

z/OS  
2.5

*ISPF Services Guide*



**Note**

Before using this information and the product it supports, read the information in [“Notices” on page 383](#).

This edition applies to Version 2 Release 5 of z/OS® (5650-ZOS) and to all subsequent releases and modifications until otherwise indicated in new editions.

Last updated: 2023-06-25

© **Copyright International Business Machines Corporation 1980, 2023.**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

---

# Contents

<b>Figures.....</b>	<b>xv</b>
<b>Tables.....</b>	<b>xvii</b>
<b>Preface.....</b>	<b>xix</b>
Who should use this document?.....	xix
What is in this document?.....	xix
How to read the syntax diagrams.....	xix
<b>z/OS information.....</b>	<b>xxiii</b>
<b>How to send your comments to IBM.....</b>	<b>xxv</b>
If you have a technical problem.....	xxv
<b>Summary of changes.....</b>	<b>xxvii</b>
Summary of changes for z/OS Version 2 Release 5 (V2R5).....	xxvii
Summary of changes for z/OS Version 2 Release 4 (V2R4).....	xxviii
Summary of changes for z/OS Version 2 Release 3 (V2R3).....	xxviii
<b>What's in the library?.....</b>	<b>xxxi</b>
<b>Chapter 1. Introduction to ISPF services.....</b>	<b>1</b>
Description of the services.....	1
Using ISPQRY to test whether ISPF is active.....	1
Invoking the ISPF services.....	2
Load module search order.....	2
Invoking services from command procedures.....	3
Invoking ISPF services with program functions.....	4
Return codes from services.....	11
Command invocation return code variable.....	12
Call invocation return code variables.....	12
Return code of 12 or higher.....	12
System variables used to format error messages.....	13
Return codes from I/O and command routines.....	13
A summary of the ISPF services.....	14
Display services.....	14
File tailoring services.....	14
Library access services.....	14
PDF component services.....	16
Table services.....	16
Variable services.....	17
Miscellaneous services.....	18
<b>Chapter 2. Description of the ISPF services.....</b>	<b>21</b>
ADDPop—start pop-up window mode.....	21
Command invocation format.....	21
Call invocation format .....	21
Parameters.....	22

Return codes.....	22
Example.....	22
BRIF—Browse interface.....	23
Command invocation format.....	24
Call invocation format.....	24
Parameters.....	24
Dialog-supplied routines.....	25
Return codes.....	27
Example.....	27
BROWSE—browse a data set.....	28
Command invocation format.....	28
Call invocation format.....	29
Parameters.....	29
Return codes.....	30
Example.....	31
CONTROL—set processing modes.....	31
Command invocation format.....	32
Call invocation format.....	32
ADDPPOP/REMPPOP service in relation to CONTROL service.....	33
Parameters.....	34
Return codes.....	38
Examples.....	38
DIRLIST—directory list service .....	39
Command invocation format.....	40
Call invocation format.....	40
Parameters.....	40
Return codes.....	44
Example.....	44
DISPLAY—display panels and messages.....	45
Command invocation format.....	45
Call invocation format.....	45
Parameters.....	46
Using the COMMAND Option.....	47
Return codes.....	48
Examples.....	48
DSINFO—data set information dialog service.....	51
Command invocation format.....	51
Call invocation format.....	51
Parameters.....	51
Return codes.....	53
Example.....	54
EDIF—Edit interface.....	54
Command invocation format.....	55
Call invocation format.....	55
Parameters.....	56
Dialog-supplied routines.....	57
Return codes.....	59
Example.....	60
EDIREC - Initialize Edit Recovery.....	60
Command invocation format.....	60
Call invocation format.....	61
Parameters.....	61
Return codes.....	62
Example.....	62
EDIT—edit a data set.....	62
Command invocation format.....	63
Call invocation format.....	65
Parameters.....	65

Return codes.....	68
Examples.....	68
EDREC—specify edit recovery handling.....	69
Command invocation format.....	70
Call invocation format.....	70
Parameters.....	70
Return codes.....	72
Examples.....	72
FTCLOSE—end file tailoring.....	73
Command invocation format.....	74
Call invocation format.....	74
Parameters.....	74
Return codes.....	74
Example.....	75
FTERASE—erase file tailoring output.....	75
Command invocation format.....	75
Call invocation format.....	75
Parameters.....	75
Return codes.....	76
Example.....	76
FTINCL—include a skeleton.....	76
Command invocation format.....	76
Call invocation format.....	77
Parameters.....	77
Return codes.....	77
Example.....	77
FTOPEN—begin file tailoring.....	78
Command invocation format.....	78
Call invocation format.....	78
Parameters.....	78
Return codes.....	79
Example.....	79
GETMSG—get a message.....	79
Command invocation format.....	80
Call invocation format.....	80
Parameters.....	80
Return codes.....	81
Example.....	81
GRERROR—graphics error block service.....	81
Command invocation format.....	81
Call invocation format.....	81
Parameters.....	82
Return codes.....	82
GRINIT—graphics initialization.....	82
Command invocation format.....	83
Call invocation format.....	83
Parameters.....	83
Return codes.....	83
Example.....	83
GRTERM—graphics termination service.....	83
Command invocation format.....	84
Call invocation format.....	84
Return codes.....	84
LIBDEF—allocate application libraries.....	84
Application data element search order.....	85
Command invocation format.....	87
Call invocation format.....	87
Parameters.....	88

Usage notes.....	90
Return codes.....	93
Examples.....	94
LIST—write lines to the list data set.....	98
Command invocation format.....	98
Call invocation format.....	98
Parameters.....	99
Return codes.....	99
Formatting data to be written to the list data set.....	100
List data set characteristics affect the LIST service.....	100
Controlling line spacing, page eject, and highlighting.....	100
How carriage control characters affect truncation.....	101
Examples.....	102
LMCLOSE—close a data set.....	103
Command invocation format.....	103
Call invocation format.....	103
Parameters.....	103
Return codes.....	104
Example.....	104
LMCOMP—compresses a partitioned data set.....	104
Command invocation format.....	104
Call invocation format.....	104
Parameters.....	105
Return codes.....	105
Example.....	105
LMCOPY—copy members of a data set.....	106
Command invocation format.....	107
Call invocation format.....	107
Parameters.....	107
Return codes.....	108
Example.....	109
LMDDISP—data set list service.....	109
Command invocation format.....	110
Call invocation format.....	110
Parameters.....	111
Return codes.....	112
Example.....	112
LMDFREE—free a data set list ID.....	112
Command invocation format.....	112
Call invocation format.....	113
Parameters.....	113
Return codes.....	113
Example.....	113
LMDINIT—initialize a data set list.....	114
Command invocation format.....	114
Call invocation format.....	114
Parameters.....	114
Return codes.....	115
Examples.....	115
LMDLIST—list data sets.....	117
Command invocation format.....	117
Call invocation format.....	118
Parameters.....	118
Return codes.....	120
Examples.....	121
LMERASE—erase a data set.....	122
Command invocation format.....	123
Call invocation format.....	123

Parameters.....	123
Return codes.....	124
Example.....	124
LMFREE—free data set from its association with data ID.....	125
Command invocation format.....	125
Call invocation format.....	125
Parameters.....	125
Return codes.....	125
Example.....	126
LMGET—read a logical record from a data set.....	126
Command invocation format.....	127
Call invocation format.....	127
Parameters.....	127
Return codes.....	128
Example 1.....	128
Example 2.....	129
Example 3 (MULTX).....	130
LMINIT—generate a data ID for a data set.....	130
Command invocation format.....	131
Call invocation format.....	131
Parameters.....	132
Return codes.....	133
Examples.....	134
LMMADD—add a member to a data set.....	135
Command invocation format.....	135
Call invocation format.....	136
Parameters.....	136
Return codes.....	138
Example.....	138
LMMDEL—delete members from a data set.....	139
Command invocation format.....	139
Call invocation format.....	139
Parameters.....	139
Return codes.....	140
Example.....	140
LMMDISP—member list service.....	140
Dialog variables.....	141
DISPLAY option.....	142
GET option.....	146
PUT option.....	147
ADD option.....	149
DELETE option.....	151
FREE option.....	153
LMMFIND—find a library member.....	154
Command invocation format.....	154
Call invocation format.....	154
Parameters.....	155
Return codes.....	157
Example.....	158
LMMLIST—list a library's members.....	158
Command invocation format.....	159
Call invocation format.....	159
Parameters.....	159
Return codes.....	160
Examples.....	161
LMMOVE—move members of a data set.....	163
Command invocation format.....	164
Call invocation format.....	164

Parameters.....	164
Return codes.....	165
Example.....	166
LMMREN—rename a data set member.....	166
Command invocation format.....	167
Call invocation format.....	167
Parameters.....	167
Return codes.....	167
Example.....	168
LMMREP—replace a member of a data set.....	168
Command invocation format.....	168
Call invocation format.....	169
Parameters.....	169
Return codes.....	171
Example.....	171
LMMSTATS—set and store, or delete ISPF statistics.....	171
Command invocation format.....	172
Call invocation format.....	173
Parameters.....	173
Return codes.....	175
Example.....	175
LMOPEN—open a data set.....	176
Command invocation format.....	176
Call invocation format.....	177
Parameters.....	177
Return codes.....	177
Example.....	178
LMPRINT—print a partitioned or sequential data set.....	178
Command invocation format.....	179
Call invocation format.....	179
Parameters.....	179
Return codes.....	179
Example.....	180
LMPUT—write a logical record to a data set.....	180
Command invocation format.....	181
Call invocation format.....	181
Parameters.....	181
Return codes.....	182
Example.....	182
Example (MULTX).....	183
LMQUERY—give a dialog information about a data set.....	183
Command invocation format.....	184
Call invocation format.....	184
Parameters.....	184
Return codes.....	186
Example.....	186
LMRENAME—rename an ISPF library.....	187
Command invocation format.....	187
Call invocation format.....	187
Parameters.....	187
Return codes.....	188
Example.....	188
LOG—write a message to the log data set.....	189
Command invocation format.....	189
Call invocation format.....	189
Parameters.....	189
Return codes.....	189
Example 1.....	190



Example 2.....	190
Example 3.....	190
MEMLIST—member list dialog service.....	190
Command invocation format.....	191
Call invocation format.....	191
Parameters.....	191
Return codes.....	192
Example.....	192
PQUERY—obtain panel information.....	193
Command invocation format.....	193
Call invocation format.....	193
Parameters.....	193
Return codes.....	194
Example.....	194
QBASELIB—query base library information.....	195
Command invocation format.....	195
Call invocation format.....	195
Parameters.....	195
Return codes.....	195
Example.....	196
QLIBDEF—query LIBDEF definition information.....	196
Command invocation format.....	196
Call invocation format.....	196
Parameters.....	196
Return codes.....	197
Example.....	197
QTABOPEN—query open ISPF tables.....	198
Command invocation format.....	198
Call invocation format.....	198
Parameters.....	198
Return codes.....	198
Example.....	198
QUERYENQ—query system ENQ data.....	198
Command invocation format.....	199
Call invocation format.....	199
Parameters.....	199
Variables returned in each row of the table.....	200
Return codes.....	200
REMPop—remove a pop-up window.....	201
Command invocation format.....	201
Call invocation format.....	201
Parameters.....	201
Return codes.....	201
SELECT—select a panel or function.....	201
Command invocation format.....	202
Call invocation format.....	202
Parameters.....	203
Return codes.....	207
Examples.....	207
SETMSG—set next message.....	208
Command invocation format.....	209
Call invocation format.....	209
Parameters.....	209
Return codes.....	209
Example 1.....	210
Example 2.....	210
TBADD—add a row to a table.....	210
Command invocation format.....	211

Call invocation format.....	211
Parameters.....	211
Return codes.....	212
Example 1.....	212
Example 2.....	213
TBBOTTOM—set the row pointer to bottom.....	213
Command invocation format.....	213
Call invocation format.....	213
Parameters.....	213
Return codes.....	214
Example.....	214
TBCLOSE—close and save a table.....	215
Command invocation format.....	215
Call invocation format.....	215
Parameters.....	216
Return codes.....	216
Example.....	217
TBCREATE—create a new table.....	217
Command invocation format.....	217
Call invocation format.....	217
Parameters.....	218
Return codes.....	219
Examples.....	219
TBDELETE—delete a row from a table.....	220
Command invocation format.....	220
Call invocation format.....	220
Parameters.....	220
Return codes.....	220
Example.....	221
TBDISPL—display table information.....	221
TBDISPL operation.....	222
Operational results from user actions.....	222
Command invocation format.....	223
Call invocation format.....	223
Parameters.....	224
Parameter processing.....	225
Return codes.....	226
Example.....	227
System variables related to TBDISPL.....	227
Panel control variables related to TBDISPL.....	229
Parameter variables related to TBDISPL.....	229
Using TBDISPL with other services.....	229
Techniques for using the TBDISPL service.....	230
Rules applying to variable model lines.....	232
Example—using the TBDISPL and TBPOT services.....	237
TBDISPL summary.....	242
TBEND—close a table without saving.....	245
Command invocation format.....	246
Call invocation format.....	246
Parameters.....	246
Return codes.....	246
Example.....	246
TBERASE—erase a table.....	246
Command invocation format.....	247
Call invocation format.....	247
Parameters.....	247
Return codes.....	247
Example.....	247

TBEXIST—determine whether a row exists in a table.....	248
Command invocation format.....	248
Call invocation format.....	248
Parameters.....	248
Return codes.....	248
Example.....	248
TBGET—retrieve a row from a table.....	249
Command invocation format.....	249
Call invocation format.....	249
Parameters.....	250
Return codes.....	250
Example.....	250
TBMOD—modify a row in a table.....	251
Command invocation format.....	251
Call invocation format.....	251
Parameters.....	251
Return codes.....	252
Example.....	252
TBOPEN—open a table.....	252
Command invocation format.....	253
Call invocation format.....	253
Parameters.....	253
Return codes.....	254
Example.....	254
TBPUT—update a row in a table.....	254
Command invocation format.....	255
Call invocation format.....	255
Parameters.....	255
Return codes.....	255
Example.....	256
TBQUERY—obtain table information.....	256
Command invocation format.....	256
Call invocation format.....	256
Parameters.....	257
Return codes.....	258
Example.....	258
TBSARG—define a search argument.....	258
Command invocation format.....	259
Call invocation format.....	259
Parameters.....	260
Return codes.....	261
Examples.....	261
TBSAVE—save a table.....	262
Command invocation format.....	262
Call invocation format.....	262
Parameters.....	262
Return codes.....	263
Example.....	264
TBSCAN—search a table.....	264
Command invocation format.....	265
Call invocation format.....	265
Parameters.....	265
Return codes.....	267
Examples.....	267
TBSKIP—move the row pointer.....	268
Command invocation format.....	268
Call invocation format.....	268
Parameters.....	269

Return codes.....	269
Example.....	270
TBSORT—sort a table.....	270
Command invocation format.....	271
Call invocation format.....	271
Parameters.....	271
Return codes.....	272
Example 1.....	272
Example 2.....	273
TBSTATS—retrieve table statistics.....	273
Command invocation format.....	274
Call invocation format.....	274
Parameters.....	275
Return codes.....	277
Example.....	277
TBTOP—set the row pointer to the top.....	277
Command invocation format.....	277
Call invocation format.....	277
Parameters.....	278
Return codes.....	278
Example.....	278
TBVCLEAR—clear table variables.....	278
Command invocation format.....	278
Call invocation format.....	279
Parameters.....	279
Return codes.....	279
Example.....	279
TRANS—translate CCSID data.....	279
Command invocation format.....	280
Call invocation format.....	280
Parameters.....	280
Return codes.....	281
VCOPY—create a copy of a variable.....	281
Command invocation format.....	281
Call invocation format.....	281
Parameters.....	282
Return codes.....	282
Example.....	282
VDEFINE—define function variables.....	283
Command invocation format.....	283
Call invocation format.....	283
Parameters.....	283
Return codes.....	287
Examples.....	287
VDEFINE exit routine.....	288
VDELETE—remove a definition of function variables.....	291
Command invocation format.....	291
Call invocation format.....	291
Parameters.....	292
Return codes.....	292
Example.....	292
VERASE—remove variables from shared or profile pool.....	292
Command invocation format.....	292
Call invocation format.....	292
Parameters.....	293
Return codes.....	293
Example.....	293
VGET—retrieve variables from a pool or profile or system symbol.....	294

Command invocation format.....	294
Call invocation format.....	294
Parameters.....	294
Return codes.....	295
Examples.....	296
VIEW—view a data set.....	296
Command invocation format.....	297
Call invocation format.....	299
Parameters.....	299
Return codes.....	301
Examples.....	302
VIIF—View interface.....	303
Command invocation format.....	303
Call invocation format.....	303
Parameters.....	304
Dialog-supplied routines.....	306
Return codes.....	308
Example.....	309
VMASK—mask and edit processing.....	309
VMASK call invocation.....	310
Parameters.....	310
Return codes.....	312
Example.....	312
The VEDIT statement.....	312
VPUT—update variables in the shared or profile pool.....	313
Command invocation format.....	313
Call invocation format.....	313
Parameters.....	313
Return codes.....	313
Example.....	314
VREPLACE—replace a variable.....	314
Command invocation format.....	314
Call invocation format.....	314
Parameters.....	315
Return codes.....	315
Example.....	315
VRESET—reset function variables.....	315
Command invocation format.....	315
Call invocation format.....	315
Return codes.....	315
Example.....	316
VSYM—resolve system symbols.....	316
Command invocation format.....	316
Call invocation format.....	316
Parameters.....	316
Return codes.....	316
Example.....	317

## **Appendix A. JSON API.....319**

JSON data structures and variables used to communicate between ISPF and a client.....	319
JSON data structures sent from TSO to client (message type 2).....	319
TSO Message JSON.....	319
TSO prompt JSON.....	320
JSON data structures sent from ISPF to client (message type 3).....	320
ISPF panel display JSON.....	321
ISPF action JSON.....	374
JSON data structures sent from client to TSO (message type 7).....	375

TSO user response JSON.....	375
TSO action request JSON.....	375
JSON data structures sent from client to ISPF (message type 8).....	376
User response JSON.....	376
Client action JSON.....	378
ISPF variables.....	379
<b>Appendix B. Accessibility.....</b>	<b>381</b>
<b>Notices.....</b>	<b>383</b>
Terms and conditions for product documentation.....	384
IBM Online Privacy Statement.....	385
Policy for unsupported hardware.....	385
Minimum supported hardware.....	385
Programming Interface Information.....	386
Trademarks.....	386
<b>Index.....</b>	<b>387</b>

---

# Figures

1. Sample syntax diagram.....	xx
2. Multiple Pop-up Windows.....	23
3. z/OS UNIX Directory List.....	42
4. ISPLIBD - all LIBDEF definitions.....	91
5. ISPLIBD ISPPLIB - ISPPLIB LIBDEF definition.....	92
6. ISPLIBD ISPPLIB - ISPPLIB LIBDEF stacked definition.....	92
7. Variable Model Lines: Panel Definition.....	233
8. Variable Model Lines: Display 1.....	234
9. (Part 1 of 2). Variable Model Lines: Display 1.....	234
10. (Part 2 of 2). Variable Model Lines: Display 2.....	235
11. SFIHDR Keyword in Variable Model Lines: Panel Definition.....	236
12. SFIHDR Keyword in Variable Model Lines: Panel Example 1.....	237
13. SFIHDR Keyword in Variable Model Lines: Panel Example 2.....	237
14. Five Rows in Table TAB1.....	239
15. Table TAB1 as Displayed Using Panel PAN1.....	239
16. Table Display Panel Definition PAN1.....	240





---

# Tables

1. Service Return Codes..... 11

2. ISPF-defined column arrangement..... 41

3. Column abbreviations and widths..... 41

4. Variables used to pass data between ISPF and the line command processor..... 43

5. Dialog variables saved by the DSINFO service..... 51

6. Search Sequence for Libraries..... 86

7. List of dialog variables containing information about a data set..... 119

8. Variables saved by LMMDISP in the function pool..... 141

9. Variables Returned in Each Row of the Table..... 200

10. Decimal Point Representations..... 272

11. ISPF variables used to indicate ISPF is running on behalf of a client..... 379



## 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 this 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.

## How to read the syntax diagrams

---

The syntactical structure of commands described in this document is shown by means of syntax diagrams.

Figure 1 on page xx shows a sample syntax diagram that includes the various notations used to indicate such things as whether:

- An item is a keyword or a variable.
- An item is required or optional.
- A choice is available.
- A default applies if you do not specify a value.
- You can repeat an item.

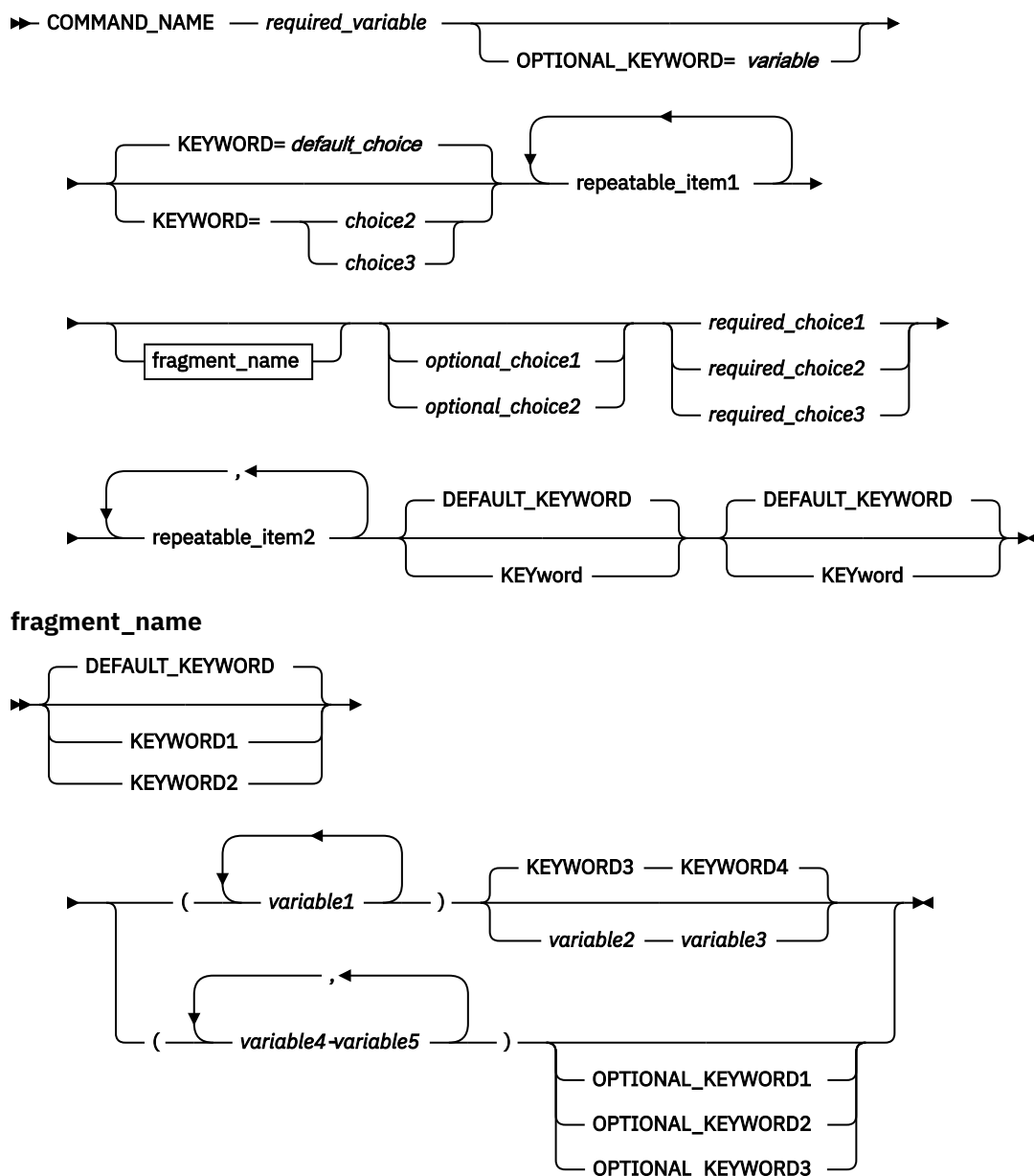


Figure 1. Sample syntax diagram

Here are some tips for reading and understanding syntax diagrams:

### Order of reading

Read the syntax diagrams from left to right, from top to bottom, following the path of the line.

The ➡ symbol indicates the beginning of a statement.

The ➡ symbol indicates that a statement is continued on the next line.

The ➡ symbol indicates that a statement is continued from the previous line.

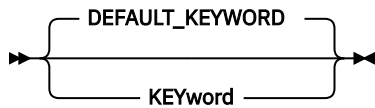
The ➡ symbol indicates the end of a statement.

### Keywords

Keywords appear in uppercase letters.

➡ COMMAND\_NAME ➡

Sometimes you only need to type the first few letters of a keyword, The required part of the keyword appears in uppercase letters.



In this example, you could type "KEY", "KEYW", "KEYWO", "KEYWOR" or "KEYWORD".

The abbreviated or whole keyword you enter must be spelled exactly as shown.

### Variables

Variables appear in lowercase letters. They represent user-supplied names or values.

➡ *required\_variable* ➡

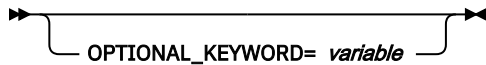
### Required items

Required items appear on the horizontal line (the main path).

➡ **COMMAND\_NAME** — *required\_variable* ➡

### Optional items

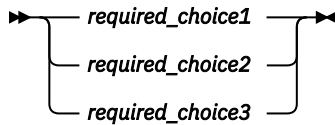
Optional items appear below the main path.



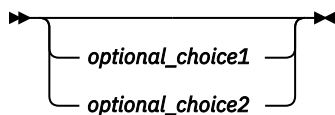
### Choice of items

If you can choose from two or more items, they appear vertically, in a stack.

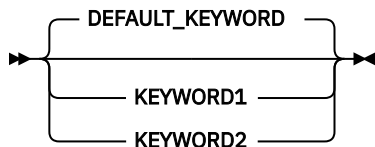
If you *must* choose one of the items, one item of the stack appears on the main path.



If choosing one of the items is optional, the entire stack appears below the main path.



If a default value applies when you do not choose any of the items, the default value appears above the main path.

