTH specified, a 0 return code indicates that ISPF found and removed the variable from the profile and/or the shared pool. A return code of 8 indicates that ISPF did not find or remove the variable from either the profile or the shared pool.
3. Be careful not to erase variables that provide functions for you during the ISPF session. For example, if you erase function key variables (ZPF01-ZPF24) and do not subsequently specify them, the keys become inoperative.

## Example

In a CLIST, remove variables NAME, PHONE, and ADDRESS from both the shared and application profile pools.

```
ISPEXEC VERASE (NAME PHONE ADDRESS) BOTH
```

or alternately

Set program variable BUFFER to:

```
VERASE (NAME PHONE ADDRESS) BOTH
```

Then set program variable BUFLEN to the length of variable BUFFER and issue:

```
CALL ISPEXEC (BUFLEN, BUFFER);
```

or alternately

```
CALL ISPLINK ('VERASE  ','(NAME PHONE ADDRESS)','BOTH    ');
```

# VGET – Retrieve Variables from a Pool or Profile

The VGET service copies values from dialog variables in the shared pool or the application profile pool to the function pool variables with the same names. If a named function variable already exists, it is updated. If not, it is created as an implicit function variable, and then updated.

## Command Invocation Format

```
ISPEXEC  VGET  name-list  [ASIS|SHARED|PROFILE]
```

## Call Invocation Format

```
CALL  ISPEXEC (buf-len, buffer);

OR

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

## Parameters

**name-list**
Specifies the names of one or more dialog variables whose values are to be copied from the shared or profile pool to the function pool. The names are passed in the standard name-list format.

**ASIS**
Specifies that the variables are to be copied from the shared pool or, if not found there, from the profile pool.

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

**PROFILE**
Specifies that the variables are to be copied from the application profile. A shared pool variable with the same name is deleted, even if it is not found in the profile pool.

**buf-len**
Specifies a fullword fixed binary integer containing the length of "buffer."

**buffer**
Specifies a buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC call for a command procedure.

## Return Codes

The following return codes are possible:

**0** Normal completion.

**8** Variable not found.

**12** Validation failed.

**16** Translation error or truncation occurred during data movement.

**20** Severe error.

**Note:** If you issue a VGET request for a variable that does not exist in the pool from which you are trying to copy (shared or profile), the value of the function pool variable is still updated. Character variables are set to blanks. Fixed, bit, and hex variables are set to nulls (all zeros).

### Example

In a CLIST, copy from the shared pool to the function pool values for variables whose names are listed in variable VARLIST.

```
ISPEXEC VGET (&VARLIST) SHARED
```

In a PL/I program, VARLIST contains a list of variable names. Copy values for these variables from the shared pool to the function pool. The variable VARLIST has been made accessible to ISPF by a previous VDEFINE operation. Set the program variable BUFFER to contain:

```
VGET (&VARLIST) SHARED
```

Set program variable BUFLEN to the length of the variable BUFFER. Issue the following command:

```
CALL ISPEXEC (BUFLEN, BUFFER);
```

or alternately

```
CALL ISPLINK ('VGET    ',VARLIST,'SHARED  ');
```

## VIEW – View a Data Set

The VIEW service enables you to manipulate data without the risk of saving changes. As in the EDIT service, data can be manipulated through the use of familiar line and primary commands.

The VIEW service functions exactly like the EDIT service, with the following exceptions:

- You must use the REPLACE or CREATE primary command to save data. The SAVE primary command is disabled when using the VIEW service.
- If you are AUTOSAVE mode and enter the END primary command after you have altered the file being viewed, the View Warning pop-up panel gives you the option of exiting with no changes saved (by entering the END command again), or using the CREATE or REPLACE command to save your changes. If you have made no changes to the data set or member being viewed, the VIEW service terminates as it would in EDIT mode.

The VIEW service provides an interface to the VIEW function and bypasses the display of the View Entry Panel. The VIEW interface allows you to use a customized panel for displaying data (use panel ISREFR01 as a model when creating your panel), and lets you specify the initial macro and the edit profile to be used.

You can use VIEW to view any ISPF library, concatenation of ISPF libraries, or data set that can be allocated by using the LMINIT service. You can use the service

recursively, either through nested dialogs or by entering a VIEW command while viewing a member or data set. In addition, the EDIT and BROWSE commands can be nested within a VIEW session until you run out of storage.

**Notes:**

1. Dialogs that invoke the VIEW service may invoke the EDREC service first to start view recovery, because the VIEW service does not do view recovery.
2. The VIEW service might alter the DISPLAY environment. Do not expect the DISPLAY environment that existed before invoking the VIEW service to remain unchanged.
3. The VIEW service cannot be issued by a PL/I main program that also uses subtasking.
4. When you do an EDREC QUERY, ZEDMODE is set to V for View or E for Edit.

When VIEW is operating in recovery mode, a record of your interactions is automatically recorded in a PDF-controlled data set. Following a system failure, you can use the record to recover the data you were viewing.

A dialog using VIEW can place data into the ZEDUSER dialog variable in the shared pool. The data in ZEDUSER is saved in the edit recovery table as an extension variable when the recovery data set is initialized. This is done if RECOVERY is ON when first entering view or after using the CREATE or REPLACE command. The data is then made available in dialog variable ZEDUSER at the time view recovery is processed.

You can use VIEW to display workstation files on the host and host data sets on the workstation. The ZWSWFN variable is the workstation working file name that is generated by ISPF. The variable ZLRECL is the LRECL of the host data set being edited. Both can be used in the workstation VIEW parameters field. ISPF interprets any string that starts with an ampersand (&) as a system variable and evaluates it before passing to the workstation command. Strings that do not start with an ampersand are passed as is.

Batch commands can also be specified in the Workstation Browse/View fields, besides the View program name. If you have a file transferred to the workstation that you wish to do some work on besides View, you can do that in the beginning of the batch file before invoking the editor. Depending on the parameters passed to the batch command, you can also have conditional logic to perform other functions as well.

## Command Invocation Format

```
ISPEXEC  VIEW  DATASET(dsname)   [VOLUME(serial)]
                                 [PASSWORD(pswd-value)]
                                 [PANEL(panel-name)]
                                 [MACRO(macro-name)]
                                 [PROFILE(profile-name)]
                                 [FORMAT(format-name)]
                                 [MIXED(YES|NO)]
                                 [CONFIRM(YES|NO)]
                                 [WS(YES|NO)]
```

```
                                    [CHGWARN(YES|NO)]
                                    [PARM(parm-var) ]
OR
ISPEXEC  VIEW  DATAID(data-id)      [MEMBER(member-name)]
                                    [PANEL(panel-name)]
                                    [MACRO(macro-name)]
                                    [PROFILE(profile-name)]
                                    [FORMAT(format-name)]
                                    [MIXED(YES|NO)]
                                    [CONFIRM(YES|NO)]
                                    [WS(YES|NO)]
                                    [CHGWARN(YES|NO)]
                                    [PARM(parm-var)]
OR
ISPEXEC  VIEW  WSFN(ws-filename)    [PANEL(panel-name)]
                                    [MACRO(macro-name)]
                                    [PROFILE(profile-name)]
                                    [FORMAT(format-name)]
                                    [MIXED(YES|NO)]
                                    [CONFIRM(YES|NO)]
                                    [WS(YES|NO)]
                                    [CHGWARN(YES|NO)]
                                    [PARM(parm-var)]
```

## Call Invocation Format

```
CALL  ISPLINK ('VIEW    ', {dsname} ,[serial]
                                    ,[pswd-value]
                                    ,[panel-name]
                                    ,[macro-name]
                                    ,[profile-name]
                                    ,{data-id}
                                    ,[member-name]
                                    ,[format-name]
                                    ,['YES     '|'NO      ']
                                    ,['YES     '|'NO      ']
```

```
                                        ,{ws-filename-buffer-name}
                                        ,['YES'|'NO']
                                        , ['YES'|'NO']
                                        , [parm-var] );
OR
CALL ISPEXEC (buf-len, buffer);
```

## Parameters

**dsname**
The data set name, in TSO syntax, of the data set to be viewed. This is equivalent to the "other" data set name on the View Entry Panel. You can specify a fully qualified data set name enclosed in apostrophes (' '). If the apostrophes are omitted, the TSO data set prefix from the user's TSO profile is automatically attached to the data set name. The maximum length of this parameter is 56 characters.

For ISPF libraries and MVS partitioned data sets, you can specify a member name or a pattern enclosed in parentheses. If you do not specify a member name or if you specify a member pattern as part of the dsname specification when the DATASET keyword is used, a member selection list for the ISPF library, concatenation of libraries, or MVS partitioned data set is displayed. For more information about patterns and pattern matching, see the *ISPF User's Guide Volume I*.

**Note:** You can also specify a VSAM data set name. If a VSAM data set is specified, ISPF checks the ISPF configuration table to see if VSAM support is enabled. If it is, the specified tool is invoked. If VSAM support is not enabled, an error message is displayed.

**serial**
The serial number of the volume on which the data set resides. If you omit this parameter or code it as blank, the system catalog is searched for the data set name. The maximum length of this parameter is 6 characters.

**pswd-value**
The password if the data set has MVS password protection. Do not specify a password for RACF- or PCF-protected data sets.

**panel-name**
The name of a customized view panel, created by you, to be used when displaying the data. Refer to *ISPF Planning and Customizing* for information about developing a customized panel.

**macro-name**
The name of the first edit macro to be executed after the data is read, but before it is displayed. Refer to *ISPF Edit and Edit Macros* for more information.

**profile-name**
The name of the edit profile to be used. If you do not specify a profile name, the profile name defaults to the ISPF library type or last qualifier of the "other" TSO data set name. Refer to *ISPF Edit and Edit Macros* for more information.

**format-name**
The name of the format to be used to reformat the data. The format-name parameter is provided to support the IBM 5550 terminal using the Double-Byte Character Set (DBCS).

**YES | NO**
For the MIXED parameter, if YES is specified, the VIEW service treats the data as mixed-mode DBCS data. If NO (the default value) is specified, the data is treated as EBCDIC (single-byte) data. This parameter is provided to support the IBM 5550 terminal using the Double-Byte Character Set (DBCS).

**YES | NO**
For the CONFIRM parameter, if you specify YES (the default) and then attempt to CANCEL, MOVE, or REPLACE data while in VIEW mode, ISPF displays a pop-up panel that requires you to confirm the action. Because members or data sets that are cancelled, moved, or replaced are deleted, CONFIRM acts as a safeguard against accidental data loss. If you want to terminate the view session without saving the data, press the ENTER key. If you made a mistake and want to return to the view session, enter the END command. If you specify NO as the CONFIRM value, you will not be required to confirm a CANCEL, MOVE, or REPLACE.

**YES | NO**
For the WS keyword, if you specify YES the VIEW service enables you to view the host data set or workstation file on the workstation using the workstation tool configured in the ISPF Tool Configurator. For more information, see the section on Workstation Tool Integration in the Settings (Option 0) chapter of the *ISPF User's Guide Volume II*. If you specify NO as the WS value, the VIEW service views the host data set or workstation file on the host.

**YES | NO**
For the CHGWARN keyword, if you specify YES the VIEW service gives a warning when the first data change is made, indicating that data cannot be saved in View. If you specify NO, no data change warning is given.

**data-id**
The data ID that was returned from the LMINIT service. The maximum length of this parameter is 8 characters.

You can use the LMINIT service in either of two ways before invoking the VIEW service:

- You can use LMINIT to allocate existing data sets by specifying a data set name or ISPF library qualifiers. LMINIT returns a data ID as output. This data ID, rather than a data set name, is then passed as input to the VIEW service.
- The dialog can allocate its own data sets by using the TSO ALLOCATE command or MVS dynamic allocation, and then passing the ddname to LMINIT. Again, a data ID is returned as output from LMINIT and subsequently passed to the VIEW service. This procedure is called the *ddname interface* to VIEW. It is particularly useful for viewing VIO data sets, which cannot be accessed by data set name because they are not cataloged.

**member-name**
A member of an ISPF library or MVS partitioned data set, or a pattern. If you do not specify a member name when the MEMBER keyword or call invocation is used, or if a pattern is specified, a member selection list for the ISPF library, concatenation of libraries, or MVS partitioned data set is displayed. For more information about patterns and pattern matching, see the *ISPF User's Guide Volume I*.

**ws-filename-buffer-name**
Specifies the name of a variable containing the path and the file name (in the syntax of the workstation's operating system) of the workstation file to be edited. The maximum length of the path and the workstation file name within this variable is 256. If the path is omitted, the working directory configured in the ISPF Tool Configurator will be inserted in front of the workstation file name to resolve the relative path. For more information, see the section on Workstation Tool Integration in the Settings (Option 0) chapter of the *ISPF User's Guide Volume II*.

**buf-len**
A fullword fixed binary integer containing the length of the buffer parameter.

**buffer**
A buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC invocation for a command invocation.

**parm-var**
The name of an ISPF variable that contains parameters which are to be passed to the initial macro specified by *macro-name*. The variable value must not exceed 200 bytes in length. If no macro name is specified, parm-var must be blank or not specified.

## Return Codes

The following return codes are possible:

**0** Normal completion. Browse was substituted for VIEW if insufficient storage was available to read in the requested data.

**Note:** Data can only be saved through the CREATE or REPLACE primary commands.

**12** VIEW has been disabled through the ISPF configuration table.

**14** Member or sequential data set in use.

**16** One of the following:
- No members matched the specified pattern
- No members in the partitioned data set.

**18** A VSAM data set was specified but the ISPF Configuration Table does not allow VSAM processing.

**20** Severe error; unable to continue.

## Example 1

This example invokes the EDIT service for TELOUT, a member of the ISPFPROJ.FTOUTPUT data set.

### Command Invocation

```
ISPEXEC VIEW DATASET('ISPFPROJ.FTOUTPUT(TELOUT)') WS(YES)

OR

ISPEXEC LMINIT DATAID(EDT) DATASET('ISPFPROJ.FTOUTPUT')
ISPEXEC VIEW DATAID(&EDT) MEMBER(TELOUT) WS(YES)
```

### Call Invocation

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

*OR*

Set the program variable BUFFER to contain:

```
BUFFER = 'VIEW DATASET('ISPFPROJ.FTOUTPUT(TELOUT)'') WS(YES)';
```

Set the program variable BUFFLN to the length of the variable BUFFER. Issue the following:

```
CALL ISPEXEC (BUFFLN, BUFFER);
```

## Example 2

This example invokes the EDIT service for a workstation file, c:\config.sys, using the PDF editor on the host.

## Command Invocation

Set the command variable WSFNNAME to contain:

```
WSFNNAME='c:\config.sys'

ISPEXEC VIEW WSFN(WSFNNAME) WS(NO)
```

### Call Invocation

Set the program variable to contain:

```
WSFNNAME='c:\config.sys';

CALL ISPLINK('VIEW',' ',' ',' ',' ',' ',' ',
    ' ',' ',' ',' ',' ',' ',' WFSNNAME','NO');
```

*OR*

Set the program variable WSFNNAME to contain:

```
WSFNNAME='c:\config.sys';
```

Set the program variable BUFFER to contain:

```
BUFFER='VIEW WSFN(WSFNNAME) WS(NO)';
```

Set the program variable BUFFLEN to the length of the variable BUFFER. Issue the following:

```
CAll ISPEXEC(BUFFLEN,BUFFER);
```

# VIIF – View Interface

The View Interface (VIIF) service provides view functions for data accessed through dialog-supplied I/O routines. The invoking dialog must perform all environment-dependent functions such as file allocation, opening, reading, closing, and freeing. The dialog is also responsible for any Enqueue/Dequeue serialization that is required. With the dialog providing the I/O routines, VIIF allows you to:

- View data other than partitioned data sets or sequential files such as subsystem data, and in-storage data.
- Do preprocessing and post-processing of the data being viewed.

The invoking dialog must provide addresses to routines that:

- Read the data sequentially from beginning to end, returning to View one record on each invocation.
- Perform processing for the MOVE, COPY, and VIEW primary commands. If this routine is not specified, ISPF processes these commands.

These addresses must be 31-bit addresses, and the routines must have an addressing mode (AMODE) of 31.

When a View session is operating in recovery mode, a record of your interactions is automatically recorded in a PDF-controlled data set. Following a system failure, you can use the record to recover the data you were viewing.

**Note:** Dialogs that invoke the VIIF service may invoke the EDIREC service first to start view recovery. The VIIF service itself does not do view recovery.

A dialog using VIIF can place data into the ZEIUSER dialog variable in the shared pool. When the system initializes the recovery data set, the system also saves the data in ZEIUSER in the Edit recovery table as an extension variable. This is done if RECOVERY is ON when first entering View or after you use the CREATE or REPLACE commands. This data is then made available in dialog variable ZEIUSER at the time view recovery is processed.

## Command Invocation Format

You cannot use command procedures to invoke this service.

## Call Invocation Format

The format for invoking VIIF can be different depending on whether you want to process a pending view recovery. If you do not want to process a pending view recovery, the format is:

```
CALL ISPLINK ('VIIF    ', [data-name] ,profile-name
                         ,rec-format ,rec-len

                         ,read-routine

                         ,[cmd-routine] ,[dialog-data]

                         ,[edit-len] ,[panel-name]

                         ,[macro-name] ,[format-name]

                         ,['YES     '|'NO      ']

                         ,['YES     '|'NO      ']

                         ,[parm-var] );
```

You must use the VIIF service to recover data viewed in a previous VIIF session. You must invoke the EDIREC service first to see if a recovery is pending. If you want to process a pending recovery, use the following format for VIIF, specifying YES for the recovery-request parameter.