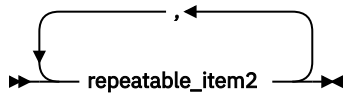


### Repeatable items

An arrow returning to the left above the main line indicates an item that can be repeated.

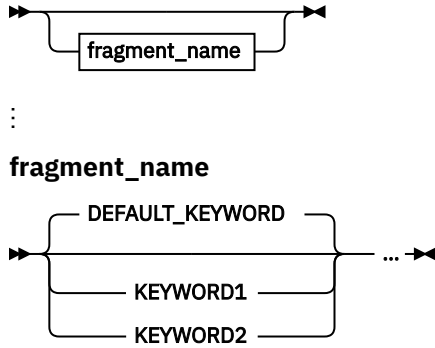


If you need to specify a separator character (such as a comma) between repeatable items, the line with the arrow returning to the left shows the separator character you must specify.



### Fragments

Where it makes the syntax diagram easier to read, a section or *fragment* of the syntax is sometimes shown separately.



### Symbol for blank

This symbol ( ) in syntax diagrams indicates a blank.

## z/OS information

---

This information explains how z/OS references information in other documents and on the web.

When possible, this information uses cross-document links that go directly to the topic in reference using shortened versions of the document title. For complete titles and order numbers of the documents for all products that are part of z/OS, see *z/OS Information Roadmap*.

To find the complete z/OS library, go to [IBM Documentation \(www.ibm.com/docs/en/zos\)](http://www.ibm.com/docs/en/zos).





## How to send your comments to IBM

---

We invite you to submit comments about the z/OS product documentation. Your valuable feedback helps to ensure accurate and high-quality information.

**Important:** If your comment regards a technical question or problem, see instead [“If you have a technical problem”](#) on page xxv.

Submit your feedback by using the appropriate method for your type of comment or question:

### **Feedback on z/OS function**

If your comment or question is about z/OS itself, submit a request through the [IBM RFE Community](#) ([www.ibm.com/developerworks/rfe/](http://www.ibm.com/developerworks/rfe/)).

### **Feedback on IBM® Documentation function**

If your comment or question is about the IBM Documentation functionality, for example search capabilities or how to arrange the browser view, send a detailed email to IBM Documentation Support at [ibmdocs@us.ibm.com](mailto:ibmdocs@us.ibm.com).

### **Feedback on the z/OS product documentation and content**

If your comment is about the information that is provided in the z/OS product documentation library, send a detailed email to [mhvrcfs@us.ibm.com](mailto:mhvrcfs@us.ibm.com). We welcome any feedback that you have, including comments on the clarity, accuracy, or completeness of the information.

To help us better process your submission, include the following information:

- Your name, company/university/institution name, and email address
- The following deliverable title and order number: z/OS ISPF Services Guide, SC19-3626-40
- The section title of the specific information to which your comment relates
- The text of your comment.

When you send comments to IBM, you grant IBM a nonexclusive authority to use or distribute the comments in any way appropriate without incurring any obligation to you.

IBM or any other organizations use the personal information that you supply to contact you only about the issues that you submit.

## If you have a technical problem

---

If you have a technical problem or question, do not use the feedback methods that are provided for sending documentation comments. Instead, take one or more of the following actions:

- Go to the [IBM Support Portal](#) ([support.ibm.com](http://support.ibm.com)).
- Contact your IBM service representative.
- Call IBM technical support.



## Summary of changes

---

This information includes terminology, maintenance, and editorial changes. Technical changes or additions to the text and illustrations for the current edition are indicated by a vertical line to the left of the change.

**Note:** IBM z/OS policy for the integration of service information into the z/OS product documentation library is documented on the z/OS Internet Library under [IBM z/OS Product Documentation Update Policy \(www-01.ibm.com/servers/resourcelink/svc00100.nsf/pages/ibm-zos-doc-update-policy?OpenDocument\)](http://www-01.ibm.com/servers/resourcelink/svc00100.nsf/pages/ibm-zos-doc-update-policy?OpenDocument).

## Summary of changes for z/OS Version 2 Release 5 (V2R5)

---

The following changes are made for z/OS Version 2 Release 5 (V2R5).

### Changed information

- Removal of support for Workstation Agent, see the following topics:
  - [“Load module search order” on page 2](#)
  - [“Library access services” on page 14](#)
  - [“PDF component services” on page 16](#)
  - [“Table services” on page 16](#)
  - [“Variable services” on page 17](#)
  - [“Miscellaneous services” on page 18](#)
  - [“DISPLAY—display panels and messages” on page 45](#)
  - [“EDIT—edit a data set” on page 62](#)
  - [“GRINIT—graphics initialization” on page 82](#)
  - [“LIBDEF—allocate application libraries” on page 84](#)
  - [“QLIBDEF—query LIBDEF definition information” on page 196](#)
  - [“SELECT—select a panel or function” on page 201](#)
  - [“SETMSG—set next message” on page 208](#)
  - [“TBDISPL—display table information” on page 221](#)
  - [“VIEW—view a data set” on page 296](#)
  - [“ISPF panel display JSON examples” on page 325](#)
  - [“ISPF variables” on page 379](#)

### Deleted information

- Removal of support for Workstation Agent:
  - *Example 2: Edit a workstation file*
  - *FILESTAT - statistics for a file*
  - *FILEXFER - upload or download file*
  - *WSCON - connect to a workstation*
  - *WSDISCON - disconnect from a workstation*
  - *Example 2 under “VIEW—view a data set” on page 296*

## Summary of changes for z/OS Version 2 Release 4 (V2R4)

---

The following changes are made for z/OS Version 2 Release 4 (V2R4).

### Changed information

- [“DSINFO—data set information dialog service” on page 51](#)
  - [“Parameters” on page 51](#)
  - [“Return codes” on page 53](#)
- [“LIBDEF—allocate application libraries” on page 84](#)
  - [“Command invocation format” on page 87](#)
  - [“Call invocation format” on page 87](#)
- [“QUERYENQ—query system ENQ data” on page 198](#)
  - [“Parameters” on page 199](#)
- [“DSINFO—data set information dialog service” on page 51](#)
  - [“Parameters” on page 51](#)

## Summary of changes for z/OS Version 2 Release 3 (V2R3)

---

The following changes are made for z/OS Version 2 Release 3 (V2R3).

### June 2019

Maintenance and terminology changes are made for z/OS Version 2 Release 3 in June 2019.

### April 2019

#### Changed information

- [“QUERYENQ—query system ENQ data” on page 198](#)
  - [“Parameters” on page 199](#)
- [“DSINFO—data set information dialog service” on page 51](#)
  - [“Parameters” on page 51](#)
- 

### May 2018

Maintenance and terminology changes are made for z/OS Version 2 Release 3 in May 2018.

### September 2017

#### Changed information

- TSO 8-character ID, see the following topics:
  - [“LMMADD—add a member to a data set” on page 135](#)
  - [“LMMDISP—member list service” on page 140](#)
  - [“LMMFIND—find a library member” on page 154](#)
  - [“LMMREP—replace a member of a data set” on page 168](#)
  - [“LMMSTATS—set and store, or delete ISPF statistics” on page 171](#)
- Extended statistics usability, see the following topics:
  - [“LMMADD—add a member to a data set” on page 135](#)
  - [“LMMDISP—member list service” on page 140](#)

- [“LMMFIND—find a library member” on page 154](#)
- [“LMMREP—replace a member of a data set” on page 168](#)
- [“LMMSTATS—set and store, or delete ISPF statistics” on page 171](#)



## What's in the library?

---

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

**Title**

**Order Number**

***z/OS ISPF Dialog Developer's Guide and Reference***

SC19-3619-40

***z/OS ISPF Dialog Tag Language Guide and Reference***

SC19-3620-40

***z/OS ISPF Edit and Edit Macros***

SC19-3621-40

***z/OS ISPF Messages and Codes***

SC19-3622-40

***z/OS ISPF Planning and Customizing***

GC19-3623-40

***z/OS ISPF Reference Summary***

SC19-3624-40

***z/OS ISPF Software Configuration and Library Manager Guide and Reference***

SC19-3625-40

***z/OS ISPF Services Guide***

SC19-3626-40

***z/OS ISPF User's Guide Vol I***

SC19-3627-40

***z/OS ISPF User's Guide Vol II***

SC19-3628-40





---

# 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 [z/OS 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 [z/OS ISPF Edit and Edit Macros](#).

---

## Description of the services

The services are described in alphabetical order and each service description consists of this 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 [z/OS ISPF Edit and Edit Macros](#) for a description of these facilities.

---

## Using ISPQRY to test whether ISPF is active

A program can determine whether ISPF services are currently available to it through use of ISPQRY. Your syntax will vary depending on the language you are using.

For a PL/I program to test the availability of ISPF, the PL/I function would issue:

```
CALL ISPQRY;
```

Under TSO/E REXX that uses ADDRESS TSO you can use:

```
"ISPQRY"
```

## Invoking the ISPF services

This would use the normal TSO load module search process to find and execute the ISPQRY service.

There are no parameters associated with the call to ISPQRY. No messages are written to the terminal. The response from ISPQRY is one of these 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 this format:

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

or alternatively:

```
lstrc = 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 or a new release of ISPF, and an ISPLLIB is allocated, those data sets allocated to STEPLIB that contain ISPF load modules 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 [z/OS ISPF Dialog Developer's Guide and Reference](#).

If you are using the z/OS UNIX Table Utility, the Language Environment® run-time library data sets SCEERUN and SCEERUN2 must be in STEPLIB or LNKLIST. The modules in these data sets are not searched for in ISPLLIB.

## Invoking services from command procedures

To invoke ISPF services for a command invocation, use either:

- 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.

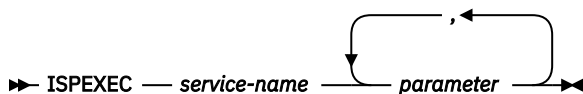
These services are *not* available using the ISPEXEC call from a command procedure:

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:



The command invocation statement must be specified in uppercase.

## ISPEXEC parameter conventions

### service-name

Alphabetic; up to 8 characters long.

### parameter (first)

Positional parameter; required for some services.

### parameter (subsequent)

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 shown below the line. 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. See [z/OS TSO/E CLISTS](#), [z/OS TSO/E REXX User's Guide](#), and [z/OS TSO/E REXX 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 [z/OS TSO/E CLISTS](#) and [z/OS TSO/E Programming Guide](#).

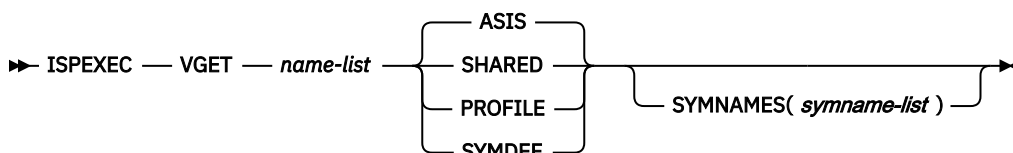
### Passing dialog variables as parameters

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:

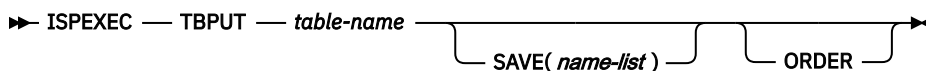


In this case, "name-list" is a positional parameter. It can consist of a list of up to 254 dialog variable names, each name separated by commas or blanks. If the name-list consists of more than one name, it must be enclosed in parentheses. Parentheses can be omitted if a single name constitutes the list.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



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

```
ISPEXEC TBPOT TBLA SAVE(LNAME FNAME,I)
ISPEXEC TBPOT XTBLA 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* , *parameter* ); ➤



## CALL ISPLINK parameters

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

Parameters shown below the line 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.

These 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 this 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')'
```

### routine address

A fullword address indicating the entry point of a routine.

## 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);
```

These 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 these:

```
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' '/'
```

## Invoking the ISPF services

```
⋮  
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 [z/OS 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:.....

      *
      *
```

This 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 these 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

For functions written in COBOL, arguments can be passed as variables or as literals, as in these examples:

## COBOL examples

```
PROCEDURE DIVISION.
    CALL 'ISPLINK' USING BY CONTENT 'TBOPEN ' 'XTABLE ' 'NOWRITE '.
```

```
WORKING-STORAGE SECTION.
77 SERVIS      PICTURE A(8) VALUE 'LMOPEN '.
77 DATAID     PICTURE A(8).
77 OPTSHUN     PICTURE A(8) VALUE 'INPUT '.
:
```

## Invoking the ISPF services

```
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 these:

```
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 [z/OS 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 example shown in “[Assembler example](#)” on page 10 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'SAMPLE &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,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
CNOPI  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 on page 11](#).

*Table 1. Service Return Codes*

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

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

## 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 this example:

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

### PL/I

PL/I programs can examine the return code by using the PLIRETV built-in function. These 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. This 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, VIIF, 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, VIIF, and BRIF sections in Chapter 2, “Description of the ISPF services,” on page 21.

## A summary of the ISPF services

---

See:

- [“Display services” on page 14](#)
- [“File tailoring services” on page 14](#)
- [“Library access services” on page 14](#)
- [“PDF component services” on page 16](#)
- [“Table services” on page 16](#)
- [“Variable services” on page 17](#)
- [“Miscellaneous services” on page 18](#)

### Display services

#### **ADDDPOP**

Specifies that the panel displays listed 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.

#### **REMPPOP**

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

#### **DIRLIST**

Creates a list of files in a z/OS UNIX directory.

#### **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 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 DSNNAME LEVEL and VOLUME combination.

**LMDINIT**

Associates a DSNNAME LEVEL and VOLUME combination with a dslist ID. Thereafter, this dslist ID is used to identify the DSNNAME 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 listed in [“Library access services” on page 14](#).

### **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 partitioned data sets, sequential data sets, z/OS UNIX files, and data sets that can be allocated by using the LMINIT service.

### **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 modify partitioned data sets, sequential data sets, z/OS UNIX files, and data sets 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.

### **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 these 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 or profile pool.

### **VGET**

Retrieves variables from the shared pool or profile pool or retrieves the value of a system symbolic variable.

### **VPUT**

Updates variables in the shared pool or profile pool.

### **VSYM**

Updates the value of dialog variables found in the function pool by resolving the values of any system symbols.

## **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.

### **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.



## Chapter 2. Description of the ISPF services

The services are listed in alphabetical order.

Each service description consists of:

### 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.

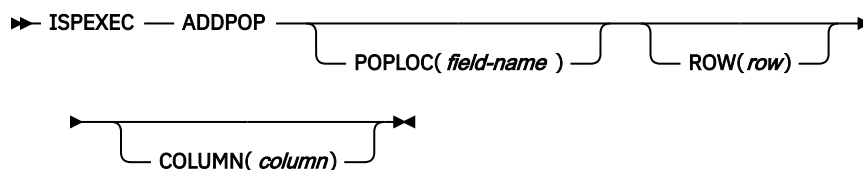
## ADDPOP—start pop-up window mode

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.

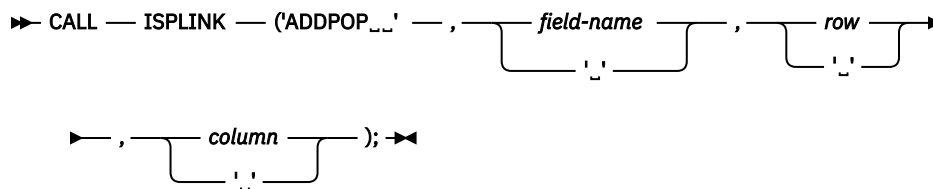
### Command invocation format



### Call invocation format

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

or



## Parameters

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

If you omit the *field-name* parameter when using the ADDPOP service, the Dialog Manager offsets 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

These 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

This 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 this panel:

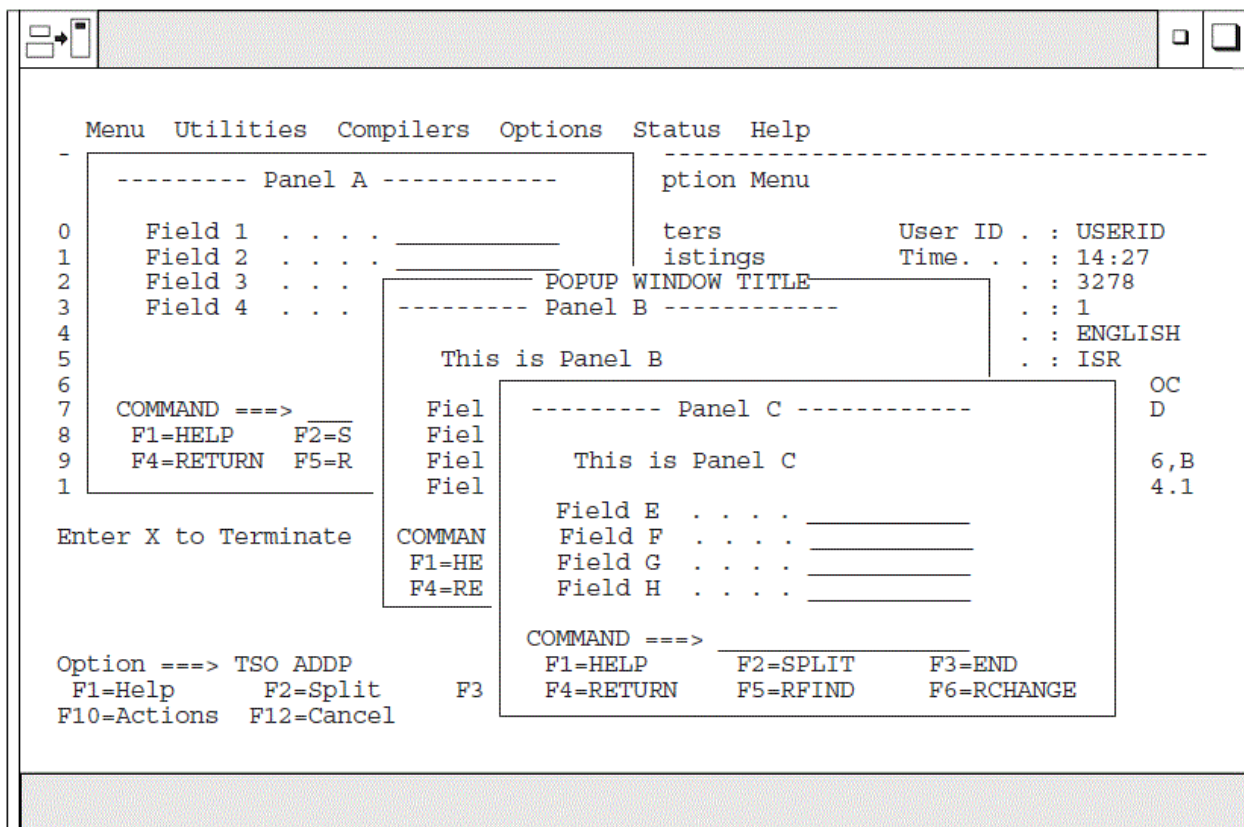


Figure 2. Multiple Pop-up Windows

## 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.

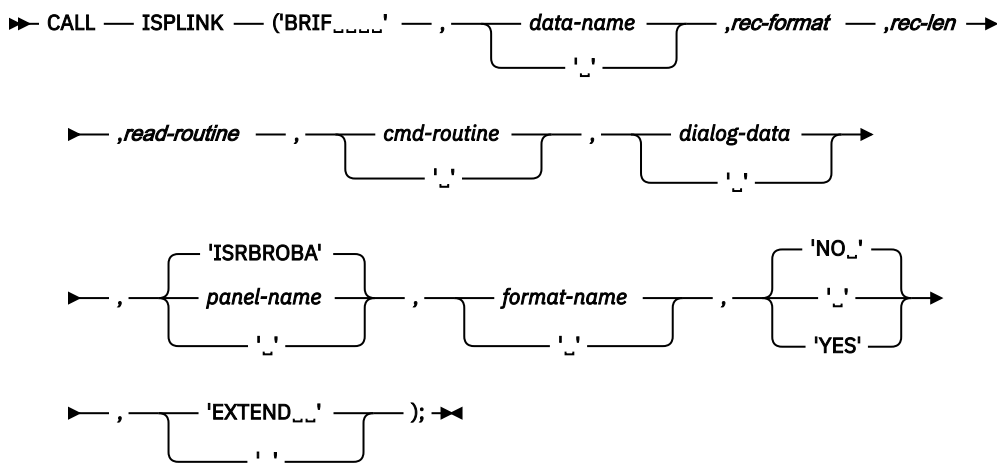
All addresses must be 31-bit addresses, and the routines must have an addressing mode (AMODE) of 31.

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



## 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. It is recommended that the high-order bit of this value be set ON. 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. It is recommended that the high-order bit of this value be set ON. 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 (ISRBROBA). Refer to [z/OS 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 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).

**EXTEND**

Indicates that the read routine has been changed to accept record numbers that exceed 99999999.

## 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.

A BRIF read routine must have an assembler interface to be used in a call to BRIF.

The dialog-supplied read 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:**

1. The dialog-supplied routines and the read and command exits can be written in languages that use the Language Environment runtime environment, provided the runtime environment has the Language Environment TRAP(ON) option set. However, a mixture of Language Environment-conforming main dialog code and service routine code is not supported. Dialogs and service routines must either all be Language Environment-conforming or all be Language Environment-nonconforming.
2. Language Environment applications that use the ISPF EDIF, VIIF, or BRIF service must use the Language Environment OS\_UPSTACK option. For ISPF to invoke the user routines with a valid LE dynamic save area, the Language Environment application must issue a CONTROL LE ON service request before each EDIF, VIIF, or BRIF service request and a CONTROL LE OFF service request after each EDIF, VIIF, or BRIF service request.

## Read routine

The read routine is invoked with these 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. When the EXTEND parameter is used relative record 2147483646 is requested. 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



## Call invocation format

```

➤ CALL — ISPLINK — ('BROWSE_<_>' — , <dsname> , <serial>
                                     ' ' ' '
➤ , <pswd-value> , <panel-name> , <data-id>
      ' ' ' '
➤ , <member-name> , <format-name> , { 'NO_'
      ' ' ' '                        ' '
                                     'YES' } , ➤
➤ <file-var> , <rec-len> <generation> ); ➤
      ' ' ' '

```

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 topic on naming ISPF libraries and data sets in the [z/OS ISPF User's Guide Vol 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®).

### panel-name

The name of a customized browse panel that you create, to be used when displaying the data. See [z/OS 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 (MIXED)**

For the MIXED parameter, if YES is specified, the BROWSE service treats the data as mixed-mode DBCS data. If NO 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).

### **file-var**

The name of an ISPF variable containing the path name for a z/OS UNIX regular file or directory. An absolute path name or a path name relative to the current working directory can be specified. Absolute path names begin with '/'. Relative path names begin with '..'. If the path name is for a directory, a directory selection list is displayed.

### **rec-len**

A numeric value specifying the record length to be used when browsing a z/OS UNIX file. This parameter causes newline characters in the data to be ignored as record delimiters.

### **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.

### **generation**

A fullword fixed integer containing the relative or absolute generation of the member to be browsed. If the value is negative, it is a relative generation. If the value is positive, it is an absolute generation that the caller has determined to be valid. The value 0 (zero) indicates the current generation and is equivalent to not specifying the parameter. This parameter is valid only when the specified member is in a PDSE version 2 data set that is configured for member generations.

### **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**

These return codes are possible:

**0**

Normal completion.

**12**

Zero-length data; empty sequential data set or z/OS UNIX file, or zero-length member of a partitioned data set.

**14**

Member or generation (if specified) not found.

**16**

Either:

- 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

The first examples invoke 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. The second example invokes the BROWSE service for z/OS UNIX file /u/user1/filea.

### Command invocation

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

or

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

or

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

```
FILEVAR = '/u/user1/filea'  
ISPEXEC BROWSE FILE(FILEVAR)
```

### Call invocation

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

or

Set the program variable BUFFER to contain:

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

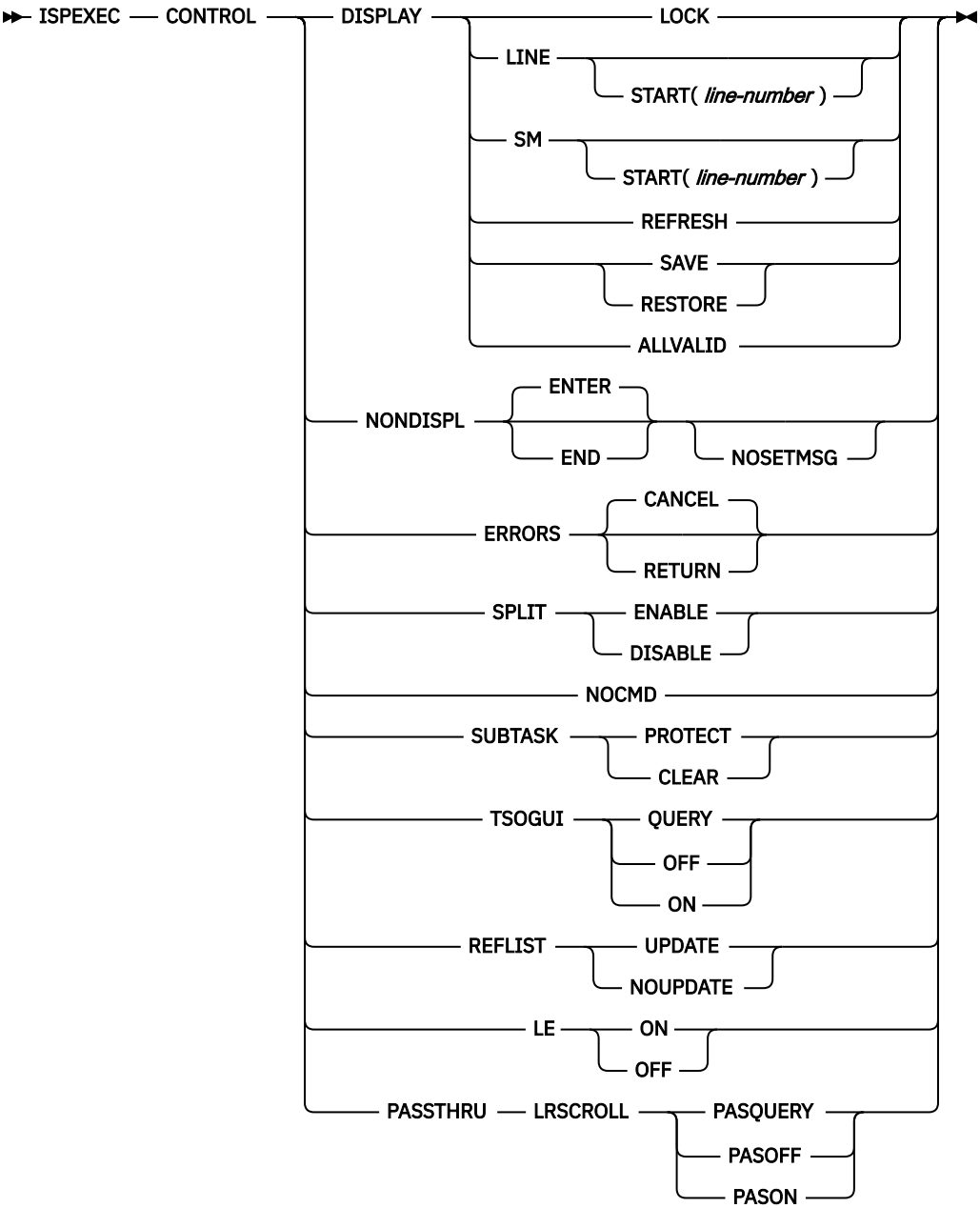
Set the program variable BUFLen to the length of the variable BUFFER. Enter the command:

```
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



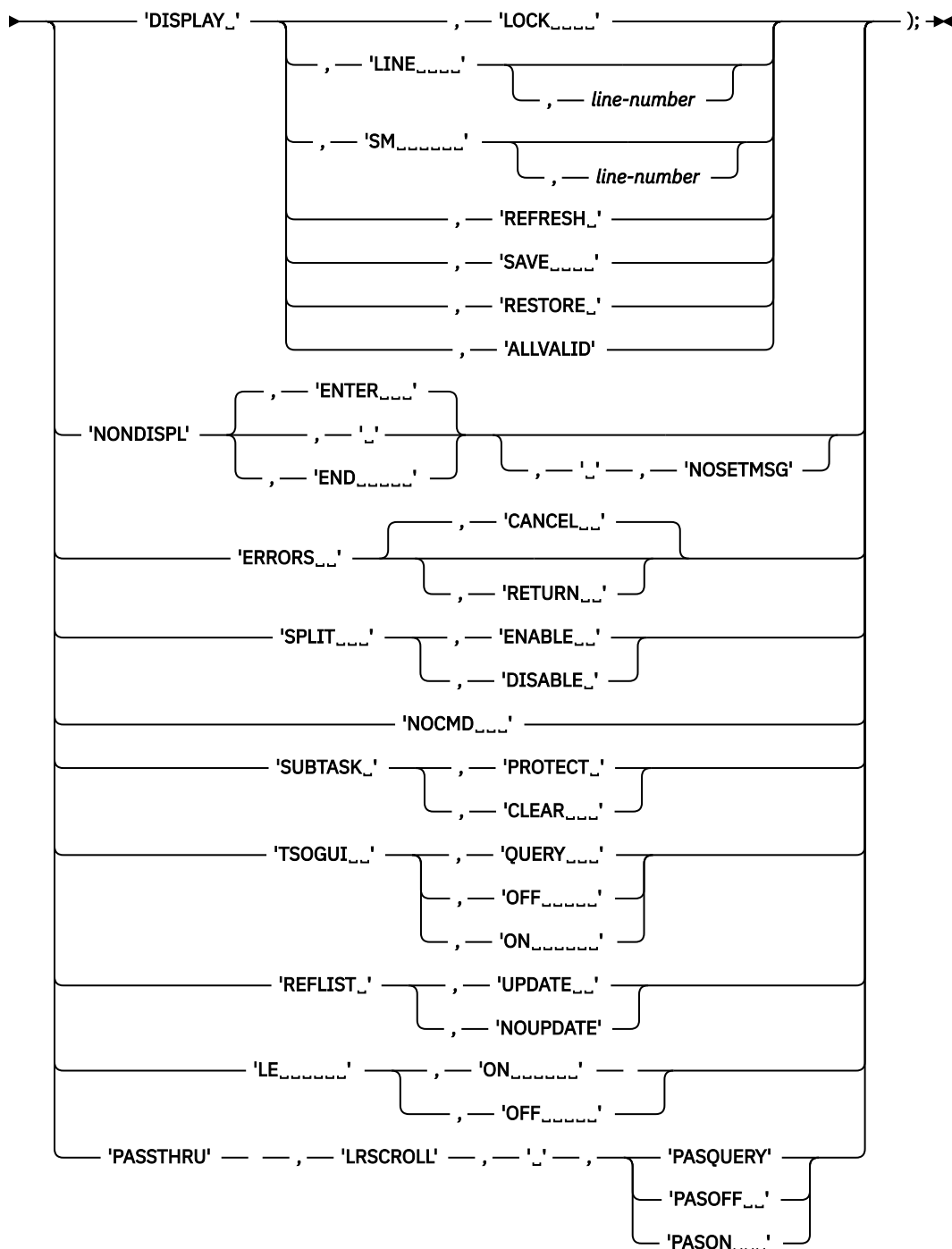
Call invocation format

➤ CALL — ISPEXEC — (buf-len, — buffer); ➤

or



➡ CALL — ISPLINK — ('CONTROL\_' — , — ➡



## ADDPop/REMPop service in 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, RESTORE, SM, and ALLVALID. LINE mode 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 output from 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. 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 being overlaid by line-mode messages. To ensure that the screen is fully rewritten you must follow the CONTROL DISPLAY LOCK request with a CONTROL DISPLAY LINE request.

CONTROL DISPLAY LOCK can be used to display an "in process" message during a long-running operation.

### LINE

Specifies that terminal line-mode output is expected, for example 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.

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

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

The MODE parameter of the SELECT service can be used to override this use of the percent sign.

### 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, following a GRINIT service request but before a GRTERM service request has 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. To ensure a return to full-screen mode in ISPF, the dialog should issue CONTROL DISPLAY LINE.

### SAVE

Used in conjunction with DISPLAY, TBDISPL, BROWSE, EDIT, or VIEW 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, EDIT, or VIEW 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 ensure 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:

1. 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.
2. Using START to invoke multiple screens might cause a panel to display after 'CONTROL NONDISPL' due to how the commands are stacked in the command field. An additional command delimiter might be needed after the command that has a 'CONTROL NONDISPL'. For example, if the ABC command has a 'CONTROL NONDISPL' before the first DISPLAY, you can specify START ABC;;START DEF. The START ABC string starts the ABC application. The first semicolon causes the panel to display in NONDISPLAY mode. The second semicolon is a delimiter before the second START DEF string, which invokes the DEF application.

## **CONTROL**

### **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. The error mode that has been set is not propagated to any new function invoked by the SELECT 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 ZERRxxxx, as described in [“Return codes from services” on page 11](#), 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, this 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.

## TSOGUI

### QUERY

Gives the current source and destination of TSO input and output:

#### Return code = 0

TSOGUI QUERY always returns a return code of 0, indicating that all TSO input and output is directed to the 3270 session.

### OFF

TSOGUI OFF is ignored if specified on the CONTROL service.

### ON

TSOGUI ON is ignored if specified on the CONTROL service.

## REFLIST

### UPDATE

Enable ISPF allocations to add entries to the data set and library reference lists.

### NOUPDATE

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

### Note:

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 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 ensure 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

## CONTROL

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.

### LE

ISPF initialisation for Language Environment support.

### ON

CONTROL LE ON is required before each EDIF, VIIF, or BRIF call where the application has provided Language Environment-enabled command, read, or write routines.

### OFF

CONTROL LE OFF is required after each such call.

## PASSTHRU

### LRSCROLL

#### PASQUERY

Gives the current status of processing for the LEFT and RIGHT scroll commands:

#### Return code = 0

The LEFT and RIGHT scroll commands are not being passed to the dialog program.

#### Return code = 1

The LEFT and RIGHT scroll commands are being passed to the dialog program for processing.

#### PASOFF

Specifies that the LEFT and RIGHT scroll commands are not to be passed to the dialog program.

#### PASON

Specifies that the LEFT and RIGHT scroll commands are to be passed to the dialog program for processing.

### 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

These 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.

## Examples

Here are some examples of the CONTROL service:

**Example 1:**

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. Enter the command:

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

or alternately

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

**Example 2:**

Return control to the dialog if there is an error in browse. After the browse service completes, terminate the dialog if any subsequent services receive return code 12 or higher.

```
ISPEXEC "CONTROL ERRORS RETURN"  
ISPEXEC "BROWSE DATASET('dsn')"  
ISPEXEC "CONTROL ERRORS CANCEL"
```

**Example 3:**

Lock the screen while displaying a status message.

```
ISPEXEC "CONTROL DISPLAY LOCK"  
ISPEXEC "DISPLAY MSG(ISPC069C)"
```

**Example 4:**

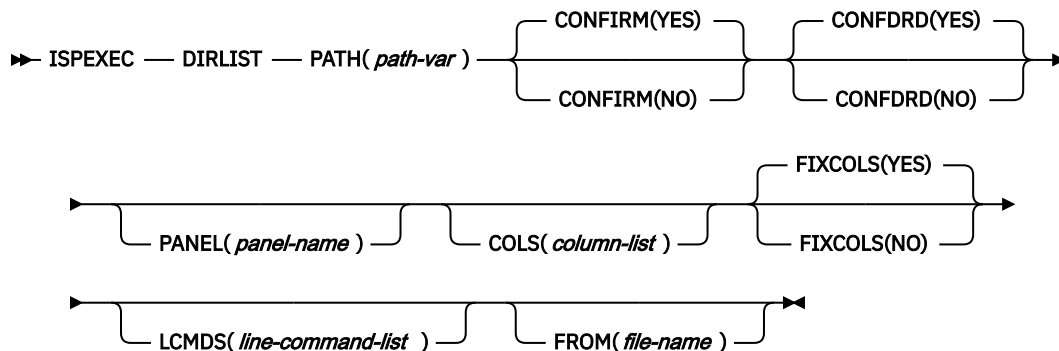
Process a panel definition without displaying the panel on the screen.

```
ISPEXEC "CONTROL NONDISPL"  
ISPEXEC "DISPLAY PANEL('global.panelname')"
```

## DIRLIST—directory list service

The DIRLIST service allows you to write your own z/OS UNIX directory list dialog. This service is similar to ISPF option 3.17, the z/OS UNIX Directory List Utility, which displays the list of files in a z/OS UNIX directory. The DIRLIST service allows the caller to control the format of the data displayed in the list and to process line commands entered against entries in the list.

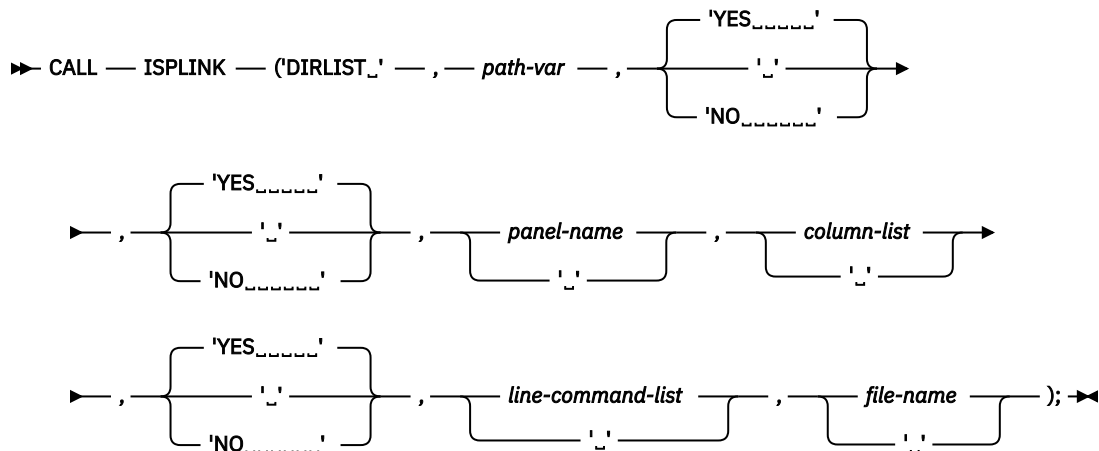
## Command invocation format



## Call invocation format

➡ CALL — ISPEXEC — (*buf-len*, — *buffer*); ➡

or



## Parameters

### path-var

The name of an ISPF variable containing the pathname for a z/OS UNIX directory. The pathname can contain glob characters to perform searching of the z/OS UNIX file system.

### YES|NO (CONFIRM)

Specifies whether the Confirm Delete panel appears when using the D (delete) line command from the displayed directory list to delete a file or empty directory. 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 file.

If NO is specified, ISPF does not display the Confirm Delete panel. The file is deleted without your having to take any additional actions.

### YES|NO (CONFDRD)

Specifies whether the Confirm Non-empty Directory Delete panel appears when using the D (delete) line command from the displayed directory list to delete a directory containing files.

YES is the default. If YES is specified, ISPF displays the Confirm Non-empty Directory Delete panel. This gives you the opportunity to change your mind and keep the directory.



If NO is specified, ISPF does not display the Confirm Non-empty Directory Delete panel. The directory and all the contained files and sub-directories are deleted without your having to take any additional actions.

### panel-name

The name of the panel to use for the display of the directory list. The default is the panel (ISRUUDL0) used for the directory list displayed using ISPF option 3.17, the z/OS UNIX Directory List Utility.

### column-list

Specifies the columns of data displayed on the z/OS UNIX directory list. If the parameter is omitted, the directory list is displayed using the column arrangement defined by the user using the **Directory List Column Arrangement** choice from the z/OS UNIX Directory List Utility Options pull-down menu.

To request that the ISPF-defined default column arrangement is used for the directory list display, specify an asterisk: COLS(\*).

Table 2 on page 41 shows the ISPF-defined column arrangement:

<i>Table 2. ISPF-defined column arrangement</i>	
Column	Width
Type	4
Permissions	10
Audit	6
Extended Attributes (Ext)	4
Format (Fmat)	4
Owner	8
Group	8
Links	6
Size	10
Modified Date/Time	19
Changed Date/Time	19
Accessed Date/Time	19
Created Date/Time	19

Alternatively, you can specify a paired list of columns and widths values. You specify the column values using the abbreviations shown in Table 3 on page 41. The table also shows the maximum widths that can be specified for each column.

<i>Table 3. Column abbreviations and widths</i>		
Column	Abbreviation	Maximum width
Type	TY	4
Permissions	PE <sup>1</sup>	10
Permissions - Octal	PO <sup>1</sup>	4
Audit	AU	6
Extended Attributes (Ext)	EX	4
Format (Fmat)	FM	4
Owner	OW	8

Table 3. Column abbreviations and widths (continued)		
Column	Abbreviation	Maximum width
Group	GR	8
Links	LI	14
Size	SZ	20
Modified Date/Time	MD	19
Changed Date/Time	CH	19
Accessed Date/Time	AC	19
Created Date/Time	CR	19
<b>Note:</b> 1. PE (Permissions) and PO (Permissions - Octal) are mutually exclusive.		

If a column is not specified with the COLS parameter, then that column is not displayed with the directory list. For example, the COLS parameter shown here causes the display of a z/OS UNIX directory list (see Figure 3 on page 42) showing only a Type column, Owner column, and a Changed Date/Time column 10 characters in width:

```
COLS(TY,4,OW,8,CH,10)
```

```

                                z/OS UNIX Directory List                Row 1 to 11 of 11
Command ==>                                Scroll ==> CSR

Pathname . : /SYSTEM

Command  Filename      Message      Type Owner    Changed
-----
.                Dir  IBMUSER  2007/01/31
..               Dir  IBMUSER  2007/05/02
bin              Syml IBMUSER  2007/01/31
dev              Dir  IBMUSER  2007/05/13
etc              Dir  IBMUSER  2007/06/12
lib              Syml IBMUSER  2007/01/31
opt              Syml IBMUSER  2007/01/31
samples          Syml IBMUSER  2007/01/31
tmp              Dir  IBMUSER  2007/07/16
usr              Syml IBMUSER  2007/01/31
var              Dir  IBMUSER  2007/06/12
***** Bottom of data *****

```

Figure 3. z/OS UNIX Directory List

### YES|NO (FIXCOLS)

Specifies whether you can change the column arrangement for the directory list display using the **Directory List Column Arrangement** choice from the z/OS UNIX Directory List Utility Options pull-down menu. This parameter is ignored if the column-list parameter is not specified on the call to DIRLIST.

If you specify YES, you cannot change the column arrangement specified by the column-list parameter.

If you specify NO, you can change the column arrangement specified by the column-list parameter using the **Directory List Column Arrangement** choice from the z/OS UNIX Directory List Utility Options pull-down menu.

### line-command-list

This parameter allows the calling application to process line commands entered on the z/OS UNIX directory list display. The caller can specify a directory list command processor and a list of line commands to be processed by the line command processor, rather than the ISPF directory list utility.

The first entry in the line-command-list is the name of the line command processor. This can be the name of a REXX exec or a program. The line command processor is invoked using the SELECT CMD service when one of the line commands specified in the line-command-list is entered on the directory list display. The line commands are specified after the name of the line command processor. Line commands can be from 1 to 8 characters in length. These line commands can include line commands normally processed by ISPF, allowing the application to override the processing of a line command.

The example of the LCMDS parameter shown here causes line commands DP, LCMD1, and W to be processed by line command processor LCMDDPROC. Also the B line command, normally processed by the ISPF directory list utility to invoke the ISPF browse function, is instead handled by LCMDDPROC.

```
LCMDS (LCMDDPROC , DP , B , LCMD1 , W)
```

The variables in the shared pool shown in [Table 4 on page 43](#) are used to pass data between ISPF and the line command processor:

<i>Table 4. Variables used to pass data between ISPF and the line command processor</i>		
<b>Variable name</b>	<b>Description</b>	<b>Length</b>
ZUDLCMD	Line command	8
ZUDPATH	Pathname of file the line command was entered against	1023
ZUDFTYPE	File type	4
ZUDFPERM	File permissions	10
ZUDFPRMO	File permissions - octal	4
ZUDFOWN	Owner	8
ZUDFAUDT	Audit settings	6
ZUDFEXTA	Extended attributes	4
ZUDFFORM	File format	4
ZUDFGRP	Owner group	8
ZUDFLNKS	Links	14
ZUDFSIZE	File size	20
ZUDFMDTM	Modified date/time	19
ZUDFCDTM	Changed date/time	19
ZUDFADTM	Accessed date/time	19
ZUDCRDTM	Created date/time	19
ZUDMESSG	Allows the line command processor to set the message displayed against the file after the line command is processed	16

### Line command processor parameters

Any parameters entered with a line command are passed to the line command processor.

For a REXX line command processor, each parameter specified with the line command is passed as an argument string to the REXX program. For example, if the line command

```
LL a1 B22 c333
```

is entered and processed by this REXX line command processor

```
/* REXX */
Arg p1 p2 p3 .
:
```

the REXX variable would have these values:

```
p1 = 'A1' p2 = 'B22' p3 = 'C333'
```

For a line command processor that is a program, on entry register 1 contains the address of a Command Processor Parameter List (CPPL). The format of the CPPL is described in the *TSO/E Programming Guide*. The CPPL contains the address of the command buffer. The text area of the command buffer contains the name of the line command processor followed by any parameters specified with the line command. For example, if the line command

```
LL a1 B22 c333
```

is entered, the text area of the command buffer for the line command processor LCMDPGM would contain:

```
LCMDPGM a1 B22 c333
```

### Line command processor return codes

**0**

Line command completed successfully.

**1**

Requests that processing of this line command be handled by the ISPF directory list utility or, if not a recognized directory list line command, passed to TSO.

**>=8**

The line command failed. ISPF issues message ISRU812.

### file-name

The display starting point within the directory list. When this parameter is used, the directory list is positioned to start at the first file matching or after the specified filename.

### 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

These return codes are possible:

**0**

Normal completion.

**8**

Error building the directory list. The error condition is described in the ISPF system dialog variables.

**12**

A keyword value is incorrect.

**20**

A severe error occurred while processing the directory list.

## Example

This example shows an invocation of DIRLIST which displays the directory list for /SYSTEM/etc. The list shows columns for Permissions, File Type, and Modified Date. The line command processor LCPROC is invoked for line commands LL, B, and UPD.

## Command invocation

```
dir = '/SYSTEM/etc'
ISPEXEC DIRLIST PATH(dir) COLS(PE,10,TY,4,MO,10) LCMSD(LCPROC,LL,B,UPD)
```

## Call invocation

```
dir = '/SYSTEM/etc';
CALL ISPLINK('DIRLIST',
             'DIR',
             '(PE,10,TY,4,MO,10)',
             '(LCPROC,LL,B,UPD)');
```

# DISPLAY—display panels and messages

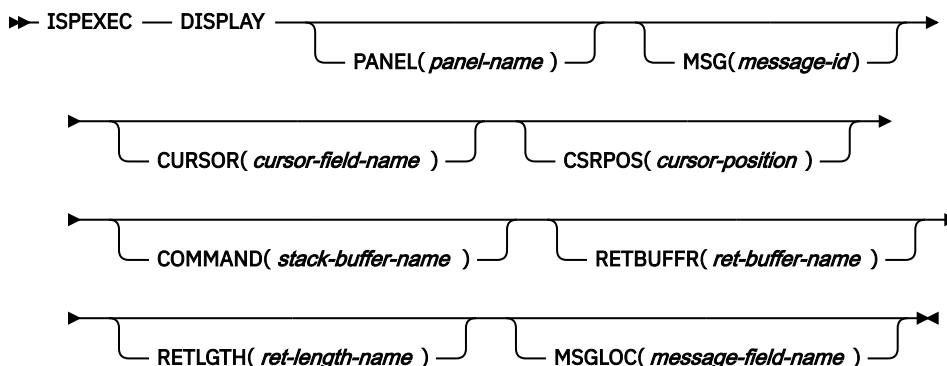
The DISPLAY service retrieves a panel definition, performs any pre-display processing specified on the panel definition, initializes variable panel fields from the corresponding dialog variables, and displays the panel on the screen. A message can optionally be displayed with the panel. If the optional 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.

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.

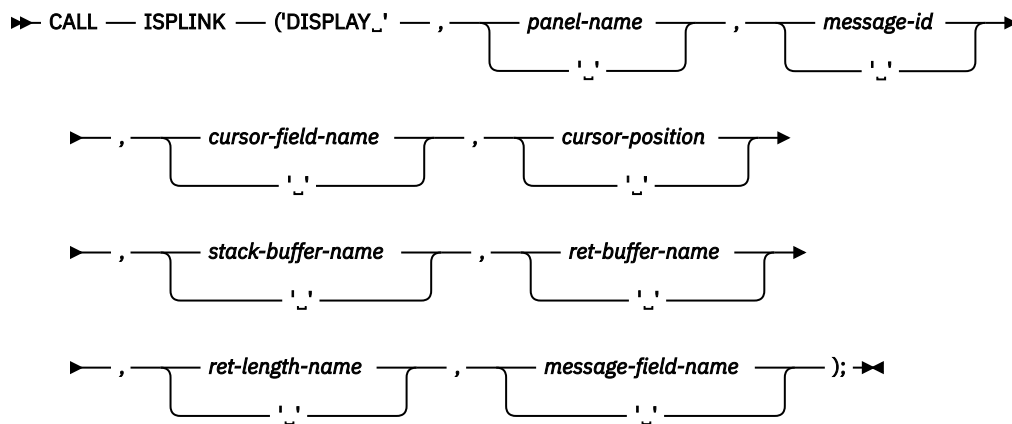
## Command invocation format



## Call invocation format

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

or



## 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.

**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 further on.

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.

## DISPLAY

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.

### Note:

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 canceled.
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

These 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.

## Examples

See:

- [“Example 1: Display variables and message, set cursor position” on page 49](#)
- [“Example 2: Unknown command handled by DISPLAY” on page 49](#)



- [“Example 3: Unknown command handled by dialog” on page 49](#)
- [“Example 4: Command stack contains an invalid parameter” on page 50](#)
- [“Example 5: Display message in a pop-up window” on page 50](#)

## Example 1: Display variables and message, set cursor position

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. Enter the command:

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

or alternately

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

## Example 2: Unknown command handled by DISPLAY

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) RETBUFR(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: Unknown command handled by dialog

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: Command stack contains an invalid parameter

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: Display message in a pop-up window

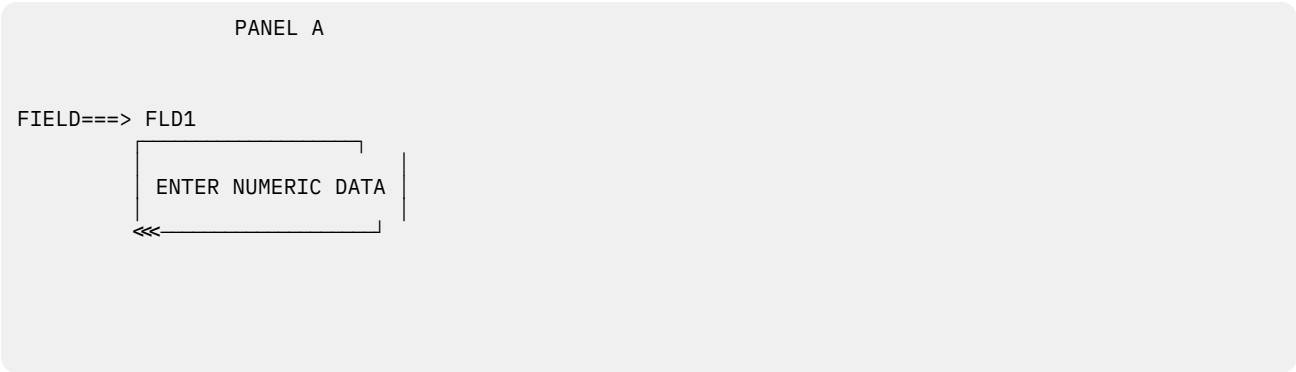
This DISPLAY request displays message TSTA110 in a message pop-up window that permits interaction with the underlying panel. The message pop-up window is positioned relative to the field FLD1.

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

Using this 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 format 1 or format 8 data set control block (DSCB). 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. This parameter must be a 46-byte length field for the call invocation format.
- 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 these dialog variables in the function pool:

Table 5. Dialog variables saved by the DSINFO service			
Variable	Information	Type	Length
ZDSVOL	First or only volume	Character	6

<i>Table 5. Dialog variables saved by the DSINFO service (continued)</i>			
<b>Variable</b>	<b>Information</b>	<b>Type</b>	<b>Length</b>
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
ZDSTOTAX	Allocated space units (long format)	Character	18
ZDSTOTUX	Used space units (long format)	Character	18
ZSDSNT	Data set name type	Character	8
ZDSSEQ	Compressed (YES/NO)	Character	4
ZDSCDATE	Creation date, shown in your national format with a 4-character year. Use system variable ZDATEF to determine the national format.	Character	10
ZDSXDATE	Expiration date, shown in your national format with a 4-character year. Use system variable ZDATEF to determine the national format.	Character	10
ZDSRDATE	Referenced date, shown in your national format with a 4-character year. Use system variable ZDATEF to determine the 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 (valid for PDS only)	Character	6
ZDSDIRU	Used directory blocks (valid for PDS only)	Character	8
ZDSDIR	Maximum directory blocks (valid for PDSE only)	Character	8
ZDS#MEM	Number of members (valid for PDS and PDSE only)	Character	13
ZDSPAGU	Pages used (valid for PDSE only)	Character	13
ZDSPERU	Percent used (valid for PDSE only)	Character	13
ZDSMC	Management class	Character	8
ZDSSC	Storage class	Character	8
ZSDSC	Data class	Character	8

<i>Table 5. Dialog variables saved by the DSINFO service (continued)</i>			
<b>Variable</b>	<b>Information</b>	<b>Type</b>	<b>Length</b>
ZDSAPF	APF status of the data set (YES/NO/ERR)  ERR indicates that an internal error occurred on authorized assembler service CSVAPF.	Character	4
ZDSLNK	LNKLST status of the data set (YES/NO/ERR)  ERR indicates that an internal error occurred on authorized assembler service CSVDYNL.	Character	4
ZDSCB1	Unformatted format 1 or format 8 data set control block	Character	96
ZDSVTAB	Volume table (contains all of the volume names for a multivolume set)	Character	354
ZDSOVF	Whether variables ZDSTOTAX and ZDSTOTUX should be used to obtain the 'allocated space units' and 'used space units' values (YES or NO). The value is YES when the 'allocated space units' value exceeds the size of variable ZDSTOTA or the 'used space units' value exceeds the size of variable ZDSTOTU.	Character	3
ZDSEATR	Indicates whether the data set can support extended attributes, as specified using the EATTR parameter on the allocation request. Extended attributes appear in the format 8 and format 9 DSCBs.  OPT: The data set can have extended attributes NO: The data set cannot have extended attributes blanks: The EATTR parameter was not specified	Character	4
ZDSENCR	Data set encryption status of the data set (YES/NO). YES indicates that the data set is encrypted using data set encryption. NO indicates that data set encryption is not supported for the data set or the data set is not encrypted using data set encryption.	Character	3
ZDSCJOBN	Create jobname (for a data set with a format 8 data set control block). If no value exists for this variable, ISPF sets the value to blanks.	Character	8
ZDSCSTPN	Create stepname (for a data set with a format 8 data set control block). If no value exists for this variable, ISPF sets the value to blanks.	Character	8
ZDSDSNV	Data set version (valid for PDSE only)	Character	1
ZDSNGEN	Maximum number of generations (valid for PDSE version 2 only)	Character	10

**Note:** ISPF cannot calculate reliable space utilization values for BDAM data sets. Therefore, the DSINFO service returns question marks (?) in variables that contain space utilization data when reporting on BDAM data sets.

## Return codes

These return codes are possible:

**0**

Normal completion.

**8**

User requested information unavailable. Dialog error variables (ZERRLM, and so on) contain further information.

**12**

One of these:

- Internal Service Failure
- Error when using the OBTAIN macro to read the DSCB
- Error obtaining directory information

**20**

Severe error.

## Example

This 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. Enter the command:

```
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 ISPF-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_<u>      </u>' — , <u>data-name</u> , <u>profile-name</u> — , <u>rec-format</u> →
                                   <u>'<u></u></u>'</u>

▶ — , <u>rec-len</u> — , <u>read-routine</u> — , <u>write-routine</u> — , <u>cmd-routine</u> →
                                   <u>'<u></u></u>'</u>

▶ — , <u>dialog-data</u> , <u>edit-len</u> , <u>panel-name</u> →
      <u>'<u></u></u>'</u> <u>'<u></u></u>'</u> <u>'<u></u></u>'</u>

▶ — , <u>macro-name</u> , <u>format-name</u> , { <u>'NO_<u></u></u>'</u>
      <u>'<u></u></u>'</u> }

▶ — , { <u>'NO_<u></u></u>'</u>
      <u>'<u></u></u>'</u> } , <u>parm-var</u> , <u>tabname</u> ); ▶▶
      <u>'<u></u></u>'</u> <u>'<u></u></u>'</u> <u>'<u></u></u>'</u>

```

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 this format for EDIF, specifying YES for the recovery-request parameter:

```

▶▶ CALL — ISPLINK — ('EDIF_<u>      </u>' — , <u>data-name</u> , '<u></u></u>' — , <u>rec-format</u> →
                                   <u>'<u></u></u>'</u> <u>'<u></u></u>'</u>

▶ — , <u>rec-len</u> , <u>read-routine</u> — , <u>write-routine</u> — , <u>cmd-routine</u> , →
      <u>'<u></u></u>'</u> <u>'<u></u></u>'</u> <u>'<u></u></u>'</u> <u>'<u></u></u>'</u> <u>'<u></u></u>'</u>

▶ — , <u>dialog-data</u> , '<u></u></u>' — , '<u></u></u>' — , '<u></u></u>' — , '<u></u></u>' — , '<u></u></u>' →
      <u>'<u></u></u>'</u> <u>'<u></u></u>'</u> <u>'<u></u></u>'</u> <u>'<u></u></u>'</u> <u>'<u></u></u>'</u>

▶ — , '<u>YES_<u></u></u>' — , '<u></u></u>' — , <u>tabname</u> ); ▶▶
      <u>'<u></u></u>'</u> <u>'<u></u></u>'</u> <u>'<u></u></u>'</u>

```

## 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). It is recommended that the high-order bit of this value be set ON. See [“Read routine” on page 58](#) for more information about this parameter.

### **write-routine**

A fullword address indicating the entry point of a dialog-supplied write routine (required). It is recommended that the high-order bit of this value be set ON. See [“Write routine” on page 58](#) 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. It is recommended that the high-order bit of this value be set ON. See [“Command routine” on page 59](#) 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. See [z/OS 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. See [z/OS 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 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.

**tablename**

The name of a user line command table to be used for the edit session. The value must be 8 characters, blank padded.

## 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:**

1. The read, write, and command exits can be written in languages that use the Language Environment runtime environment, provided the runtime environment has the Language Environment TRAP(ON) option set. However, a mixture of Language Environment-conforming main dialog code and service routine code is not supported. Dialogs and service routines must either all be Language Environment-conforming or all be Language Environment-nonconforming.
2. Language Environment applications that use the ISPF EDIF, VIIF, or BRIF service must use the Language Environment OS\_UPSTACK option. For ISPF to invoke the user routines with a valid LE dynamic save area, the Language Environment application must issue a `CONTROL LE ON` service request before each EDIF, VIIF, or BRIF service request and a `CONTROL LE OFF` service request after each EDIF, VIIF, or BRIF service request.

## 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 these 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 these 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 these 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);
```

## 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.



## Return codes

These 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 ISPF 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. [z/OS ISPF Edit and Edit Macros](#) contains a complete description of the editor.

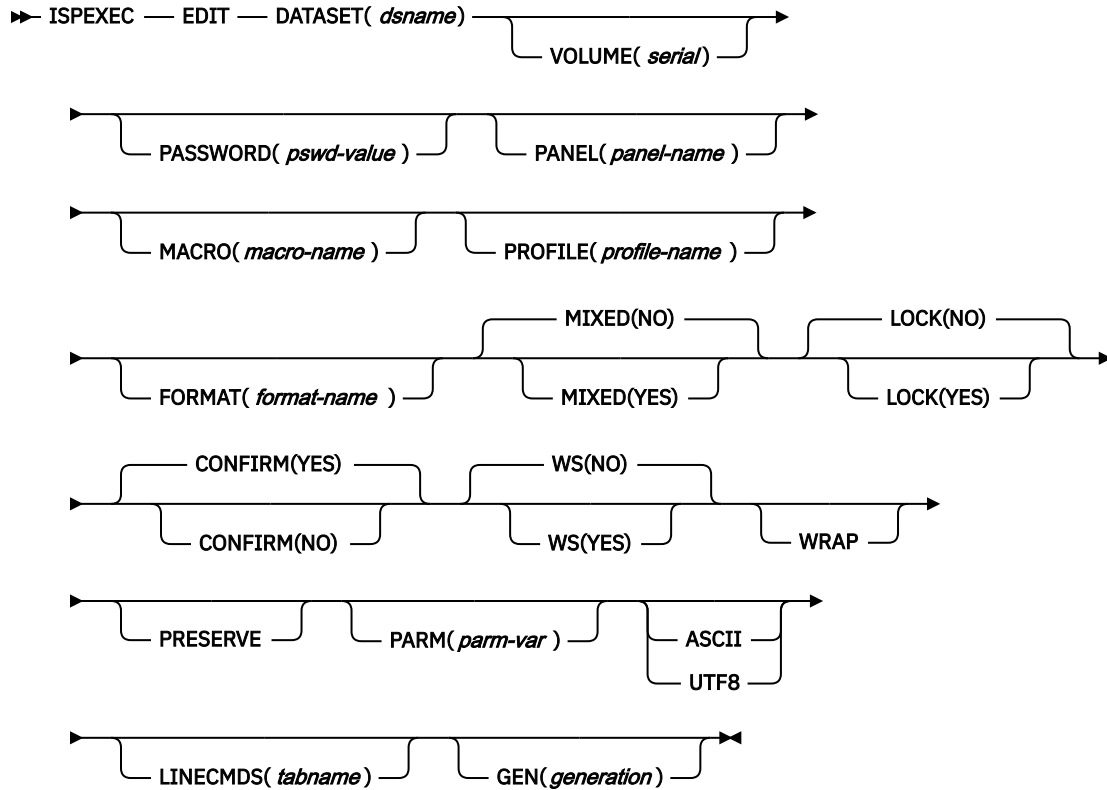
When EDIT is operating in recovery mode, an audit trail of your interactions is automatically recorded in an ISPF-controlled data set. Following a system failure, you can use the audit trail to recover the data you were editing.

### Note:

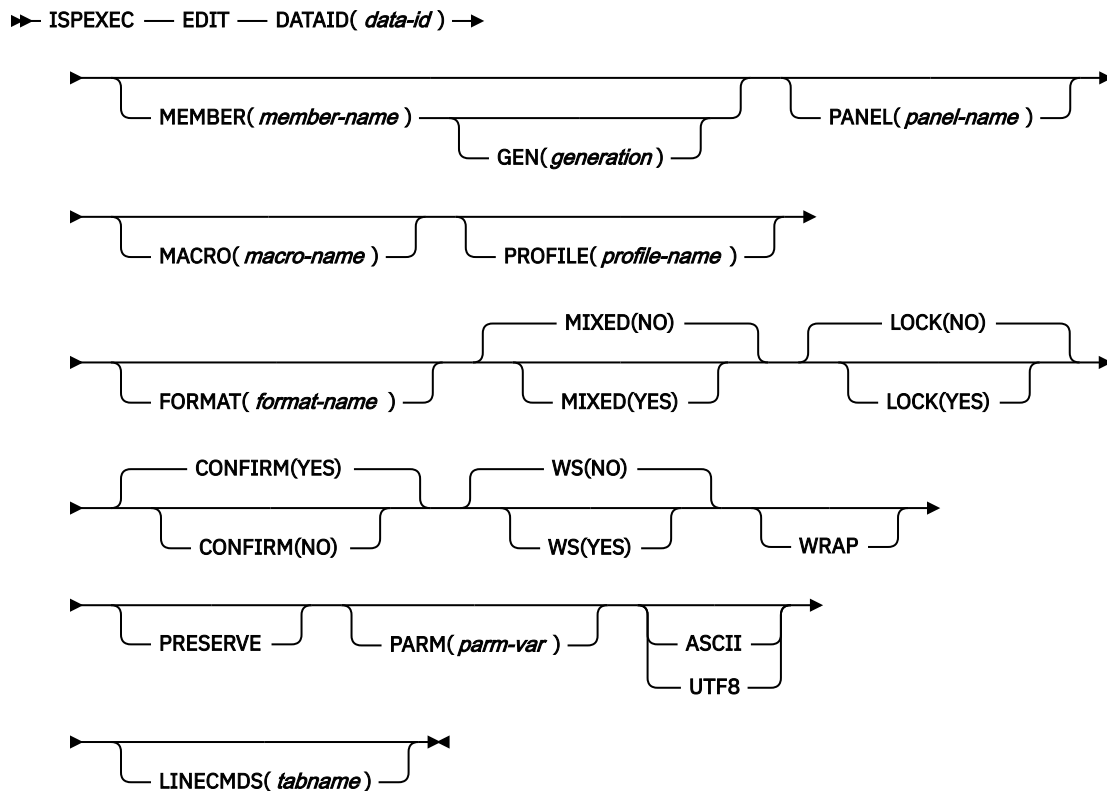
1. Dialogs that invoke the EDIT service may invoke the EDREC service first to start edit recovery, because the EDIT service does not do edit recovery.
2. The EDIT service might alter the DISPLAY environment. Do not expect the DISPLAY environment that existed before invoking the EDIT service to remain 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.

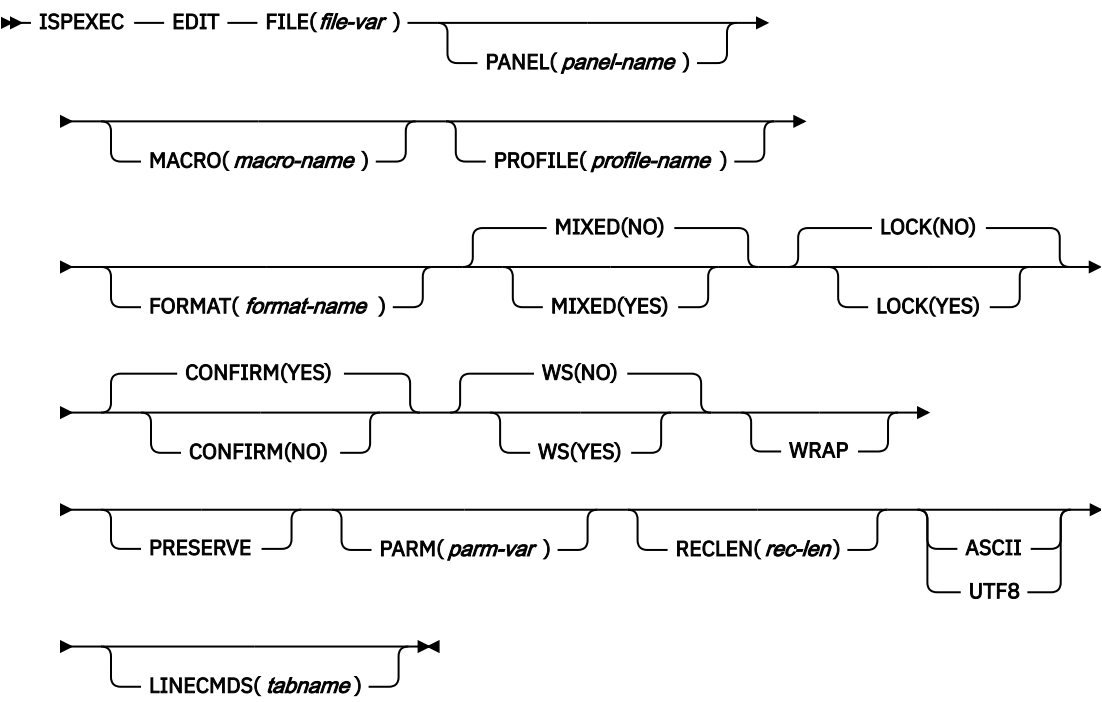
## Command invocation format



or

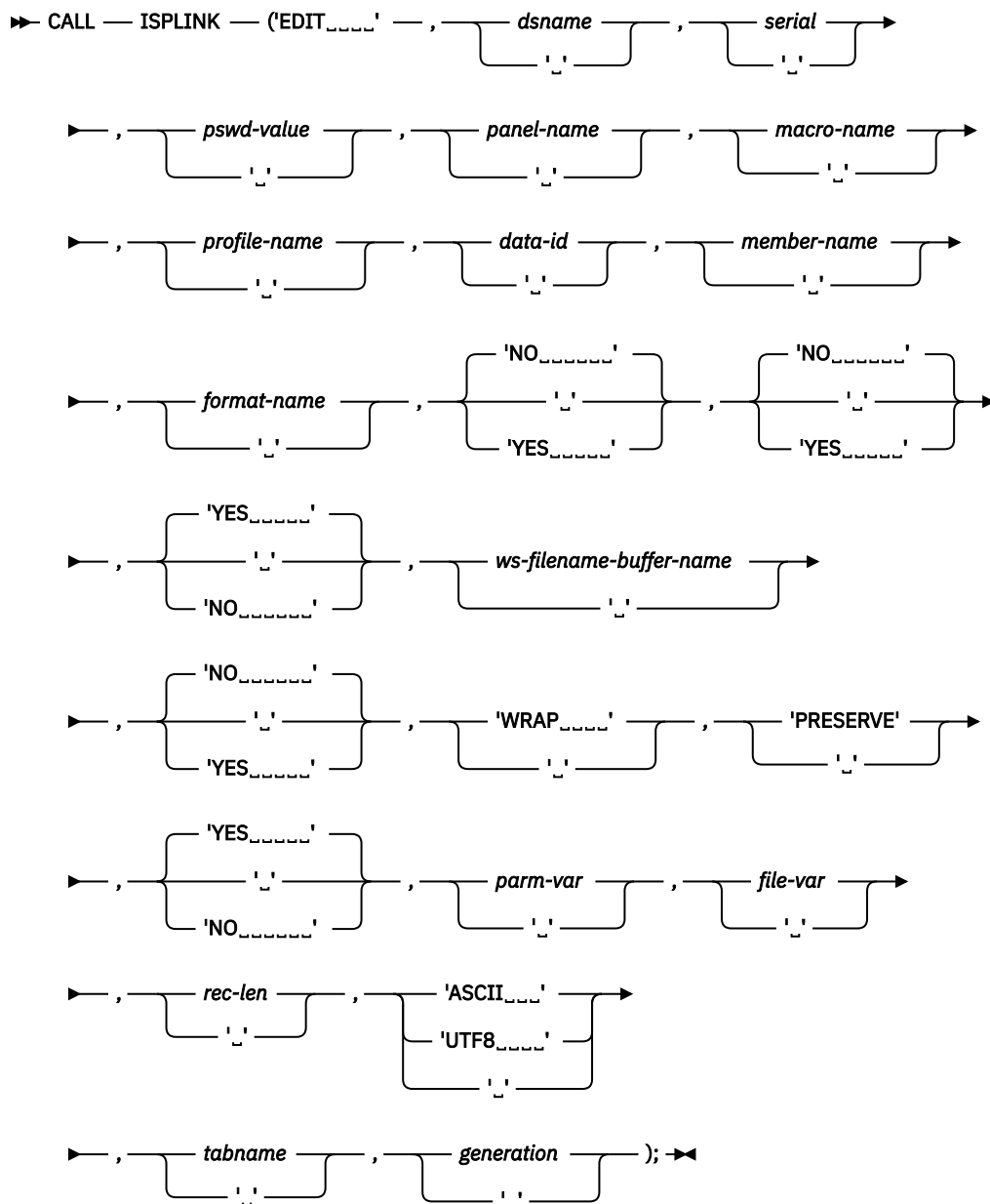


or





## Call invocation format



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 [z/OS ISPF User's Guide Vol 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-protected data sets.

#### **panel-name**

The name of a customized edit panel, created by you, to be used when displaying the data. See [z/OS 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. See [z/OS 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 [z/OS ISPF Edit and Edit Macros](#) for more information.

#### **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 [z/OS ISPF User's Guide Vol I](#) for more information about patterns and pattern matching.

#### **generation**

A fullword fixed integer containing the relative or absolute generation of the member to be edited. If the value is negative, it is a relative generation. If the value is positive, it is an absolute generation that the caller has determined to be valid. The value 0 (zero) indicates the current generation and is equivalent to not specifying the parameter. This parameter is valid only when the specified member is in a PDSE version 2 data set that is configured for member generations.

#### **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 (MIXED)**

For the MIXED parameter, if YES is specified, the EDIT service treats the data as mixed-mode DBCS data. If NO 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 (LOCK)**

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 (CONFIRM)**

For the CONFIRM parameter, if you specify YES 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, canceled, 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 ENTER. 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.

**ws-filename-buffer-name**

The *ws-filename-buffer-name* parameter is no longer accepted, but it is left in for compatibility purposes. If specified, the edit service will fail with return code 12.

**YES|NO (WS)**

The WS parameter is no longer used, but it is left in for compatibility purposes. If specified, the edit service will ignore it.

**WRAP**

The WRAP parameter is no longer used, but it is left in for compatibility purposes. If specified, the edit service will ignore it.

**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 [z/OS ISPF Edit and Edit Macros](#).

**YES|NO (CHGWARN)**

For the CHGWARN parameter, 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 warning is given. This parameter is ignored for EDIT.

**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.

**file-var**

The name of an ISPF variable containing the path name for a z/OS UNIX regular file or directory. An absolute path name or a path name relative to the current working directory can be specified. Absolute path names begin with '/'. Relative path names begin with '..'. If the path name is for a directory, a directory selection list is displayed.

**rec-len**

A numeric value specifying the record length to be used when editing a z/OS UNIX file. This parameter causes the records to be loaded into the editor as fixed length and saved back in the file as fixed length.

**ASCII|UTF8**

This parameter can be specified when invoking EDIT to edit data encoded in ASCII (or UTF-8) and the file is not tagged with a CCSID of 819 (or 1208).

When ASCII is specified or the file is tagged with CCSID 819, the editor renders the ASCII data readable by converting it to the CCSID of the terminal. Also, if set for a z/OS UNIX file, the editor breaks up the data into records using the ASCII linefeed character (X'0A') and the ASCII carriage return character (X'0D') as the record delimiter. For z/OS UNIX files, the linefeed and carriage return characters are removed from the data loaded into the editor but written back to the file when the data is saved.

When UTF8 is specified, or the file is tagged with CCSID 1208, the equivalent actions happen, except for UTF-8 instead of ASCII.

**tabname**

The name of a user line command table to be provided by the service caller.

**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**

These return codes are possible:

**0**

Normal completion; data was saved.

**4**

Normal completion; data was *not* saved for one of these 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.

**10**

Member or generation (if specified) not found.

**12**

YES was specified for the LOCK parameter or the ws-filename-buffer-name parameter was specified.

**14**

Member, sequential data set, or z/OS UNIX file in use.

**16**

Either:

- 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.

**Examples**

See:

- [“Example 1: Edit a PDS member” on page 69](#)
- [“Example 2: Edit a z/OS UNIX file” on page 69](#)

### Example 1: Edit a PDS member

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

### Command invocation

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

or

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

### Call invocation

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

or

Set the program variable BUFFER to contain:

```

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

```

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

CALL ISPEXEC (BUFFLN, BUFFER):

## Example 2: Edit a z/OS UNIX file

This example invokes the EDIT service for z/OS UNIX file /u/user1/filea.

### Command invocation

```
FILEVAR='/u/user1/filea'
ISPEXEC EDIT FILE(FILEVAR)
```

### Call invocation

```
FILEVAR='/u/user1/filea';
CALL ISPLINK('EDIT', 'FILEVAR');
```

## 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.

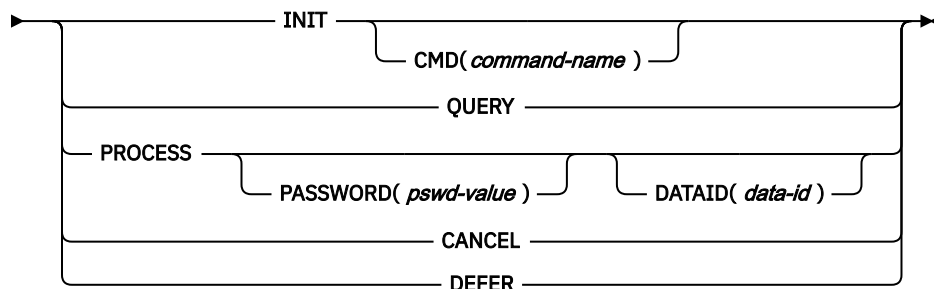
When you invoke the EDREC service, EDREC displays a special panel. Using this panel you can recover data, cancel recovery, defer recovery until a later time, or enter the END command to return to the next sequential command in your command invocation or to return to the next sequential instruction in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

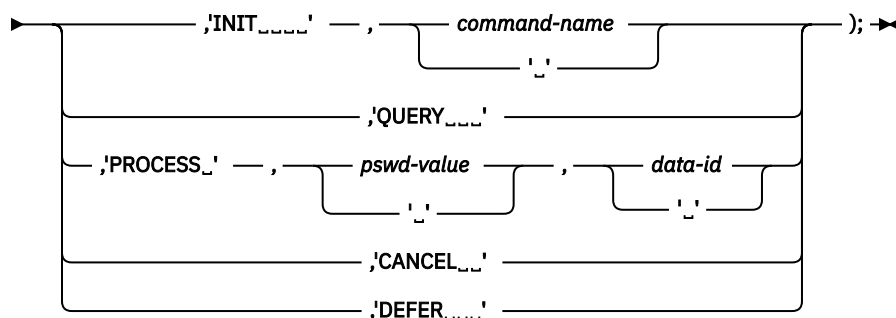
## Command invocation format

►► ISPEXEC — EDREC →



## Call invocation format

►► CALL — ISPLINK — ('EDREC\_...' →



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 xxxxEDRT, 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 ISPF-supplied CLIST named ISREDRTI. ISREDRTI 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 these EDREC functions: PROCESS, CANCEL, or DEFER.

The variables shown 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.

**ZEDBDN**

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 previous 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

These 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.