```
CALL ISPLINK ('VIIF    ' ,[data-name] ,' '
                         ,[rec-format] ,[rec-len]

                         ,read-routine

                         ,[cmd-routine] ,[dialog-data]
```

```
                                ,' ' ,' ' ,' ' ,' ' ,' '
                                ,'YES    ');
```

## Parameters

**data-name**
This parameter allows you to specify a data name for the source data to be viewed. This name appears in the title line of the default View panel. It is also the target data name for an edit recovery table entry when edit recovery is active. This name must not have any embedded blanks, and its maximum length is 54 characters. This name is stored in ZDSNT in the function pool.

**profile-name**
The name of the edit profile to be used. This parameter is required when recovery-request = NO (or is not specified); otherwise, it is not allowed.

**rec-format**
The record format: F - fixed, V - variable. This parameter is required when recovery-request = NO (or is not specified); otherwise, it is optional, but it must be the same record format that was specified when recovery was initiated for the data.

**rec-len**
The record length, in bytes. It must be a positive numeric value between 10 and 32 760, inclusive. For variable record format, this is the maximum record length. This parameter is required when recovery-request = NO (or is not specified); otherwise, it is optional, but it must be the same record length that was specified when recovery was initiated for the data.

**read-routine**
A fullword address indicating the entry point of a dialog-supplied read routine (required). See “Read Routine” on page 292 for more information about this parameter.

**cmd-routine**
A fullword address indicating the entry point of a dialog-supplied routine that processes the MOVE, COPY, and VIEW primary commands. See “Command Routine” on page 292 for more information about this parameter. If this parameter is not specified, ISPF processes these commands.

**dialog-data**
A fullword address indicating the beginning of a dialog data area. This address is passed to the dialog-supplied routines. If no address is supplied, zeros are passed to the dialog routines. This data area provides a communication area for the dialog.

**edit-len**
The length, in bytes, of the data to be displayed for viewing. This parameter indicates that the data records should be considered to have a length shorter than rec-len during viewing. Thus, the dialog may include data in the record that is not accessible for viewing.

Edit-len must be a numeric value between 10 and 32 760, inclusive, and must be less than or equal to parameter rec-len. Rec-len is the default. If the edit-len parameter is specified, the data that is not displayed are the bytes from (edit-len +1) to rec-len. That means the inaccessible record data is at the end of the record.

The edit-len parameter is optional when recovery-request = NO (or is not specified); otherwise, it is not allowed. The edit-len parameter is not allowed when format-name is specified.

**panel-name**
The name of the panel to use for displaying the data. This parameter is optional when recovery-request = NO (or is not specified); otherwise, it is not allowed. The default is the standard View data display panel. Refer to *ISPF Planning and Customizing* for information about developing a customized panel.

**macro-name**
The name of the initial macro to be executed. This parameter is optional when recovery-request = NO (or is not specified); otherwise, it is not allowed. The default is no initial macro. Refer to *ISPF Edit and Edit Macros* for more information on macros.

**format-name**
The name of the format to be used to reformat the data. This parameter is optional when recovery-request = NO (or is not specified); otherwise, it is not allowed. The default is no format. This parameter is provided to support the IBM 5550 terminal using the Double-Byte Character Set (DBCS). This parameter is not allowed when the edit-len parameter is specified.

**YES | NO (mixed-mode)**
Specifies whether the data is treated as mixed-mode DBCS data. This parameter is optional when recovery-request = NO (or is not specified); otherwise, it is not allowed. If YES is specified, the VIIF service treats the data as mixed-mode DBCS data. If NO (the default) is specified, the data is treated as EBCDIC (single-byte) data. This parameter is provided to support the IBM 5550 terminal using the Double-Byte Character Set (DBCS).

**YES | NO (recovery-request)**
Specifies whether to process a pending view recovery that was being viewed with the VIIF service when a system failure occurred. If YES is specified, the edit recovery should proceed. This function is similar to the EDREC service with the PROCESS option. If YES is specified to process the view recovery, you must specify the read routine and write routine, but you must not specify profile name, edit-len, panel-name, macro-name, format-name and mixed-mode. If NO (the default) is specified, no edit recovery is processed; VIIF views the specified data.

**parm-var**
The name of an ISPF variable that contains parameters which are to be passed to the initial macro specified by *macro-name*. The variable value must not exceed 200 bytes in length. If no macro name is specified, parm-var must be blank or not specified.

## Dialog-Supplied Routines

All dialog-supplied routines are invoked using standard linkage. All addresses must be 31-bit addresses, and the addressing mode (AMODE) of the routines must be AMODE=31.

A VIIF read routine must have an assembler interface to be used in a call to VIIF.

**Note:** The dialog-supplied routines can be written in LE member languages (languages that require the LE runtime environment). However, a mixture of

LE-conforming main dialog code and service routine code is not supported. Dialogs and service routines must either all be LE-conforming or all be LE-nonconforming.

## Read Routine

VIIF calls the read routine repeatedly to obtain all of the data records to be viewed at the beginning of the View session. This routine is also called to obtain data records for the MOVE and COPY commands when the dialog is handling the processing for these commands. The dialog-supplied read routine is invoked with the following parameters:

- Fullword pointer to record data read (output from read routine)
- Fullword fixed binary data length of record read if rec-format is 'V'
- Fullword fixed binary request code. Request settings are as follows:

  0 Read next record
  1 First read request

- Fullword dialog data area address.

## Command Routine

The dialog-supplied command routine, when specified, processes the MOVE, COPY, and VIEW primary commands. The command routine is invoked with the following parameters:

- Fullword fixed binary function code word. Decimal values of function settings are as follows:

  1*n* Move
  2*n* Copy
  5*n* Recursive view

  where *n* is 0 (beginning of function), 1 (successful completion), or 2 (unsuccessful completion). This *n* value will always be 0 for a recursive View function; that is, the View request code will be 50.

- Fullword dialog data area address.

To access parameters that can follow the command, the command routine must access the ZCMD dialog variable from the SHARED variable pool.

For a MOVE or COPY the command routine initiates the processing for the requested function. When the return code from the command routine is zero, VIIF calls the read routine to transfer the data. After the read is completed, the command routine is called once more to handle any termination processing that may be required for the requested function. For example, the MOVE function would need to delete the data that was moved.

For the VIEW command, the command routine must perform all processing required to effect the desired results for the purposes of the dialog. For example, the dialog can consider the VIEW command to be an invalid command. The command routine is called only once for each VIEW command.

## Return Codes

When a dialog routine terminates with a return code (12 or higher or an unexpected return code), the dialog can issue a SETMSG to generate a message on the next panel display. If the dialog does not set a message, the VIIF service will issue a default message.

## Read Routine

**0** Normal completion.

**8** End of data records (no data record returned).

**16** Read error. If a read error is encountered when the system builds the initial view display, the VIIF service terminates with a return code of 20. Otherwise, the view data is redisplayed.

**20** Severe error. (The VIIF service terminates immediately with a return code of 20.)

## Command Routine Return Codes

**0** Normal completion.

**4** ISPF should process the requested function.

**12** Command deferred; retain the command on the Command line. View data is redisplayed.

**20** Severe error. (The VIIF service terminates immediately with a return code of 20.)

## VIIF Service Return Codes

**0** Normal completion, data not saved.

**12** View has been disabled through the configuration table.

**16** Unexpected return code received from a dialog-supplied routine. When an unexpected return code is received, the VIIF service terminates immediately with a return code of 16.

**20** Severe error; unable to continue.

After the View session has been terminated, control is returned to the invoking dialog with a return code indicating the completion status.

## Example

This example invokes the VIIF service to view data called EDIFDSN, which has a fixed-record format with a record length of 80 characters. An edit profile (EDIFPROF), read routine (RDRTN) and command routine (CMDRTN) are supplied, as is a dialog data area (MYDATA).

### Call Invocation