## Examples

Here are some examples of the EDREC service:

### Example 1:

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 command:

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

## Example 2:

This REXX exec invokes the LMINIT service in preparation for the EDIT service call. Then, it invokes the EDREC service to create an edit recovery table if one does not exist. The exec then uses the QUERY parameter of the EDREC service to see if edit recovery is pending. If it is, then it displays the edit recovery panel, ISREDM02, and process the response. If recovery is requested from ISREDM02, the EDREC service is called with the PROCESS parameter; otherwise, the EDREC service is called with the DEFER parameter. The EDREC QUERY is invoked from within a loop so that all pending edit recovery sessions can be processed. After edit recovery processing is complete, the EDIT service is called, followed by an LMFREE to free the data ID set by the LMINIT service.

```
/* REXX exec to use edit recovery prior to edit */
address ispxexec
'lmnit dataid(data1) dataset(private.source)'
if rc = 0 then
  do
    'edrec init'                                /* create recovery table */
    do until edrc/=4 | edcon = 0
      'edrec query'                            /* check for recovery ds */
      edrc = rc
      if edrc=4 then
        do
          z1=zedtdsn                            /* set up panel variable */
                                          /* and show recovery panel */
          'display panel(isredm02) cursor(zcmd)'
          if rc = 0 & substr(zedcmd,1,1)=' ' then
            do
              'edrec process'                    /* process recovery */
              edcon = 0                          /* and end loop */
            end
          else if rc = 0 & substr(zedcmd,1,1)='c' then
            'edrec cancel'
          else
            'edrec defer'
          end
        end
      end
    end
    'edit dataid('data1') member(sample)'
    'lmfree dataid('data1')'
  end
```

## 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.

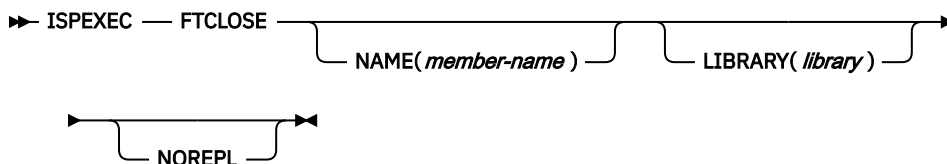
A member-name parameter should be specified if the output is a library. The file tailoring output is given the specified member name. No error condition results if the member-name parameter is not specified and the output is not stored in the library.

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 ISPF 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 ISPF 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.

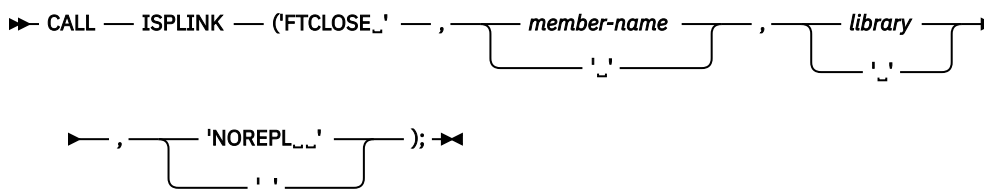
## Command invocation format



## Call invocation format

➤ CALL — ISPEXEC — (*buf-len*, — *buffer*); ➤

or



## Parameters

### member-name

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

### library

Specifies the name of a DD statement or lib-type on the LIBDEF service request that defines the output library in which the member-name exists. If specified, a generic (non-ISPF) ddname must be used. If this parameter is omitted, the default is ISPF.

### NOREPL

Specifies that FTCLOSE is not to overlay an existing member in the output library.

### 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

These return codes are possible:

**0**

Normal completion.

- 4** Member already exists in the output library and NOREPL was specified. The original member is unchanged.
- 8** File not open. FTOPEN was not used before FTCLOSE.
- 12** Output file in use. ENQ failed.
- 16** Skeleton library or output file not allocated.
- 20** Severe error.

## Example

End the file tailoring process and store the result of the processing in the file tailoring output library in member TELOUT.

```
ISPEXEC FTCLOSE NAME(TELOUT)
```

Set the program variable BUFFER to contain:

```
FTCLOSE NAME(TELOUT)
```

Set program variable BUFLen to the length of the variable BUFFER. Enter the command:

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

or alternately:

```
CALL ISPLINK ('FTCLOSE ', 'TELOUT ');
```

## FTERASE—erase file tailoring output

The FTERASE service erases a member of a file tailoring output library.

A severe error occurs if a specified library or the default, ISPFIL, is a sequential file.

### 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. ISPFIL 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

These 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. Enter the command:

```
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 — EXT — ➤
```

## Call invocation format

➤ CALL — ISPEXEC — (*buf-len* , — *buffer*); ➤

or

➤ CALL — ISPLINK — ('FTINCL\_...' — , — *skel-name* — , — 'NOFT\_...' —  
 '...' —  
 , — 'EXT\_...' — ); ➤  
 '...'

## 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.

### **EXT**

Enables support for built-in functions in all skeletons processed by the FTINCL call, unless the NOEXT parameter is specifically included on the )IM control statement when embedding a lower level skeleton.

### **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

These 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. Enter the command:

```
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 ISPFIL before invoking this service. ISPFIL 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.

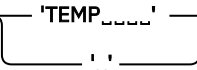
### Command invocation format

```
➤ ISPEXEC — FTOpen —  ➤
```

### Call invocation format

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

or

```
➤ CALL — ISPLINK — ('FTOPEN_...' — ,  ); ➤
```

### Parameters

#### TEMP

Specifies that the output of the file tailoring process should be placed in a temporary sequential file. Output is fixed-length 80-byte records. The file is automatically allocated by ISPF. Its name is available in system variable ZTEMPF.

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

ZTEMPF contains a fully qualified data set name. ZTEMPN contains the ddname. Generated JCL in this file can be substituted for background execution by using the TSO command:

```
SUBMIT '&ZTEMPF'
```

Before issuing the SUBMIT command, the VGET service should be invoked to initialize the variable ZTEMPF, and the FTCLOSE service must be invoked to ensure that all of the file tailoring output is included.

#### 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**

These 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. Enter the command:

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

or alternately:

```
CALL ISPLINK ('FTOPEN');
```

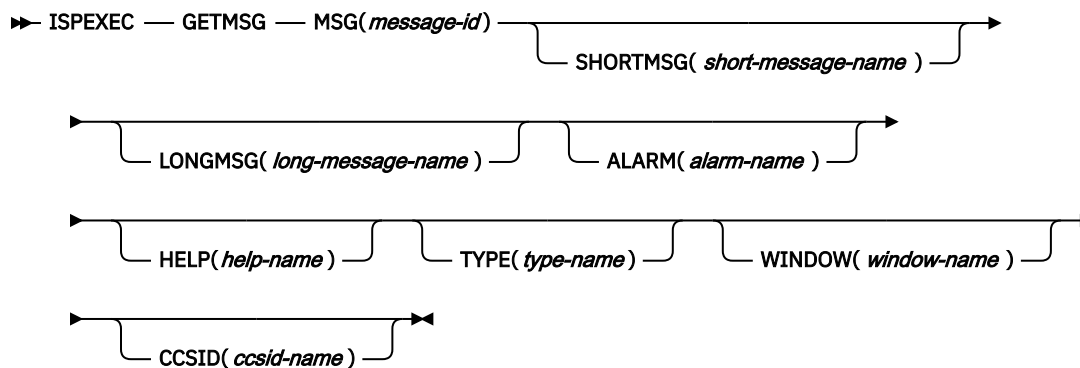
**GETMSG—get a message**

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.

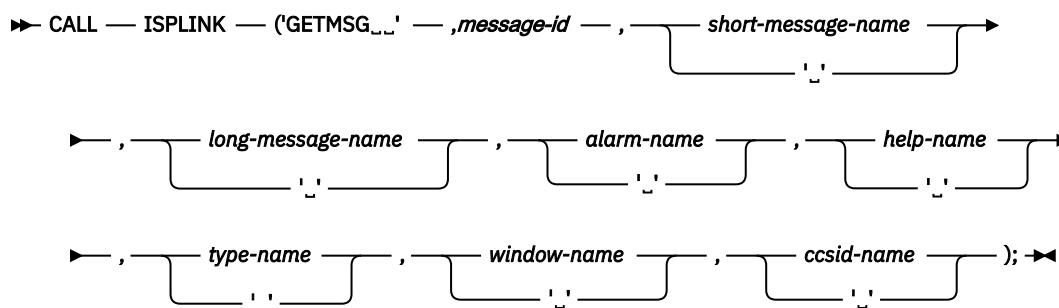
## Command invocation format



## Call invocation format

➤ CALL — ISPEXEC — (*buf-len*, — *buffer*); ➤

or



## 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**

These 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 BUFLN to the length of the variable BUFFER. Enter the command:

```
CALL ISPEXEC (BUFLN, 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 *GDDM Base Application Programming Guide* for information about the GDDM error record and *GDDM Base Application Programming Reference* for information about 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*
```

```

➡ 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

These 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.

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 these GDDM calls:

ASREAD	FSSHOR	ISFLD	MSPCRT	MSQMOD	PTNSEL	WSCRT
FSSHOW	ISQFLD	MSPQRY	MSQPOS	PTSCRT	WSDDEL	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, these 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.

### **Note:**

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

These return codes are possible:

**O**

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

These 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 these 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.

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 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.

**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 196](#).

The currently allocated ISPF libraries must still be defined before invoking ISPF and cannot be changed while in an ISPF session. 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.

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, these 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.

The DATASET (or EXCLDATA) and LIBRARY (or EXCLLIBR) keywords are mutually exclusive.

## 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 ISPLLIB) 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.

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.

Table 6 on page 86 defines the search sequence for all item types.

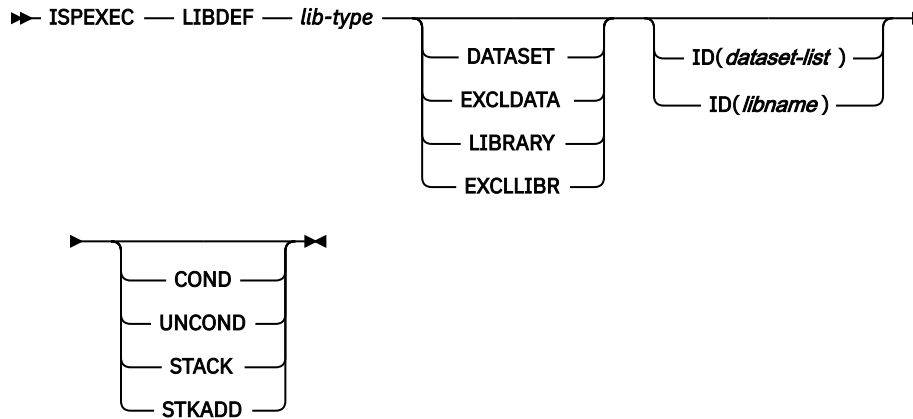
Table 6. Search Sequence for Libraries					
	No LIBDEF	LIBDEF with DATASET	LIBDEF with LIBRARY	LIBDEF with EXCLDATA	LIBDEF with EXCLLIBR
Panels	ISPPLIB	ISPPUSR LIBDEF ISPPLIB	LIBDEF ISPPLIB	Not valid	Not valid
Messages	ISPMLIB	ISPMUSR LIBDEF ISPMLIB	LIBDEF ISPMLIB	Not valid	Not valid
Table Input	ISPTLIB	ISPTUSR LIBDEF ISPTLIB	LIBDEF ISPTLIB	Not valid	Not valid
Skeleton	ISPSLIB	ISPSUSR LIBDEF ISPSLIB	LIBDEF ISPSLIB	Not valid	Not valid
Images	See <a href="#">note 3.</a>	See <a href="#">note 3.</a>	See <a href="#">note 3.</a>	See <a href="#">note 3.</a>	See <a href="#">note 3.</a>
Linklib (See note following this table)	Job Pack Area ISPLLIB STEPLIB Link Pack Area LINKLIB	Job Pack Area ISPLUSR LIBDEF ISPLLIB STEPLIB Link Pack Area LINKLIB	Job Pack Area LIBDEF ISPLLIB STEPLIB Link Pack Area LINKLIB	Job Pack Area LIBDEF Link Pack Area LINKLIB	Job Pack Area LIBDEF Link Pack Area LINKLIB
Table Output	ISPTABL	ISPTABU LIBDEF	LIBDEF	Not valid	Not valid
File Tailoring Output	ISPFIL	ISPFILU LIBDEF	LIBDEF	Not valid	Not valid
Table Services (Input) with LIBRARY Parameter	Allocated Library	(Unchanged)	LIBDEF	Not valid	Not valid
Table Services (Output) with LIBRARY Parameter	Allocated Library	LIBDEF	(Unchanged)	Not valid	Not valid
File Tailoring Services (Output) with LIBRARY Parameter	Allocated Library	LIBDEF	(Unchanged)	Not valid	Not valid

**Note:**

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. See *z/OS ISPF Planning and Customizing* for information about customizing ISPTCM for the correct search order.

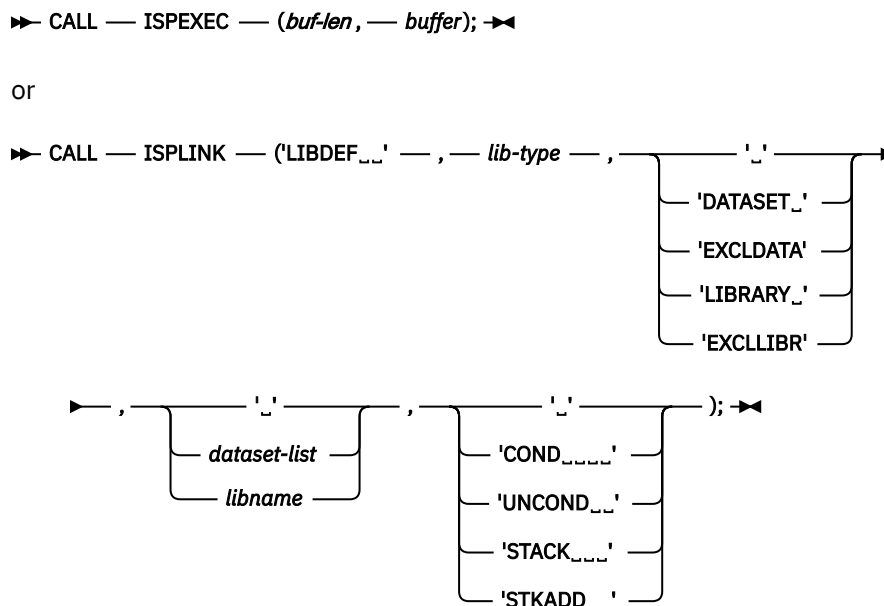
2. When using a SELECT with NEWAPPL, you must include PASSLIB to use the LIBDEFs you defined. For more details, see the description of the NEWAPPL parameter under [“SELECT—select a panel or function”](#) on page 201.
3. The image library with the associated ddname ISPILIB is no longer used by ISPF.

## Command invocation format



**Note:** If none of the processing options COND, UNCOND, STACK, or STKADD is specified, the processing option is set using the value in the ISPF configuration table keyword `DEFAULT_LIBDEF_PROCESSING_OPTION`. If this keyword is not set in the ISPF configuration table, the processing option is set to the default value UNCOND. Always specify a processing option for the LIBDEF service to ensure that changes to the `DEFAULT_LIBDEF_PROCESSING_OPTION` value in the configuration table do not cause unexpected changes to your dialog processing.

## Call invocation format



**Note:** If none of the processing options COND, UNCOND, STACK, or STKADD is specified, the processing option is set using the value in the ISPF configuration table keyword `DEFAULT_LIBDEF_PROCESSING_OPTION`. If this keyword is not set in the ISPF configuration table, the processing option is set to the default value UNCOND. Always specify a processing option for the LIBDEF service to ensure that changes to the `DEFAULT_LIBDEF_PROCESSING_OPTION` value in the configuration table do not cause unexpected changes to your dialog processing.

## 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 these types of libraries:

#### **ISPMLIB**

Message library

#### **ISPPLIB**

Panel library

#### **ISPSLIB**

Skeleton library

#### **ISPTLIB**

Table input library

#### **ISPTABL**

Table output library

#### **ISPFIL**

File tailoring output file

#### **ISPLLIB**

Load module library

#### **xxxxxxx**

Generic library

#### **ISPILIB**

Image library - The image library is no longer used by ISPF.

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

The DATASET keyword indicates that ID specifies a list of cataloged data sets 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 ISPLLIB 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.

### EXCLDATA

The EXCLDATA keyword indicates that ID specifies a list of cataloged user link library data sets. EXCLDATA can only be used with ISPLLIB.

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 93 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.

## LIBRARY

The LIBRARY keyword associates an allocated ddname with an ISPF data element type. The ID parameter specifies the ddname. See libname.

For example, if application PAYROLL uses panels PAYINIT and PAYTERM, a LIBDEF service request used to identify the panels to ISPF is:

```
ISPEXEC LIBDEF ISPLLIB 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)
```

## EXCLLIBR

The EXCLLIBR keyword associates an allocated user link library ddname with the ISPF link library dialog element type. The ID parameter specifies the ddname. See libname. (Can only be used with ISPLLIB.)

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 93 for a discussion on the effect of the EXCLLIBR specification on member searches.

## dataset-list

Indicates a list of cataloged data set names to be searched for the application. A maximum of 15 data set names can be listed. (See [data-set-list](#) 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.

## 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 ISPLLIB and request that the current ISPLLIB LIBDEF definition be stacked, even when no current ISPLLIB LIBDEF definition exists. When

the ISPLIB LIBDEF definition that requested stacking is removed, there will be no active ISPLIB 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 ISPLIB LIBDEF definition and request that the current ISPLIB definition be stacked, even when no current ISPLIB LIBDEF definition exists. A subsequent reset request of the ISPLIB LIBDEF definition will restore the previously stacked ISPLIB 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.

#### **Note:**

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.

## **Usage notes**

See:

- [“LIBDEF Display utility” on page 90](#)
- [“User link libraries” on page 93](#)
- [“Message libraries” on page 93](#)

## **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, this 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 ISPLLIB

Locates the LIBDEF definition for ISPLLIB

#### LOC ISPMLIB

Locates the LIBDEF definition for ISPMLIB

#### L ISPSLIB

Locates the LIBDEF definition for ISPSLIB

This figure shows a LIBDEF Utility display of all LIBDEF definitions. Figure 5 on page 92 shows a display of a single LIBDEF definition, and Figure 6 on page 92 shows a LIBDEF stacked definition.

LIBDEF Utility			
ISPLLSA		ISPF LIBDEF Display	
		Row 1 to 13 of 16	
Library	Type	USR Identifier	
ISPFIL		** LIBDEF not active **	
ISPLLIB	EXCLDATA	ISPFPROJ.LWGMVS41.LOAD	
		ISPFPROJ.DMTS0.LOAD	
ISPMLIB	DATASET	ISPFPROJ.LWGMVS32.MSGS	
		ISPFPROJ.LWGMVS31.MSGS	
ISPLLIB	DATASET	X	ISPFPROJ.LWG.PANELS
			ISPFPROJ.LWGMVS32.PANELS
			ISPFPROJ.LWGMVS31.PANELS
ISPSLIB	DATASET		ISPFPROJ.RGG.SKELS
ISPTABL	LIBRARY		MYTABLE
			ISPFPROJ.LWGMVS33.TABLES
ISPTLIB			** LIBDEF not active **
MYGEN1	LIBRARY		MYTABLE
Command ==>			
F1=Help	F2=Split	F3=Exit	F7=Backward
F9=Swap	F12=Cancel		F8=Forward
Scroll ==> CSR			

Figure 4. ISPLIBD - all LIBDEF definitions

```

----- LIBDEF Utility -----
ISPLLSA                ISPF LIBDEF Display                Row 1 to 3 of 3

  Library  Type      USR Identifier
  ISPLLIB  DATASET   X   ISPFPROJ.LWG.PANELS
                        ISPFPROJ.LWGMVS32.PANELS
                        ISPFPROJ.LWGMVS31.PANELS
**End**

Command ==>          Scroll ==> CSR
F1=Help      F2=Split  F3=Exit   F7=Backward  F8=Forward
F9=Swap      F12=Cancel

<<<-----

```

Figure 5. ISPLIBD ISPLLIB - ISPLLIB LIBDEF definition

```

----- LIBDEF Utility -----
ISPLLSA                ISPF LIBDEF Display                Row 1 to 4 of 4

  Library  Type      USR Identifier
  ISPLLIB  DATASET   X   ISPFPROJ.LWG.PANELS
                        ISPFPROJ.LWGMVS41.PANELS
S ISPLLIB  DATASET      ISPFPROJ.LWGMVS32.PANELS
                        ISPFPROJ.LWGMVS31.PANELS
**End**

Command ==>          Scroll ==> CSR
F1=Help      F2=Split  F3=Exit   F7=Backward  F8=Forward
F9=Swap      F12=Cancel

<<<-----

```

Figure 6. ISPLIBD ISPLLIB - ISPLLIB 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 ISPLLIB 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).

These 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.

## Return codes

These return codes are possible:

- 0** Normal completion.
- 4** When removing the application library: Application library does not exist for this type.  
When STKADD is specified: There is no existing stack.
- 8** When COND is used: Application library already exists for this type.
- 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.

## Examples

See:

- [“Example 1: The DATASET keyword” on page 94](#)
- [“Example 2: The EXCLDATA keyword” on page 95](#)
- [“Example 3: The LIBRARY keyword” on page 95](#)
- [“Example 4: The EXCLLIBR keyword” on page 96](#)
- [“Example 5: The STACK keyword” on page 96](#)
- [“Example 6: The STKADD keyword” on page 97](#)

### Example 1: The DATASET keyword

Assume that the user has issued these ALLOCATE statements for a panel library before entering ISPF:

```
ALLOCATE DATASET('ISPFPROJ.ABC.MYPAN') FILE(ISPPUSR) SHR
ALLOCATE DATASET('ISPFPROJ.ABC.PANELS') FILE(ISPLIB) SHR
```

Next, the LIBDEF service is invoked with the DATASET keyword to define an application-level panel library (a partitioned data set).

```
ISPEXEC LIBDEF ISPLIB DATASET ID('ISPFPROJ.ABC.APPAN1',
                                'ISPFPROJ.ABC.APPAN2')
```

or alternately

```
CALL ISPLINK('LIBDEF ', 'ISPLIB ', '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 ISPLIB'
or
'ISPEXEC LIBDEF ISPLIB DATASET()'
```

or additionally

```
CALL ISPLINK('LIBDEF ', 'ISPPLIB ', ' ', ' ');
or
CALL ISPLINK('LIBDEF ', 'ISPPLIB ', 'DATASET ', '()');
```

## Example 2: The EXCLDATA keyword

Assume that the user has issued these 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 these 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 ISPLLIB 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 these 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: The STACK keyword

Assume these LIBDEF commands are executed:

```
ISPEXEC LIBDEF ISPLLIB
ISPEXEC LIBDEF ISPLLIB STACK

ISPEXEC LIBDEF ISPLLIB DATASET ID('ISPFPROJ.LWG.PANELS') STACK

ISPEXEC LIBDEF ISPLLIB DATASET ID('ISPFPROJ.LWGMVS33.PANELS') STACK
```

The execution of these commands produces these results:

1. The first LIBDEF resets the ISPLLIB LIBDEF definition. This is considered a "null" definition for ISPLLIB.
2. The second LIBDEF stacks the previous "null" definition for ISPLLIB and resets the ISPLLIB LIBDEF definition. This is the second "null" definition for ISPLLIB.
3. The third LIBDEF stacks the previous "null" definition for ISPLLIB and establishes the ISPLLIB definition for data set 'ISPFPROJ.LWG.PANELS'.
4. The fourth LIBDEF stacks the previous ISPLLIB definition for data set 'ISPFPROJ.LWG.PANELS' and establishes the ISPLLIB definition for data set 'ISPFPROJ.LWGMVS33.PANELS'.

Next, these LIBDEF service calls are issued:

```
ISPEXEC LIBDEF ISPLLIB          (restores 'ISPFPROJ.LWG.PANELS')
Return code = 0

ISPEXEC LIBDEF ISPLLIB          (restores stacked "null" definition)
Return code = 0

ISPEXEC LIBDEF ISPLLIB          (restores stacked "null" definition)
Return code = 0

ISPEXEC LIBDEF ISPLLIB
Return code = 4
```

The preceding service calls produce these 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: The STKADD keyword

Assume these 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 these 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:

Command ==>		LIBDEF Utility ISPF LIBDEF Display	Row 1 to 11 of 11 Scroll ==> PAGE
Library	Type	USR Identifier	
ISPFILIB		** 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:

## LIST

LIBDEF Utility		Row 1 to 11 of 11
ISPF LIBDEF Display		Scroll ==> PAGE
Command ==>		
Library	Type	USR Identifier
ISPFILIB		** LIBDEF not active **
ISPLILIB		** LIBDEF not active **
ISPLLILIB		** LIBDEF not active **
ISPMILIB		** LIBDEF not active **
ISPLLIB	DATASET	ISPFPROJ.ABC.PANELS
		ISPFPROJ.LWG.PANELS
S ISPLLIB		** LIBDEF not active **
S ISPLLIB		** LIBDEF not active **
ISPSLIB		** LIBDEF not active **
ISPTABL		** LIBDEF not active **
ISPTLIB		** LIBDEF not active **

Next, these LIBDEF service calls are issued:

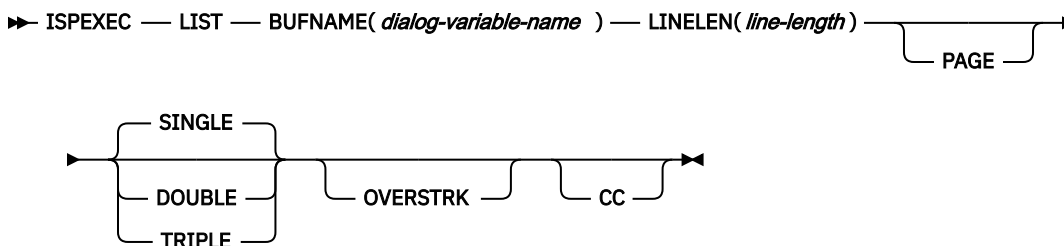
```
ISPEXEC LIBDEF ISPPLIB      (restores stacked "null" definition)
Return code = 0
ISPEXEC LIBDEF ISPPLIB      (restores stacked "null" definition)
Return code = 0
ISPEXEC LIBDEF ISPPLIB
Return code = 4
```

## 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

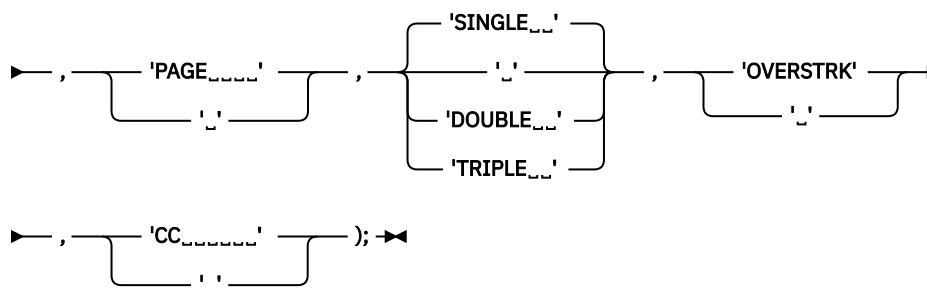


## Call invocation format

➤ CALL — ISPEXEC — (*buf-len*, — *buffer*); ➤

or

➔ CALL — ISPLINK — ('LIST\_...' — , — *dialog-variable-name* , — *line-length* — ➔



## 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

These return codes are possible:

## LIST

- 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.

D 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 shown 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 shown:

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.

## Examples

See:

- [“Example 1” on page 102](#)
- [“Example 2” on page 102](#)
- [“Example 3” on page 102](#)
- [“Example 4” on page 103](#)

### 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**

These 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 command:

```
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 a command procedure.

## Return codes

The compress request exit routine is responsible for handling all errors that occur while it is in control. The compress exit must pass the return codes to LMCOMP. See [z/OS ISPF Planning and Customizing](#) for information about the Compress Exit.

These return codes are possible:

**0**

Successful completion.

**8**

Library type is a PDSE and cannot be compressed

**10**

No data set associated with the given data ID.

**12**

One of these:

- Data set not partitioned
- Data set specified not allocated
- Data set is open
- Data set is not movable
- Data set must be allocated exclusively. Use ENQ(EXCLU) in LMINIT service.
- Concatenated libraries are not allowed for LMCOMP.

**20**

Severe error; unable to continue.

## Example

This example invokes the LMCOMP service to compress the data set associated with the data ID in variable DDVAR.

### Command invocation

```
ISPEXEC LMCOMP DATAID(&DDVAR)
```

### Call invocation

```
CALL ISPLINK('LMCOMP ',DDVAR);
```

or

Set the program variable BUFFER to contain:

```
BUFFER = 'LMCOMP DATAID(&DDVAR)';
```

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

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

## 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, replacing members, and automatic truncation are optional. Only fixed-length and variable-length data sets can be packed.

Completion of the LMINIT service is required before you can invoke LMCOPY. You must specify ENQ(MOD) with the LMINIT service if you want to use LMCOPY to append records to the "to-data-id". See [“LMINIT—generate a data ID for a data set” on page 130](#) 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.

### Note:

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 data sets and non-load data sets as well as for PDS and PDSE data sets.

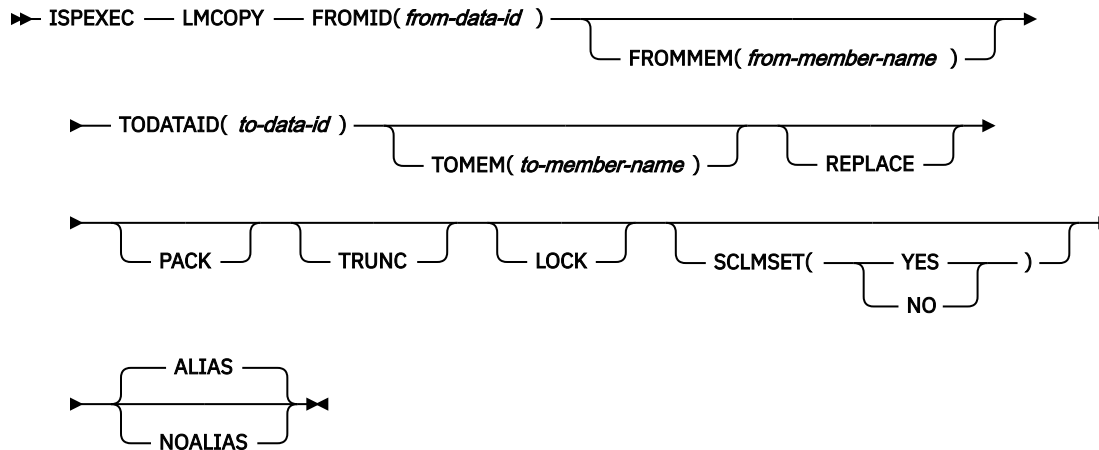
Copying to the same data set is not supported when aliases are automatically selected, as it would result in the "from" and "to" member names being the same.

4. If the NOALIAS option is in effect, LMCOPY does not copy alias members unless one of these 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.

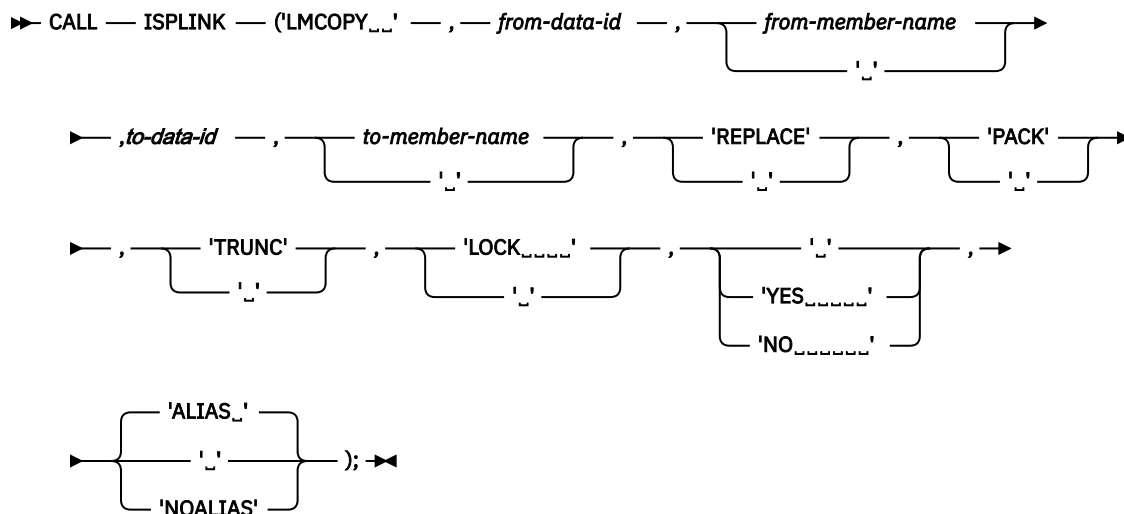
If the NOALIAS option is in effect, copying an alias member by itself will result in a new member being created, even if the main member has already been copied.

5. If *from-data-id* represents an empty sequential data set, LMCOPY performs the copy but sets the return code to 4 as a warning.
6. When the LMCOPY service is used to copy a member in a PDSE version 2 data set that is configured for member generations, only the current generation of the member is copied.

## Command invocation format



## Call invocation format



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. If an asterisk (\*) or a pattern is used, a member count is returned in these dialog variables:

#### ZSVCCPY

Number of members copied

#### ZSVCNCPY

Number of members not copied

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 LOCK and have the LMCOPY ignore the value, change the FAIL\_ON\_LMF\_LOCK keyword value in the ISPF Configuration Table to NO.

**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

These return codes are possible:

**0**

Normal completion.

**4**

Member not available, which indicates one of these situations:

- The "from" data set is empty.
- No members matched the specified pattern in the "from" data set.

**8**

- The *from-member-name* was not found.
- The same name was specified for *to-member-name* and *from-member-name*.

**10**

No data set is associated with the given data ID.

**12**

One of these:

- 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", either by you or by 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(&DDVAR2) REPLACE PACK TRUNC';
```

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

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

## LMDDISP—data set list service

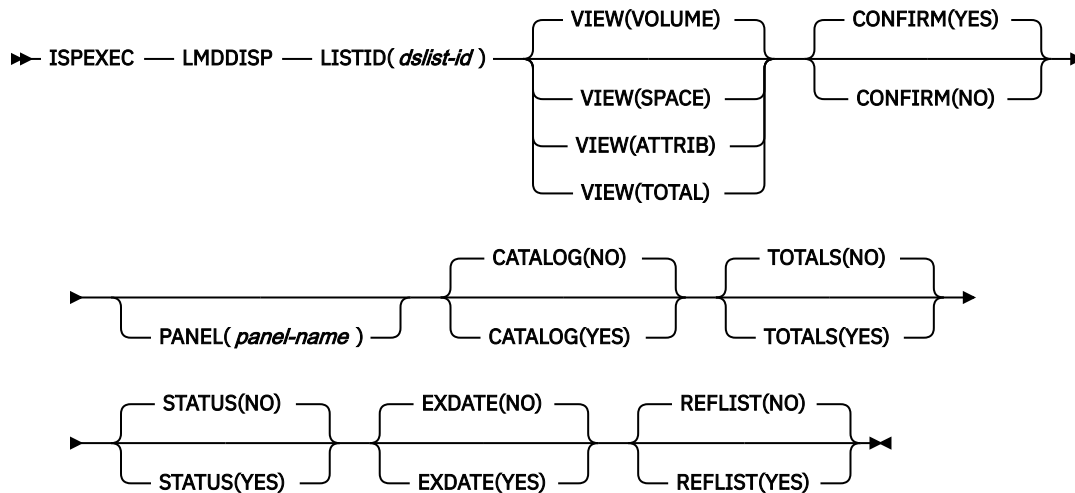
The LMDDISP service allows you to write your own data set list dialog. This service is similar to ISPF option 3.4, the data set list utility, which displays the list of data sets. The LMDDISP service displays any

## LMDDISP

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.

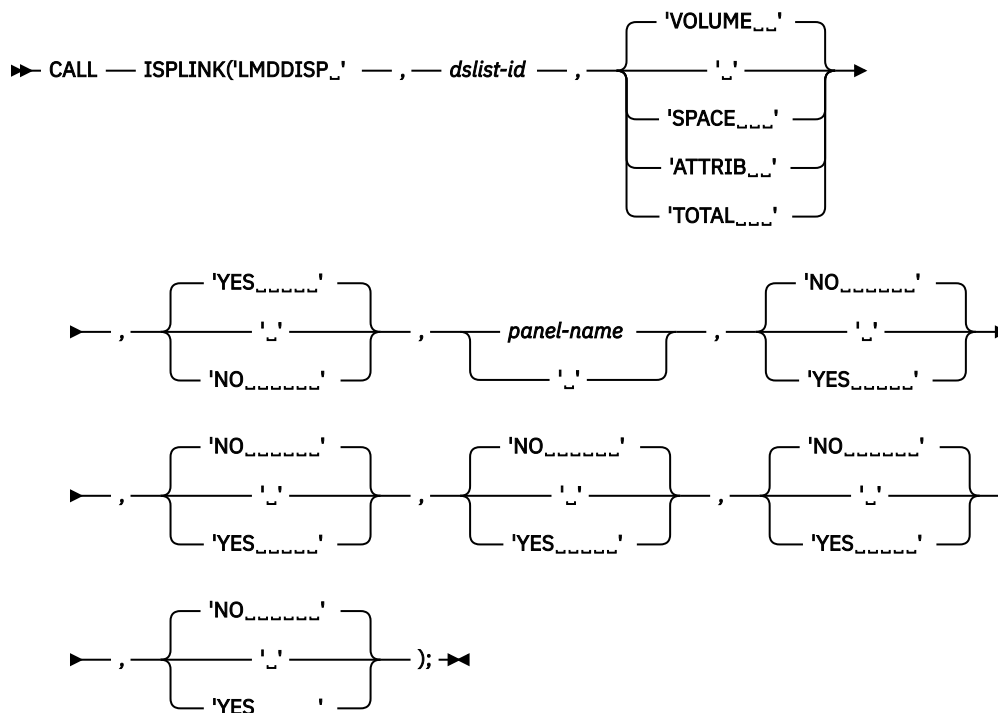
## Command invocation format



## Call invocation format

➤➤ CALL — ISPEXEC — (*buf-len*, *buffer*); ➤➤

or



## 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 114](#).

### **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 (CONFIRM)**

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. See [z/OS ISPF Planning and Customizing](#) for more information on developing a customized panel.

### **YES|NO (CATALOG)**

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.

### **YES|NO (TOTALS)**

This parameter controls whether the total tracks of all datasets, the total tracks of all non-excluded data sets, the number of all data sets and the number of all non-excluded data sets in the list is displayed in an additional header line above the column descriptions. This parameter is ignored for the VOLUME and ATTRIB view. The default for the SPACE and TOTAL view is NO. Processing time for the initial view will increase depending on the size of the data set list, as the track information for all data sets has to be collected up front. The progress of the data collection can be displayed in a popup panel by selecting option STATUS.

### **YES | NO (STATUS)**

This parameter controls, whether a popup panel with a progress bar is displayed during the collection of the data set information. The parameter is only applicable when TOTALS is selected on the SPACE or TOTAL view. Note: The progress panel will only be displayed if the data set list contains at least 50 data sets.

### **YES | NO (EXDATE)**

This parameter controls whether the expiration date or the referred date is displayed on the TOTAL view of the data set list. By default, the referred date is displayed.

### **YES | NO (REFLIST)**

This parameter controls whether the dsname level supplied to the corresponding LMDINIT is added to the ISPF reference list.

**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

These 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 example shown in [“Command invocation” on page 112](#) is 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 ',DSLIDID,'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 command:

```
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 13.](#)

**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 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:

- 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.

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. When the Call Invocation format using the ISPLINK interface is used, the *dsname-level* parameter supports system symbols. 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. When the Call Invocation format using the ISPLINK interface is used, the volume-serial parameter supports system symbols. This field has the same restrictions and syntax as the Volume field under ISPF, option 3.4. See the [z/OS ISPF User's Guide Vol II](#) for a complete description.

**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

- 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 13.
- 12** A keyword value is incorrect.
- 16** A truncation or translation error occurred in accessing dialog variables.
- 20** Severe error; unable to continue.

## Examples

Here are some examples of the LMDINIT service:

### Example 1:

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 command:

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

## Example 2:

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 command:

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

### Call invocation

In a CLIST, define the variable DSNLVL to contain a data set prefix of SYS2, and the second qualifier to that of the sysplex on which the command is executed.

```
SET DSNLVL = SYS2.&&SYSPLEX
ISPEXEC VSYM (DSNLVL)
```

When executed on a system that is a member of a sysplex named SYSPLEX1, the resulting value of DSNLVL is SYS2.SYSPLEX1.

The same example in REXX is:

```
DSNLVL = 'SYS2.&SYSPLEX'
address "ISPEXEC"
"VSYM (DSNLVL)"
```

## Example 3:

In this example the LMDINIT service generates a dslist ID for a data set list containing only the second most recent generation within the generation data group "PROD.GDG".

### Command invocation

```
ISPEXEC LMDINIT LISTID(VARNAME) LEVEL(PROD.GDG(-1));
```

### Call invocation

```
DCL DSVAR CHAR(8)
CALL ISPLINK ('VDEFINE ', 'DSVAR ', DSVAR, 'CHAR ',L8);

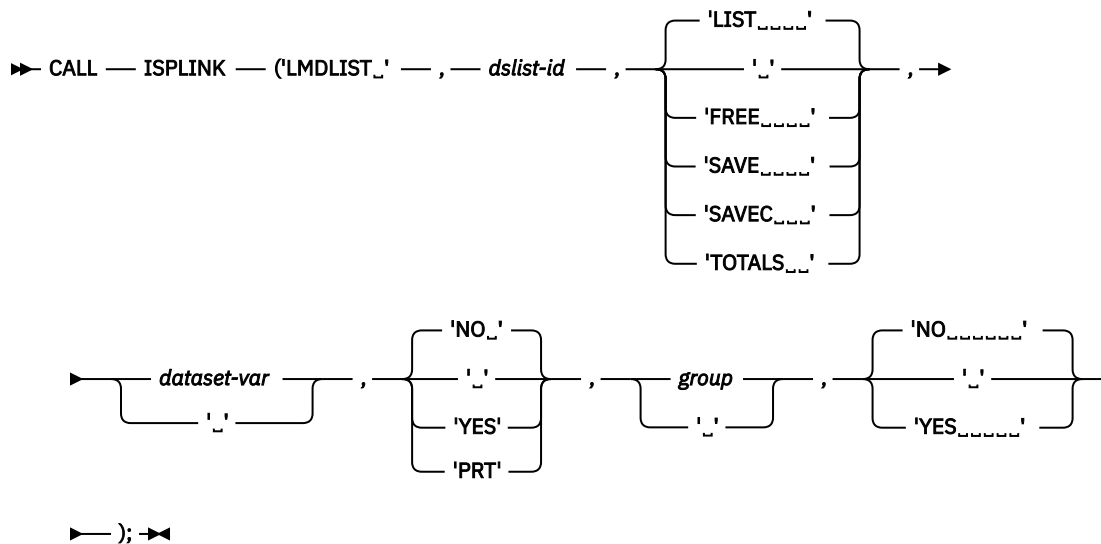
CALL ISPLINK ('LMDINIT ', 'DSVAR ', 'PROD.GDG(-1)');
```

or

Set the program variable BUFFER to contain:



## Call invocation format



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 114](#).

### LIST|FREE|SAVE|SAVEC|TOTALS

These options determine whether the LMDLIST service returns the names on the internal list to the dialog, frees the storage used by the list, writes the list to a data set, or returns the number of total tracks and the number of all data sets in the list in dialog variables ZDLSIZET and ZDLDST in the function pool.

#### 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. Any 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.

**TOTALS**

The TOTALS option returns the number of total tracks and the number of all data sets from the internal list into these dialog variables in the function pool without writing the list to a data set:

**ZDLSIZET**

Total number of tracks of all data sets in the list.

**ZDLNST**

Total number of data sets in the list.

This information is also provided with options SAVEC and SAVE when the STATS parameter is set to YES or PRT.

As processing time increases depending on the size of the data set list when either the STATS or the total tracks are collected, the progress of the data collection can be displayed in a pop-up panel by selecting the option STATUS.

**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|PRT (STATS)**

Use the STATS parameter with the LMDLIST service LIST and SAVE options. The default is STATS(NO). If you specify STATS(PRT) together with the SAVE or SAVEC option and you write the data set list to the ISPF LIST dataset, a total tracks header line is printed at the beginning of the data set list, showing the total number of datasets in the list and the total number of tracks of all datasets in the list. If you specify STATS(YES) or STATS(PRT), the LMDLIST service provides statistical information with the data set names in these dialog variables in the function pool or prints the statistical information to the data set:

*Table 7. List of dialog variables containing information about a data set*

Variable	Description
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.
ZDLSIZEEX	Data set size in tracks, long format (12 bytes).
ZDLUSED	Percentage of used tracks or pages (PDSE).
ZDLEXT	Number of extents used.
ZDLEXTX	Number of extents used, long format (5 bytes).
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.

Table 7. List of dialog variables containing information about a data set (continued)

Variable	Description
ZDLDSNTP	Dsname type (PDS, LIBRARY, or ' ').
ZDLSPACU	Space units.
ZDLMVOL	Whether the data set is multivolume (Y) or not (N).
ZDLCATNM	Name of the catalog in which the data set was located.
ZDLOVF	Whether variables ZDLEXTX and ZDLSIZE should be used to obtain the 'number of extents used' and 'data set size in tracks' values (YES or NO). The value is YES when the 'number of extents used' value exceeds the size of variable ZDLEXT or the 'data set size in tracks' value exceeds the size of variable ZDLSIZE.
ZDLEATR	Extended attribute indicator.
ZDLCJOB	Create jobname.
ZDLCSTPN	Create stepname.
ZDLDSNV	Data set version.
ZDLNGEN	Maximum number of generations

**Note:** ISPF cannot calculate reliable space utilization values for BDAM data sets. Therefore, the LMDLIST service returns question marks (?) in variables that contain space utilization data when reporting on BDAM data sets.

#### 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 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 DISP=OLD for the SAVE option.

#### YES | NO (STATUS)

This parameter controls, whether a pop-up panel with a progress bar is displayed during the collection of dataset information. This parameter is only applicable for options SAVE and SAVEC with STATS(YES) or STATS(PRT) or option TOTALS.

**Note:** The progress panel will only be displayed if the data set list contains at least 50 data sets.

#### 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

These return codes are possible:

### 0

One of these:

- 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. The total number of tracks and datasets are returned to dialog variables in the function pool, if requested.
- **SAVEC** option - Normal completion. The data set list has been successfully written to a data set. The total number of tracks and datasets are returned to dialog variables in the function pool, if requested.
- **TOTALS** option - Normal completion. No list has been written to a dataset. The total number of tracks and datasets are returned into dialog variables in the function pool.

**4**

One of these:

- No data sets matched specified search criteria (the values for keywords **LEVEL** and **VOLUME** on the **LMDINIT** service).
- An incomplete **VTOC** list. An entry was found in the **VTOC** index but the volume was not available. The name in the index has not been added to the data set list.

**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.

## Examples

Here are some examples of the **LMDLIST** service:

### Example 1:

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 command:

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

## Example 2:

In this example, the LMDINIT service creates a LISTID based on the data set name level ISPF SVC. The LMDLIST service saves the list into a data set called userid.ISPF SVC.DATASETS. The GROUP parameter told the LMDLIST service to use ISPF SVC in the data set name of the saved list. The LMDFREE service frees the LISTID.

```
/* REXX exec to create a list of data sets */
address ispxexec
' LMDINIT LISTID(LISTIDV)  LEVEL(ISPF SVC)'
If rc = 0 Then
  do
    ' LMDLIST LISTID('listidv') OPTION(SAVE) GROUP(ISPF SVC)'
    ' LMDFREE LISTID('listidv')'
  end
```

## Example 3:

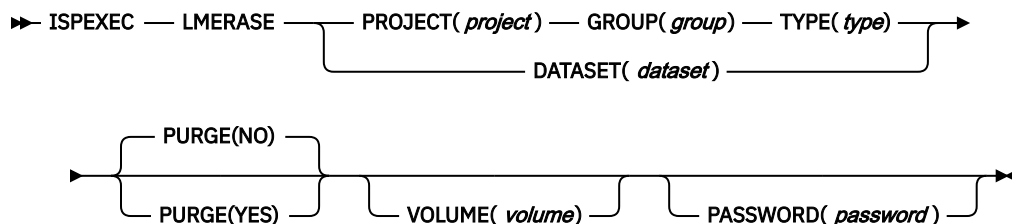
In this example, the LMDINIT service creates LISTID of the current user's data sets, and blanks out the data set name variable to prepare for the LMDLIST service. The LMDLIST service is called from within a loop, so that all data sets that match the criteria from the LMDINIT service are returned, one at a time. The LMDLIST service sets the ZDLDSORG variable, which returns PO if the data set is partitioned. For a partitioned data set, the LMINIT service is called, so that the data set can be compressed with the LMCOMP service. CONTROL ERRORS RETURN is set before the LMCOMP service so that if there is a compression error, the exec continues to run. CONTROL ERRORS CANCEL is set after the compress so that the exec halts if there is an error on any of the other services. LMFREE frees the data id, and LMDFREE frees the LISTID.

```
/* REXX exec to loop through a user's data sets and compress PDSes */
address ispxexec
' LMDINIT LISTID(lidv)  LEVEL("userid()")'
If rc = 0 Then
  do
    dsvar = " ";
    keepon = "yes"
    do until keepon = "no "
      ' LMDLIST LISTID("lidv") OPTION(list) dataset(dsvar) stats(yes)'
      If rc ^= 0 Then
        Do
          if rc ^= 8 then
            say "lmdlist failed with rc = " rc
            keepon = "no "
          End
        else
          do;
            if ZDLDSORG = "PO" then
              do;
                ' LMINIT DATAID(daid) DATASET('"dsvar"') ENQ(EXCLU)'
                if rc = 0 then
                  do
                    say "compressing "dsvar
                    "CONTROL ERRORS RETURN"
                    "LMCOMP DATAID("daid")"
                    "CONTROL ERRORS CANCEL"
                    "LMFREE DATAID("daid")"
                  end
                else
                  say "lminit failed with rc = "rc
              end;
            end
          end
        end
      ' LMDFREE LISTID("lidv")'
    end
```

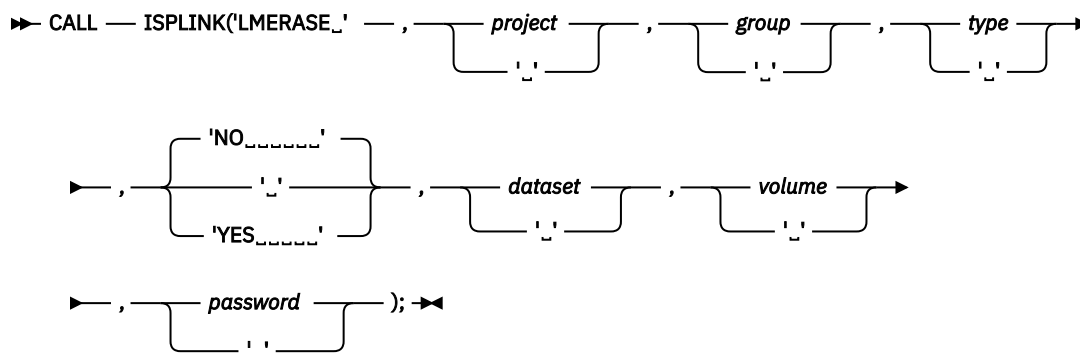
## LMERASE—erase a data set

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



## Call invocation format



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 (PURGE)

If YES is specified, LMERASE deletes the data set regardless of its expiration date. If NO 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, 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-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**

These return codes are possible:

**0**

Normal completion.

**8**

One of these:

- 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 command:

```
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

These return codes are possible:

## LMGET

- 0** Normal completion.
- 8** Free data ID failed; the error condition is described in [“System variables used to format error messages” on page 13](#).
- 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 command:

```
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 this format:

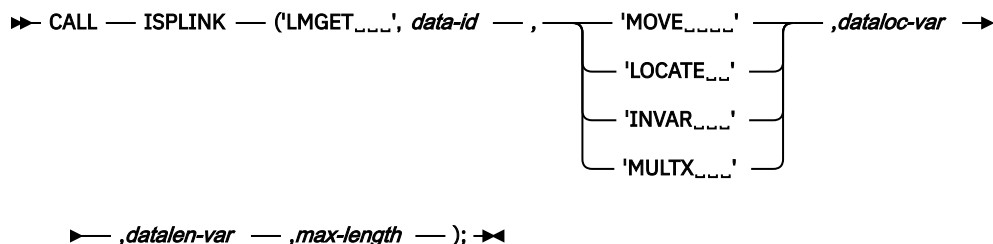
```
+-----+-----+-----+-----+-----+-----+
|len  | record  |len  | record  |len  | record  |
+-----+-----+-----+-----+-----+-----+
<----- this length is returned in datalen-var ----->
```

The data read is always unpacked. If the data set contains packed data, LMGET unpacks the data.

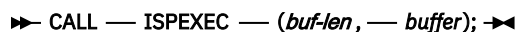
## Command invocation format



## Call invocation format



or



## 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.

## LMGET

### 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

These 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 these:

- 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 1

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 command:

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



## Example 2

This example initializes and opens an input and an output data set, using LMINIT and LMOPEN respectively. The exec loops through the all the records of the input data set using the LMGET service. Each record of the input data set is stored into the variable *srchline*. The variable *srchline* is parsed to find a member name. The string 'SELECT ' followed by the member name is stored into a variable called *selline* which is added to the output data set, using the LMPUT service. The input and output data sets are closed and freed using LMCLOSE and LMFREE.