```
CALL ISPLINK ('VIIF    ','EDIFDSN ','EDIFPROF ','F ',80,
     RDRTN,CMDRTN,MYDATA);
```

For a more complete example of using VIIF, including dialog-supplied I/O routines and source code, see the *ISPF Dialog Developer's Guide and Reference*.

# VMASK – Mask and Edit Processing

The VMASK service associates an edit mask with a dialog variable defined with VDEFINE. The edit mask is a pattern used to validate input into that variable. The mask characters are stripped from the data before it is put into the function pool, or before the data is stored in a table from a TBDISPL. When the masked variable is displayed on a panel, stored in the shared or profile pool, or accessed by VCOPY, the output string contains the mask characters. When specifying the length to receive these variables on the VCOPY call, the length should be as long as the mask, not the defined variable. The length of the mask should also be considered when defining the field in which a masked variable is displayed.

The mask is only associated with the definition of the variable that was active when the VMASK was issued and cannot be used with implicit variables.

For example:

```
VDEFINE VAR1         123        A
VMASK   VAR1        (999)       A
VDEFINE VAR1         123        B
VCOPY   VAR1         123        B
VDELETE VAR1                    B
VCOPY   VAR1        (123)       A
```

The mask is associated with the A definition of VAR1, not the B definition.

When using a masked variable on a panel, you must issue a VEDIT in the processing section of the panel for that masked variable for the data to be accessible in the function pool. You must issue the VEDIT statement before any other panel statements that reference variables, (such as VPUT or VER). If you don't, the values in the pool will be unpredictable. The VEDIT statement indicates to ISPF that the data entered into the masked variable field should be verified and the mask stripped out. If you don't issue the VEDIT for each masked variable on the panel, the resulting data in the pool will be unpredictable.

The VMASK service is supported for programming languages. The variable must be VDEFINEd with FIXED, PACK, or CHAR formats.

## VMASK Call Invocation

```
CALL ISPLINK ('VMASK   ',name-list{,'FORMAT  '{,'IDATE   '}    }
                                  {          {,'STDDATE '}    }
                                  {          {,'ITIME   '}    }
                                  {          {,'STDTIME '}    }
                                  {          {,'JDATE   '}    }
                                  {          {,'JSTD    '})   }
                                  {,'USER    ','mask',masklen)}
```

## Parameters

**name-list**
Specifies the names of one or more dialog variables whose values are to be associated with a mask pattern.

**FORMAT|USER**
Identifies the type of mask to be associated with the dialog variable. FORMAT indicates that the mask is one of the predefined mask formats. USER indicates the mask will be user defined.

If FORMAT is specified, the following keywords are predefined mask patterns that ISPF validates.

**IDATE** This specifies a data type for which the format represents a date expressed in a 2-digit year (YY), month (MM), and day (DD).

The IDATE internal format used by the dialog variable contains 6 digits representing YYMMDD. The IDATE display format contains 8 characters, including the national language date delimiter character. For the U.S., the format is YY/MM/DD. For input only, ISPF ensures the resulting IDATE internal format value is a valid date. It ensures that the internal value for YY is 00–99, for MM is 01–12 and for DD is 01–31. Validation is also done to check the date for months with fewer than 31 days and for leap years.

**STDDATE** This specifies a data type for which the format represents a date expressed in a 4-digit year (YYYY), month (MM) and day (DD).

The STDDATE internal format used by the dialog variable contains 8 digits representing YYYYMMDD. The STDDATE display format contains 10 characters including the national language date delimiter. For the U.S., the format is YYYY/MM/DD. For input only, ISPF ensures the resulting STDDATE internal value is a valid date. It ensures that the internal value for YYYY is 0000–9999, for MM is 01–12 and for DD is 01–31. Validation is also done to check the date for months with fewer than 31 days and for leap years.

**ITIME** This specifies a data type for which the format represents time expressed in hours (HH) and minutes (MM).

The ITIME internal format used by the dialog variable contains 4 digits representing HHMM. The ITIME display format contains 5 characters including the national language time delimiter. For the U.S., the format is HH:MM. Hours are specified using the 24-hour clock. For input only, ISPF ensures the resulting ITIME internal value is a valid time. It ensures that the internal value for HH is 00–23 and for MM is 00–59.

**STDTIME** This specifies a data type for which the format represents time expressed in hours (HH), minutes (MM) and seconds (SS).

The STDTIME internal format used by the dialog variable contains 6 digits representing HHMMSS. The STDTIME display format contains 8 characters including the national language time delimiter. For the U.S., the format is HH:MM:SS. Hours are specified using the 24-hour clock. For input only, ISPF ensures the resulting STDTIME internal value is a valid time. It ensures that the internal value for HH is 00–23, for MM is 00–59 and for SS is 00–59.

**JDATE** This specifies a data type for which the format represents a date expressed in a 2-digit year (YY) and day of the year (DDD).

The JDATE internal format used by the dialog variable contains 5 digits representing YYDDD. The JDATE display format contains 6 characters in the format YY.DDD. For input

only, ISPF ensures the resulting JDATE internal value is a valid date. It ensures that the internal value for YY is 00–99 and for DDD is 365. Validation is also done to check for leap years with 366 days.

**JSTD** This specifies a data type for which the format represents a date expressed in a 4-digit year (YYYY) and day of the year (DDD).

The JSTD internal format used by the dialog variable contains 7 digits representing YYYYDDD. The JSTD display format contains 8 characters in the format is YYYY.DDD. For input only, ISPF ensures the resulting JSTD internal value is a valid date. It ensures that the internal value for YYYY is 0000–9999 and for DDD is 365. Validation is also done to check for leap years.

When a user enters a value for a variable with a type of either IDATE or STDDATE, it must be entered using the national language date format. It is a good idea to display an explanation of the expected format to the user so that the value is entered properly. ISPF verifies that the value entered is a valid date, and if no errors are found, the national language date format is converted to the internal format before the value is stored in the variable pool.

If USER is specified, the following parameters must be defined:

**mask** Identifies the mask pattern associated with the dialog variable.

A mask pattern can consist of 20 characters. The following are the valid mask symbols:

**A** Any alphabetic character (A–Z, a–z)

**B** A blank space

**9** Any numeric character (0–9)

**H** Any hexadecimal digit (0–9, A–F, a–f)

**N** Any numeric or alphabetic character (0–9, A–Z, a–z)

**V** Location of the assumed decimal point

**S** The numeric data is signed

**X** Any allowable characters from the character set of the computer

**Special characters**
( ) - / , .

The data represented by the B, V and special character symbols will be stripped before the data is put into the pool. The specified mask must contain at least one of the symbols A, 9, H, N, or X.

The S symbol must be in the first position to be accepted.

**masklen** Specifies the length of the mask in bytes. The maximum length of the mask is 20. This parameter must be specified in a fullword fixed binary integer.

## Return Codes

The following return codes are possible:

**0** Normal completion

**8** Variable not found

**20** Severe error.

## Example

In this example, a character variable (CVAR) is defined with a user-defined mask for a phone number. A fixed variable (FVAR) with a time format is specified.

```
DECLARE
   FVAR   FIXED BIN(31),
   CVAR   CHAR(10),
   LENCHR FIXED BIN(31),
   LENFIX FIXED BIN(31),
   LENMSK FIXED BIN(31);

LENCHR = 10;
LENFIX = 4;
CALL ISPLINK('VDEFINE ','(CVAR )',CVAR,'CHAR    ',LENCHR);
CALL ISPLINK('VDEFINE ','(FVAR )',FVAR,'FIXED   ',LENFIX);
LENMSK = 13;
CALL ISPLINK('VMASK   ','(CVAR )','USER    ','(999)999-9999',LENMSK);
CALL ISPLINK('VMASK   ','(FVAR )','FORMAT  ','ITIME   ');
```

## The VEDIT Statement

Use the VEDIT statement to verify mask data.

---

# VPUT – Update Variables in the Shared or Profile Pool

The VPUT service copies values from dialog variables in the function pool to the shared or application profile pool. If a variable of the same name already exists in the shared or the profile pool, it is updated. If it does not exist in the shared or profile pool, it is created in the pool specified by the parameter on the VPUT service request, and then it is updated.

## Command Invocation Format

```
ISPEXEC  VPUT  name-list [ASIS|SHARED|PROFILE]
```

## Call Invocation Format

```
CALL  ISPEXEC (buf-len, buffer);

OR

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

## Parameters

**name-list**

Specifies the names of one or more dialog variables whose values are to be copied from the function pool to the shared or profile pool. See “Invoking the ISPF Services” on page 2 for specification of name lists.

**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 to the shared pool only.

**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 of the same names are deleted.

**buf-len**
Specifies a fullword fixed binary integer containing the length of "buffer."

**buffer** Specifies a buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC call for a command procedure.

## Return Codes

The following return codes are possible:

**0** Normal completion.

**8** Variable not found.

**16** Truncation occurred while copying variables to the application profile pool.

**20** Severe error.

## Example

In a CLIST, write variables, the names of which are listed in the variable VPUTLIST, from the function pool to the shared pool.

```
ISPEXEC VPUT (&VPUTLIST ) SHARED
```

In a PL/I program, write variables, the names of which are listed in program variable VPUTLIST, from the function pool to the shared pool. The variable VPUTLIST has been made available to ISPF by a previous VDEFINE operation. Set the program variable BUFFER to contain:

```
VPUT (&VPUTLIST ) SHARED
```

Set program variable BUFLEN to the length of the variable BUFFER. Issue the following:

```
CALL ISPEXEC (BUFLEN, BUFFER);
```

or alternately

```
CALL ISPLINK ('VPUT    ',VPUTLIST,'SHARED  ');
```

# VREPLACE – Replace a Variable

The VREPLACE service is used only with CALL ISPLINK or CALL ISPLNK calls.

The VREPLACE service allows a program module to update the contents of a variable in the function pool.

The variable names can be specified as single 8-character values, a list enclosed in parentheses, or a name-list structure. An array of lengths must be supplied on input to map the area that contains the data for each of the variables.

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

## Command Invocation Format

```
ISPEXEC  *This service does not apply to
          APL2 or command procedures*
```

## Call Invocation Format

```
CALL  ISPEXEC  *This service cannot be used with this interface*
```

*OR*

```
CALL  ISPLINK ('VREPLACE', name-list, lengths, values);
```

## Parameters

**name-list**
Specifies the names of the dialog variables whose values are to be updated. The standard name-list format is used.

**lengths**
Specifies an array of values giving, for each corresponding variable in the name-list, the number of bytes of the data to be used in the updating. Each field in the array must be a fullword binary integer.

**values** Specifies, in the buffer mapped by the length array, the update data to be used in the updating.

## Return Codes

The following return codes are possible:

**0** Normal completion.

**16** Truncation has occurred during data movement.

**20** Severe error.

## Example

Copy the value of a field named QROWSD from this PL/I program module to the function variable named QROWS. Before the copy operation, if no variable with this name is found in the function pool, create one, giving it the name QROWS. Program variable L8 contains a value of 8.

```
CALL ISPLINK ('VREPLACE','QROWS ',L8,QROWSD);
```

# VRESET – Reset Function Variables

The VRESET service is used only with CALL ISPLINK or CALL ISPLNK calls.

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

## Command Invocation Format

```
ISPEXEC  *This service does not apply to
          APL or command procedures*
```

### Call Invocation Format

```
CALL  ISPEXEC  *This service cannot be used with this interface*

OR

CALL  ISPLINK ('VRESET  ');
```

### Return Codes

The following return codes are possible:

**0** Normal completion.

**20** Severe error.

### Example

Remove ISPF accessibility to all PL/I program variables.

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

## WSCON — Connect to a Workstation

The WSCON service enables you to connect to the workstation without using the GUI parameter on the ISPSTART command and the Initiate Workstation Connection panel from the ISPF settings.

You can issue the WSCON service from a program, clist, or REXX exec to connect to the workstation.

### Command Invocation Format

```
ISPEXEC  WSCON  [IP(ip_var_name)]
                [LU(lu_var_name)]
                [FI      ]
                [TITLE(title_var_name)]
                [FRAME(STD | FIX | DLG)]
                [BKGRND(STD | DLG)]
                [CODEPAGE(codepage)]
                [CHARSET(character_SET)]
                [NOGUIDSP]
                [PANEL(YES | NO | ONERROR)]
```

### Call Invocation Format

```
CALL  ISPLINK ('WSCON   '
               [,ip_var_name]
               [,lu_var_name]
               [,FI]
               [,title_var_name]
               [,STD | FIX | DLG]
               [,STD | DLG]
               [,codepage]
               [,character_set]
               [,NOGUIDSP]
               [,YES | NO | ONERROR]);

OR

CALL ISPEXEC (buf-len, buffer);
```

## Parameters

**IP(ip_var_name)**
The name of the dialog variable containing the workstation's TCP/IP address or host name. The value can be a maximum of 64 characters.

**LU(lu_var_name)**
The name of the dialog variable containing the workstation's APPC address or host name. The value can be a maximum of 64 characters.

**FI** This parameter specifies that you want ISPF to search the file allocated to ISPDTPRF DD for your network address.

**TITLE(title_var_name)**
The name of the dialog variable containing the title for the ISPF GUI panels if ZWINTTL or ZAPPTTL in not defined by the dialog. The value can be a maximum of 64 characters.

**FRAME(STD | FIX | DLG)**
This parameter specifies that the first window frame displayed in GUI mode be standard (STD), fixed (FIX), or dialog (DLG). If this parameter is not specified, the value from the user's system profile is used. If no value is saved in the system profile, STD is the default.

**BKGRND(STD | DLG)**
This parameter specifies that the first window displayed in GUI mode have standard (STD) or dialog (DLG) background color. The colors are defined by the workstation. If this parameter is not specified, the value from the user's system profile is used. If no value is saved in the system profile, DLG is the default.

**CODEPAGE(codepage)**

This numeric value is used as the host codepage in translating data from the host to the workstation, regardless of the values returned from the terminal query.

If CODEPAGE is specified, CHARSET must also be specified. If these values are not specified on the WSCON service, then values previously specified on the Initiate Workstation Connection panel and saved in the user's system profile are used. If there are no codepage and character set values saved in the system profile, then values from the terminal query are used. If your terminal or emulator does not support codepages, the CODEPAGE and CHARSET parameter values on ISPSTART are used. If ISPSTART does not have the parameters specified, English is the default.

**CHARSET(character_set)**
This numeric value is used as the host character set in translating data from the host to the workstation, regardless of the values returned from the terminal query.

If CHARSET is specified, CODEPAGE must also be specified. If these values are not specified on the WSCON service, then values previously specified on the Initiate Workstation Connection panel and saved in the user's system profile are used. If there are no codepage and character set values saved in the system profile, then values from the terminal query are used. If your terminal or emulator does not support codepages, the CODEPAGE and CHARSET parameter values on ISPSTART are used. If ISPSTART does not have the parameters specified, English is the default.

**NOGUIDSP**
This parameter specifies that you want to make a connection to the

workstation, but do not want ISPF to display in GUI mode. If this parameter is not specified, ISPF displays in GUI mode.

**PANEL(YES | NO | ONERROR)**
This parameter specifies whether you want the Initiate Workstation Connection panel to be displayed.

If YES is specified, the Initiate Workstation Connection panel fields are initialized with the values of IP, LU, and TITLE that you have specified, even if the specified dialog variable value is blank.

If NO is specified, appropriate return codes are issued if there are errors with the specified parameters. Parameter values are **not** saved in the user's system profile. NO is the default.

ONERROR specifies that the panel is to be displayed only if there is an error with the specified parameters. Any parameter that is not valid causes the panel to display with the non-valid values in the appropriate fields.

**buf-len**
This specifies a fullword fixed binary integer containing the length of ″buffer″.

**buffer**
This specifies a buffer containing the name of the service and its parameters in the same form as they would appear in an ISPEXEC call for a command procedure.