```

/* rexx *****/
/*
/* Member:  SETSRCH
/*
/* Purpose: To create a srchfor.stmts data set that contains
/*           only those members with a certain string. For example,
/*           if both CLIST and REXX execs are stored in the same
/*           library, but you only want to search the REXX members
/*           search the library for the word rexx, with the options
/*           listed below to create srchfor.list. Use the resulting
/*           statements data set as input to the next search.
/*
/* Setup:   Create the srchfor output using option 3.15 with
/*           process options:
/*           LMT0 NOSUMS NOPRTCC
/*           to create an output listing data set called
/*           userid.srchfor.list without a summary section or page
/*           dividers.
/*
/* Input:   userid.SRCHFOR.LIST (as created in setup).
/*
/* Output:  userid.SRCHFOR.STMTS containing a SELECT statement
/*           for each member to be searched.
/*
*****/
address ispxexec
'lmnit dataid(srchout) dataset(srchfor.list) enq(shr) '
if rc = 0 then
do
  'lmopen dataid('srchout') option(input)'
  if rc = 0 then
  do
    'lmnit dataid(srchstmt) dataset(srchfor.stmts) enq(exclu)'
    if rc = 0 then
    do
      'lmopen dataid('srchstmt') option(output)'
      if rc = 0 then
      do
        done = 'n'
        datavar = 0
        linenum = 1
        do while done = 'n'
          'lmget dataid('srchout') mode(invar) dataloc(srchline)
            datalen(datavar) maxlen(132)'
          if rc = 0 then
          do
            linenum = linenum + 1
            if linenum > 5 then /* skip over header */
              do
                parse var srchline memname linesfnd linesrch
                if memname /= ' ' then
                do
                  selline = 'SELECT ' memname
                  'lmput dataid('srchstmt') mode(invar),
                    dataloc(selline) datalen(80)'
                  if rc > 0 then
                    done = 'y'
                end
              end
            end
          end
          done = 'y'
        end
      end
    end
  end
  'lmclose dataid('srchstmt ')'
end
  'lmfree dataid('srchstmt ')'
end
  'lmclose dataid('srchout ')'
end

```

```

    'lmfree dataid('srchout ')'
end
exit

```

## Example 3 (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 that are created by a method that does not set the format 1 or format 8 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.

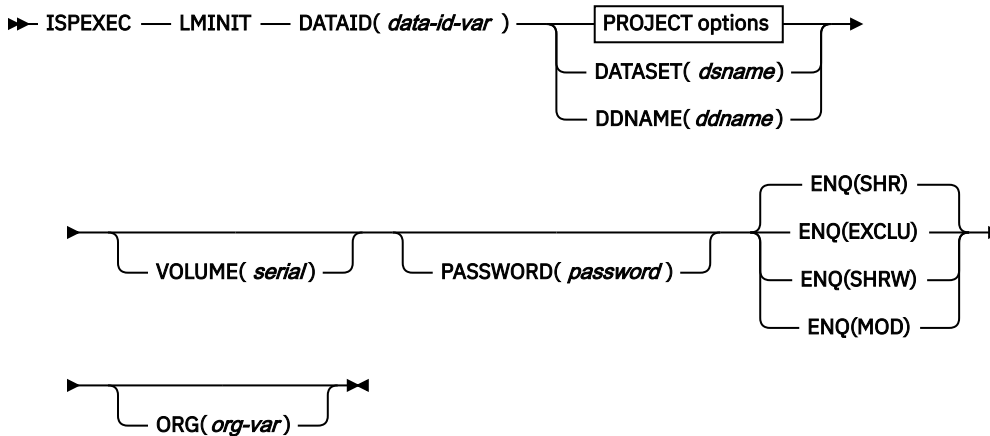
For each LMINIT invocation, you should invoke a matching LMFREE service. The LMFREE service removes the data ID generated by LMINIT. Invoke the LMFREE service when the data ID is no longer needed. Otherwise, the ISPF library or data set associated with the data ID is not released until ISPF terminates.

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.

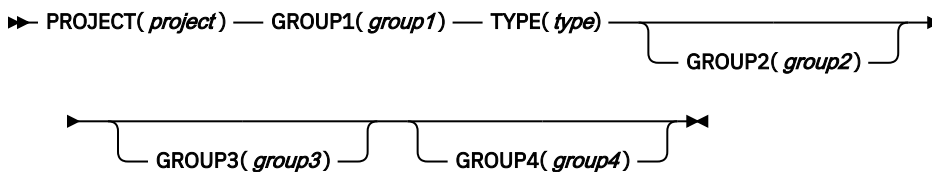
For example, if you use a service like LMCOPY or LMMOVE to modify a data ID that was defined by the LMINIT service, and the modified resource is needed for other services, then the data ID that references the modified resource must first be freed with LMFREE, then re-allocated with LMINIT. In more specific

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.

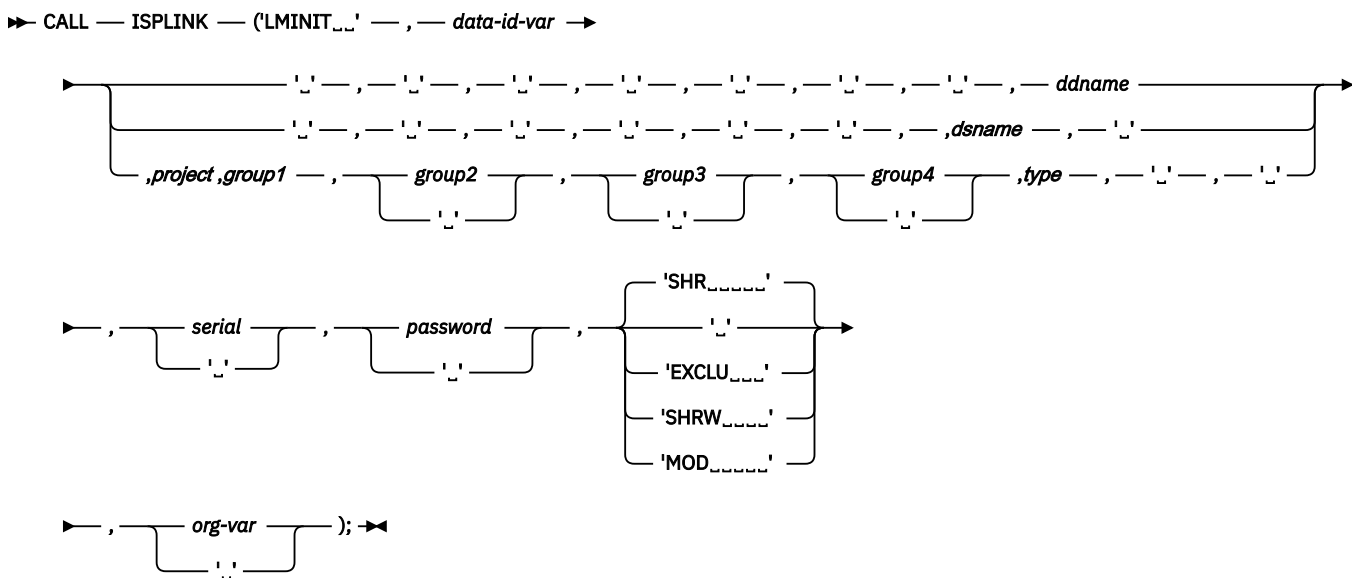
## Command invocation format



### PROJECT options



## Call invocation format



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.

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. 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. 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, 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 176). Do not specify a password for RACF-protected data sets. The maximum length of this parameter is 8 characters.

**SHR|EXCLU|SHRW|MOD**

The requirements for enqueueing (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 ISPF 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

These 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 13](#).

**12**

The parameter value is invalid.

**16**

Truncation or translation error in accessing dialog variables.

**20**

Severe error; unable to continue.

See [“System variables used to format error messages” on page 13](#) for more information about dialog variables.

**Note:** Data sets allocated with an XTIOF will return a "DDNAME Not Found" message and set RC=8 if XTIOF support is not fully enabled.

## Examples

Here are some examples of the LMINIT service:

### Example 1:

This example invokes the LMINIT service to associate a data ID with data concatenated from these 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 command:

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

## Example 2:

The example shown in “[Command invocation](#)” on page 135 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');
```

## Example 3:

The example shown in “[Command invocation](#)” on page 135 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.

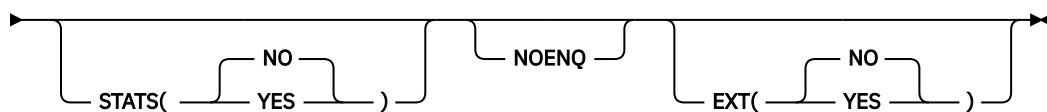
```
CALL ISPLINK ('LMINIT','DDVAR','MYDD');
```

## LMMADD—add a member to a data set

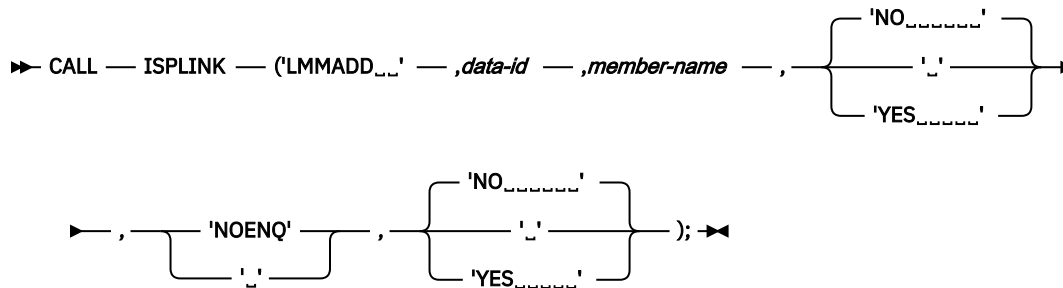
The LMMADD service adds a member to the specified ISPF library or MVS partitioned data set. LMMADD then updates the data set directory with information about the member to be added. If the member already exists, the member name entry is not added. The LMINIT with either ENQ(SHRW) or ENQ(EXCLU), LMOOPEN with OPTION(OUTPUT), and LMPUT services must be completed before LMMADD is used.

### Command invocation format

➔ ISPEXEC — LMMADD — DATAID( *data-id* ) — MEMBER( *member-name* ) ➔



## Call invocation format



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 (STATS)

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 ISPF 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 or ZLC4DATE) and the date of most recent change (ZLMDATE or ZLM4DATE) must be provided in the member statistics.

If you specify NO, statistics are not updated.

These dialog variables are used to pass statistical information from the dialog invoking the LMADD 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. Use system variable ZDATEF to determine the national format. Either ZLCDATE or ZLC4DATE is required. If both ZLCDATE and ZLC4DATE are entered, ZLCDATE is used.

### ZLMDATE

Last change date, a character value shown in your national format. Use system variable ZDATEF to determine the national format. Either ZLMDATE or ZLM4DATE is required. If both ZLMDATE and ZLM4DATE are entered, ZLMDATE is used.

### 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

- When extended statistics are not enabled in the site configuration, this variable contains a number in the range 0 - 65535.
- When extended statistics are enabled in the site configuration, this variable contains a number in the range 0 - 2147483647.

If no value exists for this variable, ISPF sets the value to blanks.

## ZLINORC

Initial number of records

- When extended statistics are not enabled in the site configuration, this variable contains a number in the range 0 - 65535.
- When extended statistics are enabled in the site configuration, this variable contains a number in the range 0 - 2147483647.

## ZLMNORC

Number of changed records

- When extended statistics are not enabled in the site configuration, this variable contains a number in the range 0 - 65535.
- When extended statistics are enabled in the site configuration, this variable contains a number in the range 0 - 2147483647.

## ZLUSER

User ID of the last user to change the given member; the user ID has a maximum length of 7 characters. To specify an 8-character user ID, you must use the ZLUSER8 variable.

## ZLC4DATE

Creation date, a character variable shown in your national format with a 4-character year. Use system variable ZDATEF to determine the national format. Either ZLCDATE or ZLC4DATE is required. If both ZLCDATE and ZLC4DATE are entered, ZLCDATE is used.

## ZLM4DATE

Last modified date, a character variable shown in your national format with a 4-character year. Use system variable ZDATEF to determine the national format. Either ZLMDATE or ZLM4DATE is required. If both ZLMDATE and ZLM4DATE are entered, ZLMDATE is used.

## ZLUSER8

User ID of the last user to change the given member. When 8-character user IDs are enabled on the system, the user ID has a maximum length of 8 characters. When 8-character user IDs are not enabled on the system, the user ID has a maximum length of 7 characters.

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 ISPF 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. By default, ISPF will issue the ENQ unless NOENQ is specified.

**YES|NO (EXT)**

Starting in z/OS V2R3 ISPF, you do not need to specify this parameter. Extended statistics are automatically generated if they are enabled in the site configuration and any of the line count values exceed 65535.

**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**

These 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 these:

- 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 command:

```
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 LMOOPEN 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 LMOOPEN must have been done for OUTPUT. An LMINIT with ENQ(EXCLU) is required when MEMBER(\*) is specified.

**Note:** When the LMMDEL service is used to delete a member in a PDSE version 2 data set that is configured for member generations, the member and any previous generations of the member are deleted.

### 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.

Where *member-name* is the name of a primary member, the primary name and all associated alias names are deleted. Where *member-name* is an alias member, only the alias name and its directory entry are deleted.

Where a member pattern has been specified for the LMMDEL service, these rules apply:

- All primary members whose name matches the member pattern are deleted.
- All aliases that are associated with a primary member whose name matches the member pattern are deleted, even if the alias name itself does not match the member pattern.
- All aliases whose name matches the member pattern are deleted, even if the alias is associated with a primary member whose name does not match the member pattern.

#### NOENQ

An optional parameter that specifies that ISPF 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. By default, ISPF 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**

These 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 these:

- 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 command:

```
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

Table 8 on page 141 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).

<i>Table 8. Variables saved by LMMDISP in the function pool</i>		
<b>Variable Name</b>	<b>Returned</b>	<b>Variable Description</b>
ZLAC	STATS(YES)	2-character field containing the authorization code of the member.
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; a number in the range 0 - 65535.
ZLCNORCE	STATS(YES)	10-character field. If ZLEXT=YES, contains the current number of records; a number in the range 0 - 2147483647.
ZLEXT	STATS(YES)	3-character field. If ZLEXT=YES, then ZLCNORCE, ZLINORCE, and ZLMNORCE contain values.
ZLINORC	STATS(YES)	Initial number of records; a number in the range 0 - 65535.

<i>Table 8. Variables saved by LMMDISP in the function pool (continued)</i>		
<b>Variable Name</b>	<b>Returned</b>	<b>Variable Description</b>
ZLINORCE	STATS(YES)	10-character field. If ZLEXT=YES, contains the initial number of records; a number in the range 0 - 2147483647.
ZLLCMD	always	Line command used to select the member.
ZLLIB	STATS(YES)	Number from 1 to 16 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; a number in the range 0 - 65535.
ZLMNORCE	STATS(YES)	10-character field. If ZLEXT=YES, contains the number of modified records; a number in the range 0 - 2147483647.
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 <sup>1</sup>	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)	User ID of the last user to change the member; an alphanumeric field with a maximum length of 7 characters. When the user ID of the last user to change the member is an 8-character value, this variable contains the value '>7CHARS'; the 8-character user ID can be obtained from the ZLUSER8 variable.
ZLUSER8	STATS(YES)	User ID of the last user to change the member; an alphanumeric field with a maximum length of 8 characters.
ZLVERS	STATS(YES)	PDF version number.
ZSCLM	STATS(YES)	Indicates whether the system was last modified by SCLM or ISPF.

## 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.

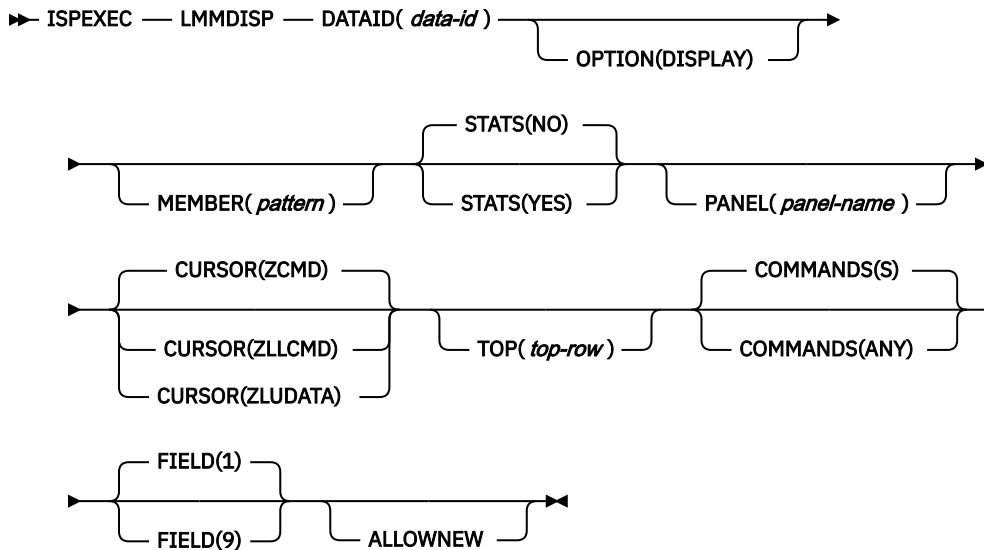
<sup>1</sup> ZLPDSUDA is put in the ISPF function pool only if STATS(YES) was specified and the selected member being returned had member statistics that did not conform to ISPF standards. For example, a load module member of a partitioned data set usually has load module statistics, and not ISPF statistics.

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 MEMBER, COMMANDS, and FIELD are ignored after a member list has been created until it is freed by an LMMDISP invocation with OPTION(FREE).

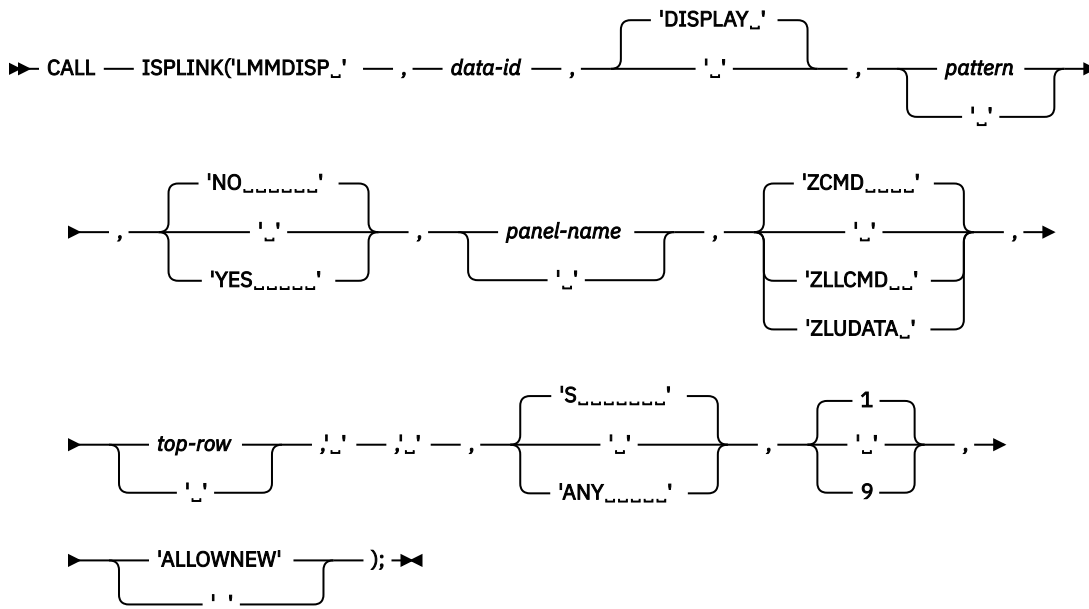
When the member list panel is displayed, you can select members for processing by entering valid line commands next to the member names or by using the SELECT primary command.

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



## Call invocation format



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. See the [z/OS ISPF User's Guide Vol I](#) for a more complete description of patterns and pattern matching.

### YES|NO (STATS)

Indicates if LMMDISP is to return member statistics via dialog variables. See “[Dialog variables](#)” on [page 141](#) for a list of the dialog variables.

### panel-name

The name of the panel on which the member list is to be displayed. See [z/OS ISPF Planning and Customizing](#) for the 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.

### ALLOWNEW

Indicates that nonexistent members can also be selected. Omitting this parameter causes only existing members to be selected.

### 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

These 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 these 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)';
```

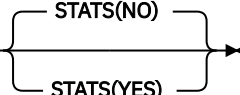
Set the program variable BUFLen to the length of the variable BUFFER. Issue the command:

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

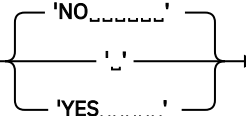
## GET option

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) — 

### Call invocation format

►► CALL — ISPLINK('LMMDISP\_' — , — *data-id* — , 'GET\_.....' — , ' ' — ,   
 ►► ); ►►

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 (STATS)

Indicates whether LMMDISP is to return member statistics through dialog variables to the dialog. See [“Dialog variables” on page 141](#) 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 these 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 command:

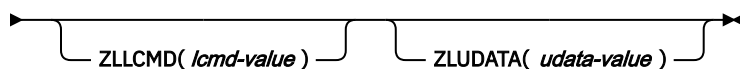
```
CALL ISPEXEC (BUFLen,BUFFER);
```

## 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* ) ►►



## Call invocation format

```
➤ CALL — ISPLINK('LMMDISP_' — , — data-id — , 'PUT_XXXX' — , 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 the 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 these 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 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\_XXXX' — , *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 these 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 command:

```
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 these 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. Enter the command:

```
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 these 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 command:

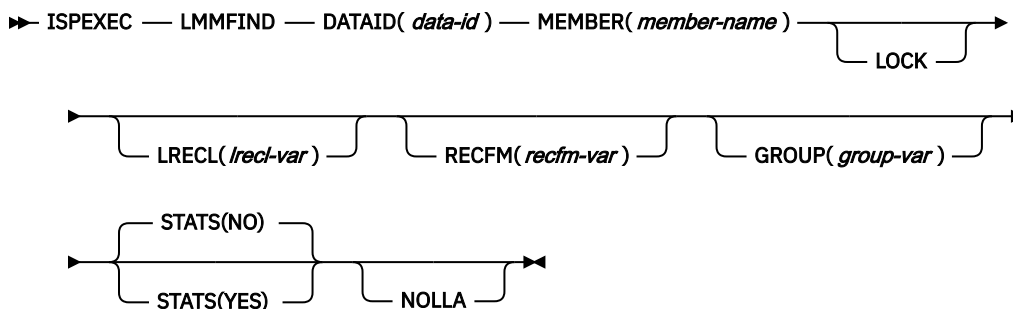
```
CALL ISPEXEC (BUFLN,BUFFER);
```

## LMMFIND—find a library member

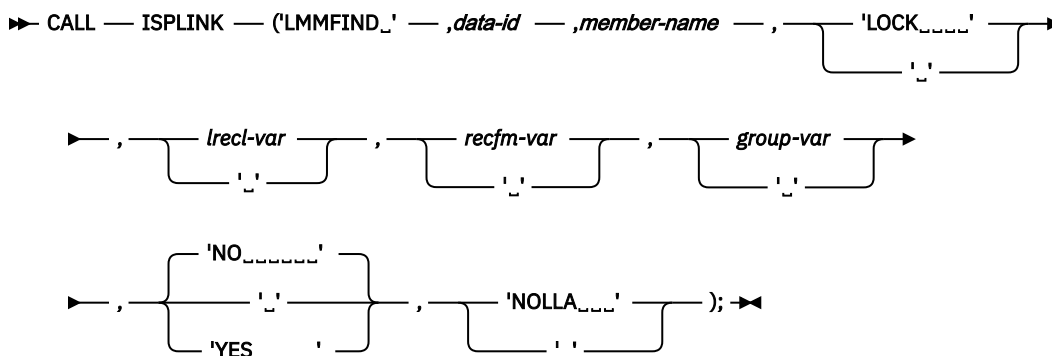
The LMMFIND service finds a specified member of an ISPF library or partitioned data set associated with a given data ID. You can also use LMMFIND to return member statistics to you. If the data ID represents a concatenated set of ISPF libraries, LMMFIND finds the first occurrence of the member in the set of libraries.

The LMINIT and LMOPEN services must be completed before LMMFIND can be used.

## Command invocation format



## Call invocation format



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 LOCK and have LMMFIND ignore the value, change the value of the FAIL\_ON\_LMF\_LOCK keyword in the ISPF Configuration Table to NO.

### **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 (STATS)**

Whether statistics for the member are to be returned to the dialog invoking the service. If you specify NO, 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 these 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. These 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 16.

**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 these dialog variables:

**ZLC4DATE**

Creation date, a character variable shown in your national format with a 4-character year. Use system variable ZDATEF to determine the national format. If no value exists for this variable, ISPF sets the value to blanks.