## Return Codes

The following return codes are possible:

**0** Normal completion. Connection established.

**8** The user pressed End, Exit, or Cancel from the Initiate Workstation Connection panel without making a connection.

**12** Already in GUI mode. Recursive error.

**14** Connecting in GUI mode is not supported when in partition mode or split screen.

**16** Cannot connect to workstation.

**20** Parameters not valid or syntax conflict. For example, both IP and LU specified.

## Example

This example defines the workstation address and title variables, and invokes the WSCON service to initiate a GUI display.

```
DECLARE
   GUI_TITLE CHAR(64),
   IP_ADDRESS CHAR(64),
   BLANKS   CHAR(8);

IP_ADDRESS = '32.225.17.228';
CALL ISPLINK('VDEFINE ','MYADDR  ',IP_ADDRESS,'CHAR',64);

GUI_TITLE = 'THIS IS MY TITLE FOR MY GUI SESSION';
CALL ISPLINK('VDEFINE ','MYTITLE ',GUI_TITLE,'CHAR',64);

CALL ISPLINK('WSCON   ',
       'MYADDR',          /* IP  */
       BLANKS,            /* LU  */
       BLANKS,            /* FI  */
```

```
'MYTITLE',         /* TITLE  */
'FIX',             /* FRAME  */
'STD',             /* BACKGROUND COLOR  */
37,                /* CODEPAGE  */
697,               /* CHARACTER SET  */
BLANKS,            /* NOGUIDSP  */
'ONERROR');        /* PANEL  */
```

### Restrictions

When connecting to the workstation in GUI mode, the first panel displayed in GUI mode might not contain group boxes and images that are defined on the panel. Other constructs defined within the panel based on the ZGUI variable might not display properly. After the user presses the Enter key, causing the panel to be reprocessed, then these constructs will be visible in GUI mode.

## WSDISCON — Disconnect from a Workstation

The WSDISCON service enables you to disconnect from the workstation without having to terminate your ISPF session.

You can issue the WSDISCON service from a program, clist, or REXX exec to disconnect from the workstation.

### Command Invocation Format

```
ISPEXEC  WSDISCON
```

### Call Invocation Format

```
CALL  ISPLINK ('WSDISCON');
OR

CALL ISPEXEC (buf-len, buffer);
```

### Parameters

**buf-len**
This specifies a fullword fixed binary integer containing the length of ″buffer″.

**buffer**
This specifies a buffer containing the name of the service.

### Return Codes

The following return codes are possible:

**0** Normal completion. User is disconnected from workstation.

**8** User is trying to disconnect from workstation, but there is no current connection.

**10** User is trying to disconnect from GUI mode, but is connected with GUISCRD or GUISCRW values that are different than the host emulator session. User is not disconnected.

**12** User is trying to disconnect from a GUI display when running BATCH GUI mode. User is not disconnected.

**14** User is trying to disconnect from workstation while running the Workstation Tool Integration Configuration Program. User is not disconnected.

## Usage Notes

1. If the CODEPAGE and CHARSET parameters were specified on the WSCON service, the Initiate Workstation Connection panel, or the ISPSTART GUI statement when a connection was made to the workstation, their values might no longer be the host codepage and character set in 3270 mode following a WSDISCON service invocation. The values returned from the terminal query are restored as the active codepage and character set. If your terminal or emulator does not support codepages, the CODEPAGE and CHARSET parameter values originally specified on your ISPSTART statement are used. If these parameters were not specified on ISPSTART, English is the default.
2. If you are running with TSO line mode support while displaying ISPF in GUI mode, the ISPF/TSO window continues to display TSO line mode after the WSDISCON service is issued. The ISPF/TSO window is not removed until your ISPF session is terminated.

### Restrictions

- When disconnecting from GUI mode, the name of any group boxes defined on the panel that WSDISCON was issued from will display on the screen in 3270 mode. After the user presses the Enter key, causing the panel to reprocess, then these group box names disappear and any panel text under the names reappears.
- If the user is in GUI mode and in split screen mode when the WSDISCON service is invoked, he is disconnected from the workstation, and the screen from which the WSDISCON service was invoked is displayed in the full 3270 emulator session without a split line. The setting of the user's **Always show split line** parameter has no effect on this. The other ISPF sessions are hidden and available for display after the SWAP command is entered. Additional split requests cause the split line to redisplay, provided that the **Always show split line** setting is selected.
- If pop-up windows are displayed in GUI mode when the WSDISCON service is invoked, the pop-up windows are suspended on the 3270 session. The panels are displayed as full-screen panels. If new addpops are then invoked, the new panels display as pop-ups.
- The user cannot issue WSDISCON when running in Batch GUI mode.
- The user cannot issue WSDISCON when connected to the ISPF Application Server.

# 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.

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 106–0032, 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 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.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

© Copyright IBM Corp. 1980, 2004

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:

IBM Corporation,
Mail Station P300
2455 South Road
Poughkeepsie New York 12601–5400
U.S.A.

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.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

## Programming Interface Information

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

This publication 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----------------------+


+------------------End of Programming Interface information------------------+
```

## Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

AD/Cycle
APL2
BookManager
BookMaster
C++/MVS
COBOL/370
Common User Access
CUA
DFSMSrmm
DFSMS/MVS
DFSORT
FFST
GDDM
IBM
Language Environment
MVS
MVS/XA
OS/390
RACF
SAA
Systems Application Architecture
Tivoli
VTAM
z/OS

Microsoft, Windows, and Windows NT are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, and service names may be trademarks or service marks of others.

# Index

## Special characters


% sign
  beginning a command with 32, 187
& (ampersand)
  in a variable name 4
  symbolic variables 7


## A


abbreviated (generic) search argument 250
abend 35
accessing skeleton files (FTOPEN) 73
activities
  nesting xxii
adding a member to a data set or library 124
adding a row to a table (TBADD) 196
ADDPOP parameter 186
ADDPOP service
  description 15, 21
  example 22
  relation to control service 32
  return codes 22
APL2
  cannot use ISPLINK call 5
  character-vector 9
  example 10
  general call format 9
  interface with ISPF 9
  lastrc variable 13
application commands, definition 231
assembler
  example 11
  general call format 11
Assembler language, VL keyword 7
attention exits (CLIST) 4
ATTN statement 4
audit trail
  in EDIF recovery mode 47, 289
  in EDIT recovery mode 54
  in VIEW recovery mode 283
automatic and non-automatic entry into line mode 32, 187
AUTOSEL (auto-selection)
  call parameters description 211
  command procedure description 211
AUTOSEL control variable, use with TBDISPL 216


## B


BARRIER keyword 187
BKGRND 301
BRIF service, description 23
Browse Interface service 23
BROWSE service
  description 17, 28
  recursive use 28
buf-len 302, 303
buffer 302, 303


## C


C
  example 11
  general call format 11
call
  general format 5, 8
  positional parameters 5
call interfaces
  ISPEXEC 3, 7
  ISPLINK 5
call invocation
  basic interfaces 5
  general call format
    APL2 9
    COBOL 5, 10
    FORTRAN 8
    ISPLINK 13
    Pascal 8
    PL/I 5, 10
  parameters
    as symbolic variables 7
    omitting 5
    positional 5, 8
    types of 6
CALL ISPEXEC interface 7
CALL ISPLINK interface 5
CANCEL mode, effect on error processing 14
change row in table
  TBMOD 237
  TBPUT 240
character_set 301
character-vector 9
clear table variables to nulls (TBVCLEAR) 265
CLIST
  attention exits 4
  variables, used in command invocation 4
close and save table (TBCLOSE) 201
close table without saving (TBEND) 231
closing a data set 98
COBOL
  call format using ISPEXEC 7
  example 5, 10
  general call format 5
  high-order bit generation 5
  literals
    in assignment statements 10
    in call statements, not allowed 10
  return codes from services 13, 14
  RETURN-CODE built-in variable 13
codepage 301
coding requests for services
  keyword parameter 6
  numeric value parameter 6
  service name parameter 6
column of a table, defining 17
command call
  general format 3
  positional parameters 4
command invocation
  CLIST variables 3
  dialog variables as parameters 5
  general format 3
  ISPEXEC command 3
  Option 7.6, Dialog Services 3
  parameter conventions 3
  return codes 13
  variables 4
command routines and I/O, return codes from 15
commands
  nesting xxii
commands, definition of application 231
compressing a data set 99
COND keyword on SETMSG 15
CONTROL service
  ADDPOP/REMPOP service relation 32
  description 19, 31
  example 37
copying
  a variable (VCOPY) 267
  members of a data set 101
  variables to a shared pool or profile pool (VPUT) 297
create a new table (TBCREATE) 203
creating a member list 129, 145
CRP, movement of
  TBBOTTOM 199
  TBDISPL 209
  TBSCAN 250
  TBSKIP 255
  TBTOP 264
CSRROW (.CSRROW) control variable 216


## D


data id
  definition of 119
  generating 119
Data Set Display Service 104
Data Set Information Dialog Service, DSINFO 44
Data Set Information Panel, data set list dialog 104
DBCS
  defining search argument 251
  sort sequence 258
DBCS considerations
  LMGET 117
  LMPUT 165
ddname interface 30, 59, 286
define function variable (VDEFINE) 268
delete (set to nulls) table values (TBVCLEAR) 265
Delete option of LMMDISP 139


© Copyright IBM Corp. 1980, 2004

delete row from table (TBDELETE) 206
delete, a table (TBERASE) 232
dialog
  example 223
  service description 2, 21
dialog function, example 223
DISPLAY environment
  EDIT service 55
  VIEW service 283
DISPLAY service
  description 15, 38
  example 42
display services 15
distributed editing 55
DSINFO 44

## E

EDIF service
  description of 46
  recovery mode 47, 289
EDIREC service, description of 53
Edit interface service 46
edit macros, ISPF/PDF services in 1
Edit profile 189
edit recovery
  VIEW service 283
edit recovery table
  initialization of 53, 62
  scanning for pending recovery 53, 62
EDIT service
  description 17, 54
  recovery mode 54
  recursive use 54
  running in a pop-up window 55
edit, distributed 55
EDREC service
  CANCEL option 63
  DEFER option 64
  description 17, 61
  INIT option 62
  PROCESS option 63
  QUERY option 62
ending, file tailoring (FTCLOSE) 69
ENQ issued by TBOPEN 238
erase (set to nulls) table variables (TBVCLEAR) 265
erase a table (TBERASE) 232
erasing
  member of file tailoring output library (FTERASE) 70
  variables from shared or profile pool (VDELETE) 279
erasing a data set 112
error modes (return code of 12 or higher)
  CANCEL 14
  RETURN 14
exit routine, VDEFINE service 275
exits, CLIST attention 4

## F

FI 301
file tailoring services 15
FILESTAT 64
FILESTAT service 19
FILEXFER service 19, 65
find table variable TBSARG 244
finding a library member 141
FORTRAN
  example 8
  general call format 5, 8
  high-order bit generation 5
  ISPEX alternate name 5
  ISPLNK alternate name 5
  lastrc variable 8
  passing arguments 8
  return code variable 13
  return codes from services 13, 14
  variable names 4, 5
FRAME 301
freeing a data set from association with a data ID 115
FTCLOSE service
  description 69
  example 69, 70
FTERASE service
  description 70
  example 71
FTINCL service
  description 72
  example 72
FTOPEN service
  description 73
  example 74
function commands, definition 231
function variable pool, LMMDISP, variable saved 129
function variables, define in function pool (VDEFINE) 269

## G

generic search argument, specification of
  TBSARG 244
  TBSCAN 250
get a copy of variable (VCOPY) 267
get row from table (TBGET) 235
get variable from shared pool or profile pool(VGET) 281
GETMSG service
  description 19, 74
  example 76
graphics interface mode, for 3290 terminal 78
GRERROR service 76
GRINIT service
  description 77
  example 79
GRTERM service 79

## I

I/O and command routines, return codes from 15
including file tailoring skeleton (FTINCL) 72
initializing edit recovery 53
invoking
  dialog management services 2
  services 21
invoking a dialog (SELECT) 185
ip_var_name 301
ISPEX, alternate call interface name for FORTRAN and Pascal 5
ISPEX, call interface 5, 8
ISPEXEC
  call interface 3, 5, 7
  command invocation 3, 7
  using DM services 2
ISPF library, defined 17
ISPF, ISPQRY, testing if active 2
ISPF/PDF services
  BROWSE 17
  command invocation 3
  description of 17
  EDIF service 17
  EDIT 17
  EDREC 17
  introduction to 1
  invoking 2
  load module search order 2
  notation conventions 2
  prerequisites 1
  with dialog management service 3
ISPFILU ddname 81
ISPLINK
  call interface 5, 8
  parameters 5, 14
ISPLINK routine, invoking DM services 2
ISPLNK
  alternate call interface name for FORTRAN 5
  alternate call interface name for Pascal 5
  call interface 5
  parameters 8
ISPLUSR ddname 81
ISPMUSR ddname 81
ISPPUSR ddname 81
ISPQRY, testing if active 2
ISPSUSR ddname 81
ISPTABU ddname 81
ISPTUSR ddname 81

## J

jump function xxii

## K

keyword parameter 4
keyword parameter, coding requests for services 6

## L

LANG keyword 187
  CREX parameter 187
lastrc variable
  APL2 9
  FORTRAN 8
  Pascal 8
LIBDEF null statement 81, 89
LIBDEF service 19, 79
library
  opening 14

library *(continued)*
renaming 171
library access services 16
line length on LIST service 95
line mode
automatic entry 32, 187
non-automatic entry 32, 187
list data set, writing to 19, 93
LIST service description 19, 93
LMCLOSE 16
LMCLOSE service, description 98
LMCOMP 16
LMCOPY 16
LMCOPY service, description 101
LMDDISP 16
LMDDISP service, description 104
LMDFREE 16
LMDFREE service, description 106, 115
LMDINIT 16
LMDINIT service, description 107
LMDLIST 16
LMDLIST service, description 109
LMERASE 16
LMERASE service, description 112
LMFREE 16
LMFREE service, description 115
LMGET service
DBCS considerations 117
description 16, 116
LMINIT
ddname
to BROWSE 30
to EDIT 59
to VIEW 286
description 16, 119
LMMADD 16
LMMADD service
description 124, 129
statistical variables 125, 129
ZLMSEC 126
LMMDEL service 127
LMMDISP 16
LMMFIND 16
LMMFIND service
description 141
statistical variables 143
LMMLIST 16
LMMLIST service
description 145
FREE option 146
LIST option 147
statistical variables 146
LMMOVE 16, 149
LMMOVE service, description 149
LMMREN 16, 152
LMMREN service, description 152
LMMREP 16, 154
LMMREP service
ZLMSEC 155
LMMSTATS 16
LMMSTATS service, description 156
LMOPEN 16
LMOPEN service
description 161
INPUT/OUTPUT options 160
LMPRINT 16
LMPRINT service, description 162
LMPUT 16, 99
LMPUT service
DBCS considerations 164
description 165
LMQUERY 16
LMQUERY service, description 167
LMRENAME 16
LMRENAME service, description 171
LMREP service
description 154
statistical variables 155
LMxxxxxx - library access services 16
LNCT Search-For process statement 16
load module search order 3
LOG service
description 19, 173
example 174
logging a message (LOG service) 19
LookAt message retrieval tool xi
lu_var_name 301

# M

mask association with dialog variables (VMASK) 294
member
copying 101
deleting 127
erasing 112
finding 141
renaming 152
replacing 154
member list
adding a member 137
creating 145
dialog variables saved 129
displaying 130
freeing storage space associated with 140, 145
getting the next member 133, 148
putting information in the line command field and the user data field 135
Member List Dialog Service, MEMLIST 174
MEMLIST 174
message library, LIBDEF definition 86
message logging (LOG service) 19
message retrieval tool, LookAt xi
messages, setting (SETMSG) 193
model sets, example 207, 223
modify a table row
TBMOD 237
TBPUT 240
move current row pointer (CRP) 255
TBBOTTOM 255
TBSCAN 255
TBSKIP 255
TBTOP 255, 256
moving data set members 149
MULTX mode
on LMGET service 116
on LMPUT service 164

# N

naming restrictions for dialog functions 188
National Language Support (NLS), for numeric representation 258
NEST keyword 187
nested commands xxii
NEWAPPL
data element search order 83
description of command procedures 189
NOGUIDSP 301
notation conventions 2
numeric value parameter, coding requests for services 6

# O

open a table (TBOPEN) 238
open and create a table (TBCREATE) 203
opening a data set 160
opening skeleton files (FTOPEN) 73

# P

page eject on list data set 95
PANEL 302
panel definition, used by TBDISPL 15
parameters
coding rules for service requests 3, 6
specified as variables 10
used as symbolic variables 7
partition mode for 3290 terminal 78
Pascal
general call format 8
ISPEX alternate name 5
ISPLINK alternate name 5
lastrc variable 8
passing arguments as variables or literals 9
return code registers 13
return code variable 13
variable names 4, 5
PASSLIB
data element search order 83
description of command procedures 190
PDF services, with edit macros 1
percent (%) sign, beginning a command with 32, 187
PL/I
call format using ISPEXEC 7
call format using ISPLINK 5
example of statements you should use 10
high-order bit generation 5, 7
PLIRETV build-in function should use 13
return codes 13
return codes from services 13, 14
using literals in assignment statements 10
PLIRETV build-in function 13, 14
pop-up window, and EDIT service 55
POSITION, TBDISPL parameter 216

positional parameters, command invocation 3
PQUERY service, description 19, 176
printing data sets 162
Programming Control Facility (PCF) 29
put variables in shared pool or profile pool (VPUT) 297

# Q

QBASELIB 178
QBASELIB service 19
QLIBDEF service 19
QLIBDEF service, description 180
QTABOPEN service 19, 181
Query Base Library Information, QBASELIB 178
QUERYENQ – Query System ENQ Data 182
QUERYENQ service 19

# R

RACF (Resource Access Control Facility) 29
read a table into virtual storage (TBOPEN) 238
reading a data set record 116
reading row from table
  TBBOTTOM 199
  TBGET 235
  TBSCAN 250
reinitialization section of panel definition, panel processing considerations 40
remove definition of variables from function pool
  VDELETE 278
  VRESET 299
REMPOP service
  description 19, 184
  relation to control service 32
  return codes 182, 185
renaming a member 152
renaming an ISPF library 171
replace a data set member 154
replace variable in function pool (VREPLACE) 298
reset table variables to nulls (TBVCLEAR) 265
reset variables 299
restrictions on member expansion and member part lists
  I/O and command routines (return codes) 12
  service (return codes) 15
retrieve variables from shared pool or profile (VGET) 281
retrieving a row from table
  TBBOTTOM 199
  TBDISPL 209
  TBGET 235
  TBSCAN 250
  TBSKIP 255
return codes
  from services 12
  I/O and command routines 15
RETURN mode, effect on error processing 14
RETURN-CODE
  COBOL built-in variable 13
  system variables to format error messages 14
row deletion (TBDELETE) 206
row table services 18
row, determine existence (TBEXIST) 233
ROWID, TBDISPL parameter 216
rows of a table, content 17

# S

save and close table (TBCLOSE) 201
save table (TBSAVE) 248
search argument, specification of TBSARG 244
search, a table (TBSCAN) 250
SELECT command
  NEWAPPL 185
  PASSLIB parameter 190
SELECT service
  ADDPOP parameter 15, 186
  BARRIER keyword 187
  description 185
  example 192
  LANG keyword 187
  NEST keyword 187
service call, general call format 5
service interface routines 2, 5
service name parameter, coding on service requests 6
services
  command procedure format 1
  description 1, 250
  FILESTAT 19
  FILEXFER 19, 65
  QBASELIB 19
  QLIBDEF 19
  QTABOPEN 19
  QUERYENQ 19
  TRANS 19
  WSCON 19
  WSDISCON 19
services description
  CONTROL 19
  display 15
  file tailoring 15
  LOG 19
  PQUERY 19
  table 17
SETMSG service
  description 15, 193
  example 195
setting row pointer
  TBBOTTOM 199
  TBDISPL 199, 209
  TBSCAN 199, 250
  TBSKIP 199
  TBTOP 264
setting, processing modes (CONTROL) 31
SETTINGS option, affect on LIST service 95
single name parameter, coding on request for services 6
SISPSASC 3
sort information record 257
spacing on list data set 95
SSI, returning value of 130, 143
statistical information
  setting and storing statistics 156
  variables
    LMMADD 125
    LMMFIND 143
    LMMLIST 146
    LMMREP 155
storing statistics 156
suspending an activity xxii
syntax rules, services requests (parameters) 6

# T

table
  adding or updating information 196
  columns 17
  definition 17
  rows description 17
table display (TBDISPL) 207
table services
  description 17
  general services 17
  row services 18
TBADD service
  description 196
  example 198
TBBOTTOM service
  description 199
  example 200
TBCLOSE service
  description 201
  example 203, 224
TBCREATE service
  description 203
  example 205
TBDELETE service
  description 206
  example 206
TBDISPL service
  control variables related to 216
  description 15, 207
  example 214, 223
  hints, tips, and techniques 218
  notes about 229
  system variables related to 214
  use with other services 216
  using 207
TBEND service
  description 231
  example 232
TBERASE service
  description 232, 233
TBEXIST service
  description 233
  example 234
TBGET service
  description 235
  example 236
TBMOD service
  description 237
  example 238

TBOPEN service
description 238
example 224, 240
TBPUT service
description 240
example
command procedure function 224
using function variable pool values 241
using with TBDISPL service 223
TBQUERY service
description 242
example 244
TBSARG service
description 244
example 247
TBSAVE service
description 248
example 250
TBSCAN service
description 250
example 254
TBSKIP service
description 255
example 256
TBSORT service
description 257
example 259
TBSTATS service
description 260
example 263
TBTOP service
description 264
example 264
TBVCLEAR service
description 265
example 265
title_var_name 301
TRANS service 19
TRANS service, description 266
translate CCSID data (TRANS) 266

# U

update row in table
TBMOD 237
TBPUT 240
update variables in shared pool or profile pool (VPUT) 297
use count
TBCLOSE (close and save a table) 201
TBCREATE (create a new table) 205
TBEND (close a table without saving) 231

# V

variable model lines, use 220
variable services summary 18
variables
associate edit mask with (VMASK) 294
clearing to nulls (TBVCLEAR) table 265
copy (VCOPY) 267
variables *(continued)*
define in function pool (VDEFINE) 269
erase from shared profile pool (VERASE) 279
passed as parameter to services 4
remove definition from function pool (VRESET) 299
remove definition of from function pool (VDELETE) 278
replace in function pool (VREPLACE) 298
reset 299
retrieve from shared pool or profile pool (VGET) 281
TBDISPL parameters 216
update in shared pool or profile pool (VPUT) 297
VCOPY service
description 267
example 268
used to access system variables 267
VDEFINE service
description 268
example 274, 275, 277
exit routine 275
VDELETE service
description 278
example 279
VERASE service
description 18, 279
example 280
using 279
VGET service
accessing 281
accessing application profile pool 282
View Interface service 288
VIEW service
description 282
recovery mode 283
recursive use 282
VIIF service 288
VL keyword assembler language 5, 7
VMASK service
description 294
example 297
VPUT service
accessing application profile pool 297
accessing read-only extension 298
VREPLACE service
description 298
example 299
VRESET service
description 299
example 300

# W

workstation command 188
workstation command var 189
Workstation Connection
WSCON service 300
Workstation Disconnection
WSDISCON service 303
Write data set list dialog 104
writing a message to log file (LOG) 173
writing a record to a data set 164
WSCMD 188
WSCMDV 189
WSCON 300
WSCON service 19
BKGRND 301
buf-len 302
buffer 302
character_set 301
codepage 301
FI 301
FRAME 301
ip_var_name 301
lu_var_name 301
NOGUIDSP 301
PANEL 302
title_var_name 301
WSDISCON 303
WSDISCON service 19
buf-len 303
buffer 303

# Z

ZEDBDSN 63
ZEDROW 63
ZEDTDSN 63
ZEDTMEM 63
ZEDTRD 63
ZEDUSER 55, 283
ZEDUSER extension variable 63
ZEIUSER extension variable 54
ZERRALRM 14
ZERRALRM system variable 14
ZERRHM 14
ZERRHM system variable 14
ZERRLM 14
ZERRLM system variable 14
ZERRMSG 14
ZERRMSG system variable 14
ZERRSM 14
ZERRSM system variable 14
ZLC4DATE 126, 130
ZLCDATE 125, 130
ZLCNORC 125, 130
ZLINORC 126, 130
ZLLCMD 130
ZLLIB 130, 143
ZLM4DATE 126, 130
ZLMDATE 125, 130
ZLMEMBER 130
ZLMNORC 126, 130
ZLMOD
LMMADD, add a member to a data set 125
LMMDISP, member list service 130
LMMFIND, find a library member 144
LMMREP, replace a member of a data set 155
ZLMSEC 126, 130, 144, 155
ZLMTIME 125, 130
ZLMTOP 130
ZLPDSUDA 130
ZLSSI 130, 143
ZLSTLPP system variable 96

ZLSTNUML 96
ZLSTTRUN system variable 95
ZLUDATA 130
ZLUSER 126, 130
ZLVERS 125, 130
ZTDMARK system variable 214
ZTDMSG system variable 215
ZTDROWS system variable 215
ZTDSELS system variable 215
ZTDTOP system variable 209, 215
ZTEMPF system variable 73

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

**Interactive System Productivity Facility (ISPF)**
**Services Guide**
**z/OS Version 1 Release 6.0**

**Publication No. SC34-4819-03**

**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**
SC34-4819-03

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
Department J87/D325
555 Bailey Avenue
San Jose, CA
U.S.A. 95141-9989

Fold and Tape | **Please do not staple** | Fold and Tape



Cut or Fold
Along Line

File Number: S370/4300-39
Program Number: 5694-A01

Printed in USA

SC34-4819-03