**ZLCDATE**

Creation date, a character value shown in your national format. Use system variable ZDATEF to determine the national format. If no value exists for this variable, ISPF sets the value to blanks.

**ZLCNORC**

Current number of records; a number in the range 0 - 65535. If no value exists for this variable, ISPF sets the value to blanks.

**ZLCNORCE**

10-character field. If ZLEXT=YES, contains the current number of records; a number in the range 0 - 2147483647.

**ZLEXT**

Indicates whether extended PDS statistics are available. The length is 3, and possible values are blanks or YES. If ZLEXT has a value of YES, variables ZLCNORCE, ZLINORCE, and ZLMNORCE contain values.

**ZLINORC**

Initial number of records; a number in the range 0 - 65535.

**ZLINORCE**

10-character field. If ZLEXT=YES, contains the initial number of records; a number in the range 0 - 2147483647.

**ZLLIB**

Position in concatenated data set sequence; a number from 1 to 16.

**ZLM4DATE**

Last change date, a character variable shown in your national format with a 4-character year. Use system variable ZDATEF to determine the 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. Use system variable ZDATEF to determine the national format. If no value exists for this variable, ISPF sets the value to blanks.

**ZLMNORC**

Number of modified records; a number in the range 0 - 65535.

**ZLMNORCE**

10-character field. If ZLEXT=YES, contains the number of modified records; a number in the range 0 - 2147483647.

**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.

**ZUSER**

User ID of the last user to change the given member; an alphanumeric field with a maximum length of 7 characters. When the user ID of the last user to change the given member is an 8-character value, this variable contains the value '>7CHARS'; the 8-character user ID can be obtained from the ZUSER8 variable.

**ZUSER8**

User ID of the last user to change the given member; an alphanumeric field with a maximum length of 8 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 ISPF. A value of Y indicates the last update was made through SCLM. A value of N indicates that the last update was made through ISPF.

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.

**NOLLA**

If LLA is used to manage a cached directory entry, specify this keyword to ensure that the cached entry is not used.

**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

These 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 these:

- 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. Enter the command:

```
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.

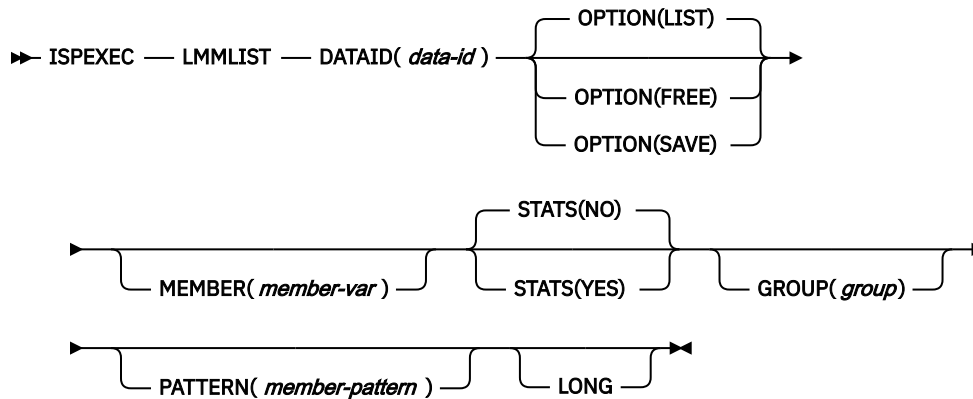
When you invoke LMMLIST for the first time with the LIST option, the MEMBER variable determines the starting position within the member list. To position at the beginning, set the MEMBER variable to blanks. If the requested member is not found, the next member in the member list is returned. The member list is sorted by member name. Repeated invocation 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.

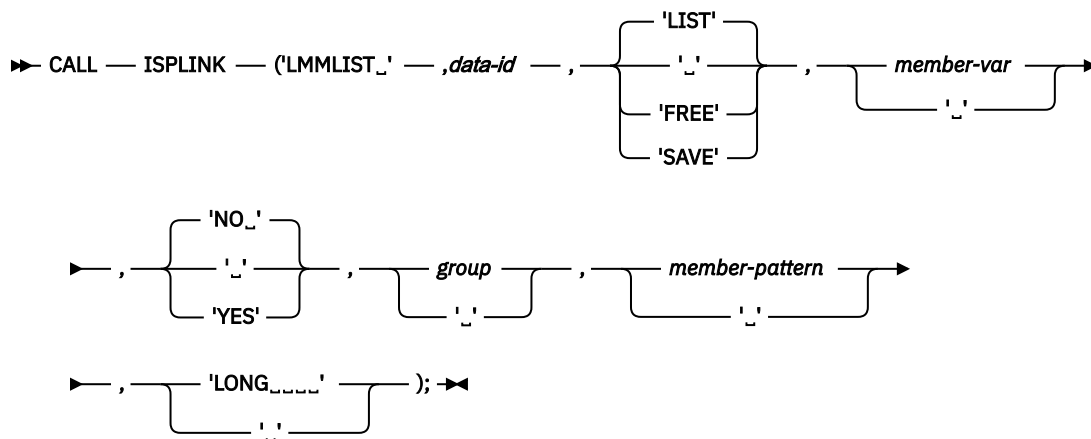
You must complete the LMINIT and LMOPEN services before using LMMLIST. Use the LMMLIST FREE option to release the list storage space when it is not needed.

**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



## Call invocation format



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, 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 (STATS)**

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 154.

**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 topic on naming ISPF libraries and data sets in the [z/OS ISPF User's Guide Vol I](#) for a more complete description of patterns and pattern matching.

**LONG**

When SAVE is selected to save the member list to a data set, 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**

These return codes are possible:

**0**

One of these:



- 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 these:

- 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 these:

- 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.

## Examples

Here are some examples of the LMMLIST service:

### Example 1:

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 name in 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 BUFLLEN to the length of the variable BUFFER. Issue the command:

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

## Example 2:

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 BUFLLEN to the length of the variable BUFFER. Issue the command:

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

## Example 3:

This example is an ISPF edit macro, which invokes the LMMLIST service to loop through all of the members of the data set being edited. This macro starts with the ISREDIT MACRO statement, which takes a parameter of a macro name. The data ID and member name are retrieved using the ISREDIT DATAID and ISREDIT MEMBER statements respectively. LMOPEN opens the data set so that LMMLIST can use it. The member name variable is blanked, so that the LMMLIST service starts at the first member in the data set. LMMLIST is called within a loop, which stops when LMMLIST returns a non-zero return code, either because all members have already been listed, or because an error occurred. If the LMMLIST service returns successfully, the EDIT service is called with the member name and the edit macro which was an input parameter. After the loop completes, the member list is freed with the OPTIONS(FREE) parameter of the LMMLIST service, and the data set is closed with the LMCLOSE service. This example is shipped with ISPF as 'ISP.SISPSAMP(ISRMBRS)'.

```
/*REXX*****
/* ISPF edit macro to process all members of partitioned data set, */
/* running a second, user-specified, ISPF edit macro against each */
/* member. */
/*
/* To run:
/* Enter "ISRMBRS macname" on the command line, where macname is */
/* the macro you want run against each member. */
/******
'ISREDIT MACRO (NESTMAC) '

/******
/* Get dataid for data set and issue LMOPEN */
/******
'ISREDIT (DATA1) = DATAID'
'ISREDIT (CURMEM) = MEMBER'
Address ispxec 'LMOPEN DATAID('data1') OPTION(INPUT)'
member = ' '
```

```

lmrc = 0

/*****
/* Loop through all members in the PDS, issuing the EDIT service for */
/* each. The macro specified on the ISRMBSR invocation is passed as */
/* an initial macro on the EDIT service call. */
*****/
Do While lmrc = 0
  Address ispxexec 'LMMLIST DATAID('data1') OPTION(LIST),
                  MEMBER(MEMBER) STATS(NO)'
  lmrc = rc
  If lmrc = 0 & member /= curmem Then
    do
      Say 'Processing member' member
      Address ispxexec 'EDIT DATAID('data1') MEMBER('member')
                      MACRO('nestmac')'
    end
  End
End

/*****
/* Free the member list and close the dataid for the PDS. */
*****/
Address ispxexec 'LMMLIST DATAID('data1') OPTION(FREE)'
Address ispxexec 'LMCLOSE DATAID('data1')'

Exit 0

```

## 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-length and variable-length data sets can be packed.

Completion of the LMINIT service is required before you invoke LMMOVE. You must specify ENQ(MOD) with the LMINIT service if you want to use LMMOVE to append records to the "to-data-id". See [“LMINIT—generate a data ID for a data set”](#) on page 130 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.

### Note:

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.

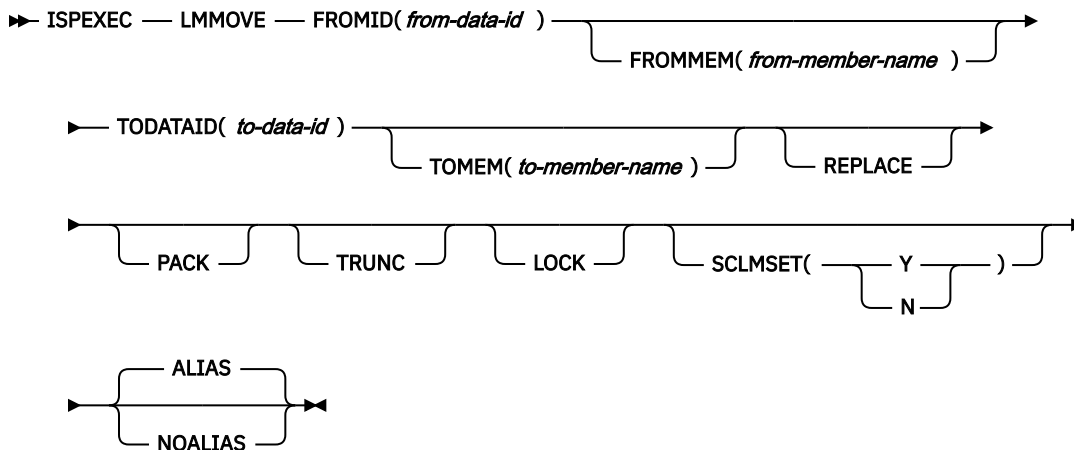
Moving to the same data set is not supported when aliases are automatically selected, as it would result in the "from" and "to" member names being the same.

4. If the NOALIAS option is in effect, LMMOVE does not move alias members unless either:
  - 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.

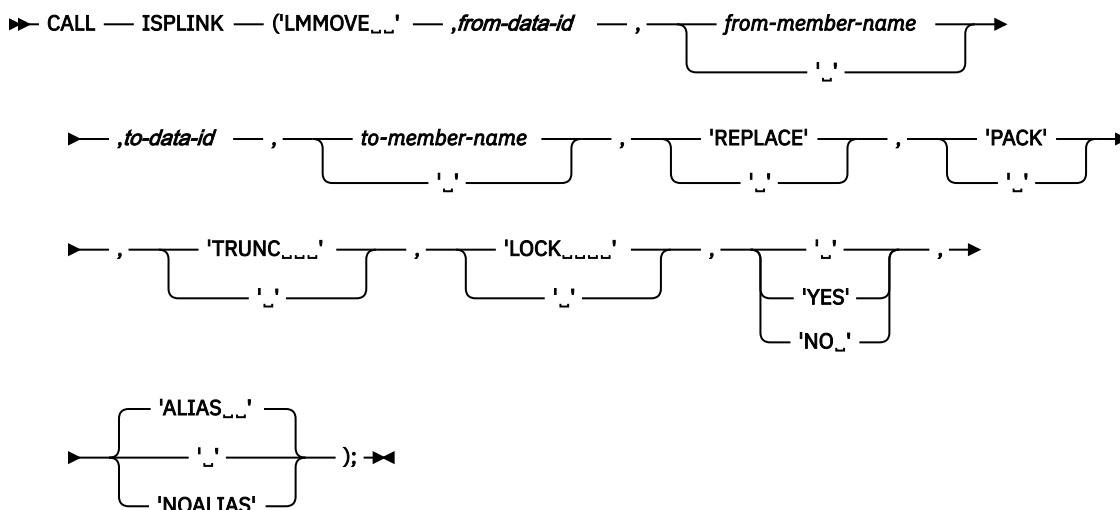
If the NOALIAS option is in effect, moving an alias member by itself will result in a new member being created, even if the main member has already been moved.

5. If *from-data-id* represents an empty sequential data set, LMMOVE performs the copy but sets the return code to 4 as a warning.
6. When the LMMOVE service is used to move a member in a PDSE version 2 data set that is configured for member generations, the current generation of the member is moved and any previous generations of the member are deleted.

## Command invocation format



## Call invocation format



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. If an asterisk (\*) or a pattern is used, a member count is returned in these dialog variables:

#### ZSVCCPY

Number members copied

#### ZSVCNCPY

Number of members not copied

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**

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**

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.

#### **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 LMMOVE service will fail with return code 12. If you want to be able to specify LOCK and have the LMMOVE ignore the value, change the FAIL\_ON\_LMF\_LOCK keyword value in the ISPF Configuration Table to NO.

#### **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**

Either:

- "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 these:

- 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 BUFLLEN to the length of the variable BUFFER. Issue the command:

```
CALL ISPEXEC (BUFLLEN, 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.

**Note:** When the LMMREN service is used to rename a member in a PDSE version 2 data set that is configured for member generations, the current generation of the member is renamed and any previous generations of the member are deleted.

## Command invocation format

```

➤ ISPEXEC — LMMREN — DATAID( data-id ) — MEMBER( old-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.

Where the *data-id* refers to a partitioned data set load library (RECFM=U), and *old-member-name* is the name of an existing primary member, the user data component of any associated alias names will be updated to refer to the renamed primary name.

### **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 ISPF 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. ISPF 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

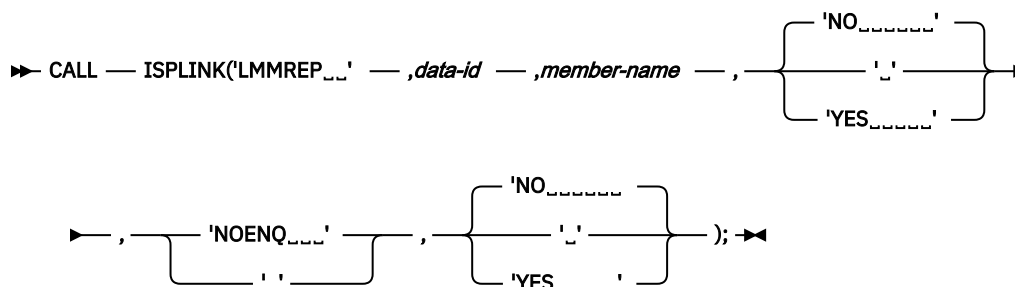
These return codes are possible:

- 0** Normal completion.
- 4** Directory already contains the specified new name.
- 8** Member not found.





## Call invocation format



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.

Where *member-name* is the name of an existing primary member, the primary name and all associated alias names are updated. Where *member-name* is the name of an existing alias member, the alias name is updated to be a primary member and any association with the original primary member name is lost.

### YES|NO (STATS)

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 specify 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 or ZLC4DATE) and the date of most recent change (ZLMDATE or ZLM4DATE) must be provided in the member statistics.

If you specify NO, the default value, the statistics are not updated.

These 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. Use system variable ZDATEF to determine the national format. Either ZLCDATE or ZLC4DATE is required. If both ZLCDATE and ZLC4DATE are entered, ZLCDATE is used.

### ZLMDATE

Last change date, a character value shown in your national format. Use system variable ZDATEF to determine the national format. Either ZLMDATE or ZLM4DATE is required. If both ZLMDATE and ZLM4DATE are entered, ZLMDATE is used.

### 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

- When extended statistics are not enabled in the site configuration, this variable contains a number in the range 0 - 65535.
- When extended statistics are enabled in the site configuration, this variable contains a number in the range 0 - 2147483647.

**ZLINORC**

Initial number of records

- When extended statistics are not enabled in the site configuration, this variable contains a number in the range 0 - 65535.
- When extended statistics are enabled in the site configuration, this variable contains a number in the range 0 - 2147483647.

**ZLMNORC**

Number of changed records

- When extended statistics are not enabled in the site configuration, this variable contains a number in the range 0 - 65535.
- When extended statistics are enabled in the site configuration, this variable contains a number in the range 0 - 2147483647.

**ZLUSER**

User ID of the last user to change the given member; the user ID has a maximum length of 7 characters. To specify an 8-character user ID, you must use the ZLUSER8 variable.

**ZLC4DATE**

Creation date, a character variable shown in your national format with a 4-character year. Use system variable ZDATEF to determine the national format. Either ZLCDATE or ZLC4DATE is required. If both ZLCDATE and ZLC4DATE are entered, ZLCDATE is used.

**ZLM4DATE**

Last modified date, a character variable shown in your national format with a 4-character year. Use system variable ZDATEF to determine the national format. Either ZLMDATE or ZLM4DATE is required. If both ZLMDATE and ZLM4DATE are entered, ZLMDATE is used.

**ZLUSER8**

User ID of the last user to change the given member. When 8-character user IDs are enabled on the system, the user ID has a maximum length of 8 characters. When 8-character user IDs are not enabled on the system, the user ID has a maximum length of 7 characters.

**NOENQ**

An optional parameter that specifies that ISPF 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. By default, ISPF will issue the ENQ unless NOENQ is specified.

**YES|NO (EXT)**

Starting in z/OS V2R3 ISPF, you do not need to specify this parameter. Extended statistics are automatically generated if they are enabled in the site configuration and any of the line count values exceed 65535.

**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**

These 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 these:

- 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 command:

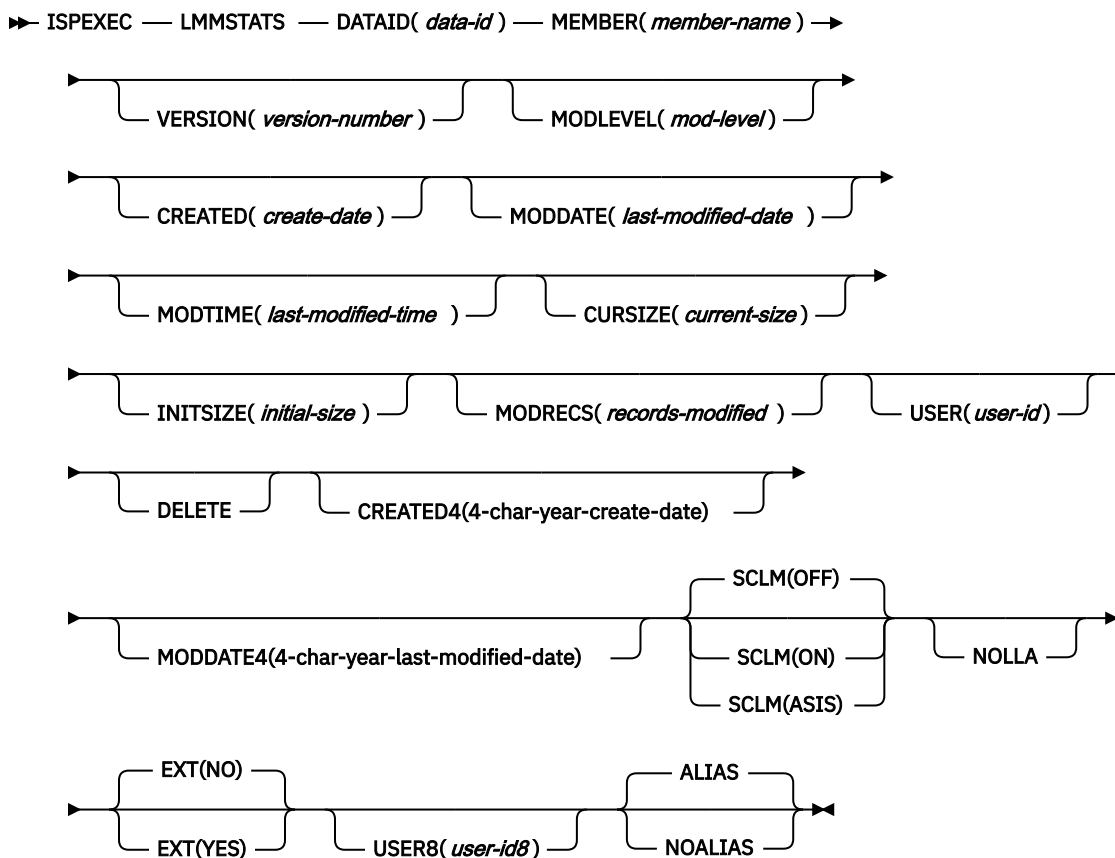
```
CALL ISPEXEC (BUFLen, BUFFER);
```

**LMMSTATS—set and store, or delete ISPF statistics**

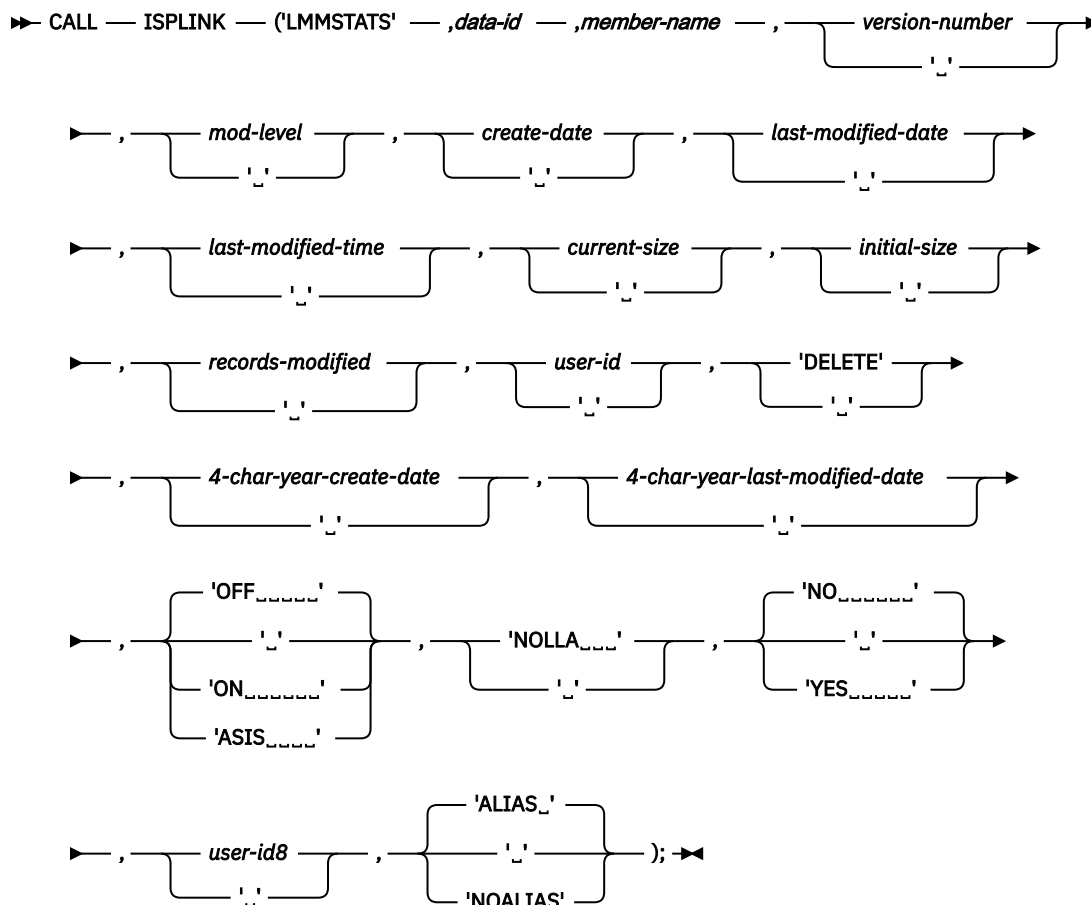
The LMMSTATS service sets and stores, or deletes ISPF statistics for members of a partitioned data set. This service can be used with ISPF libraries or an MVS partitioned data set. Any and all statistics can be

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



## Call invocation format



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 or all members. The maximum length of this parameter is 8 characters. Specify a single asterisk as a member pattern to have the statistics for all members processed.

### 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, or 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.

- When extended statistics are not enabled in the site configuration, this parameter must be a number in the range 0 - 65535.
- When extended statistics are enabled in the site configuration, this parameter must be a number in the range 0 - 2147483647.

**initial-size**

The original number of data records in the member when it was created.

- When extended statistics are not enabled in the site configuration, this parameter must be a number in the range 0 - 65535.
- When extended statistics are enabled in the site configuration, this parameter must be a number in the range 0 - 2147483647.

**records-modified**

The number of data records modified in a member since it was created.

- When extended statistics are not enabled in the site configuration, this parameter must be a number in the range 0 - 65535.
- When extended statistics are enabled in the site configuration, this parameter must be a number in the range 0 - 2147483647.

**user-id**

User ID of the user that last modified the data. The maximum length of this parameter is 7 characters. To specify an 8-character user ID, you must use the user-id8 parameter.

**user-id8**

User ID of the user that last modified the data. When 8-character user IDs are enabled on the system, the maximum length of this parameter is 8 characters. When 8-character user IDs are not enabled on the system, 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.

**NOLLA**

If LLA is used to manage a cached directory entry, specify this keyword to ensure that the cached entry is not used.

**YES|NO (EXT)**

Starting in z/OS V2R3 ISPF, you do not need to specify this parameter. Extended statistics are automatically generated if they are enabled in the site configuration and any of the line count values exceed 65535.

**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.

**ALIAS|NOALIAS**

With ALIAS in effect, an alias member of a PDS might be selected for storing or deletion of ISPF statistics. Note that if an alias member is specified, then performing the LMMSTATS operation can result in a non-ALIAS member being created which initially points to the same TTR as the original member. The statistics reset on an alias of a member of a PDSE is not supported.

If the NOALIAS option is in effect, aliases members will not be allowed for processing. In addition, if the NOALIAS option is in effect and a member pattern is specified, then ALIAS members which match the pattern will be skipped.

## Return codes

These return codes are possible:

**0**

Normal completion.

**4**

Either:

- 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 these:

- Invalid parameter value.
- Data set is not partitioned.
- Data ID represents a concatenation of data sets.
- Data set is opened for output.
- Data set name is an alias. And the NOALIAS parameter was specified.

**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 command:

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

## LMOPEN—open a data set

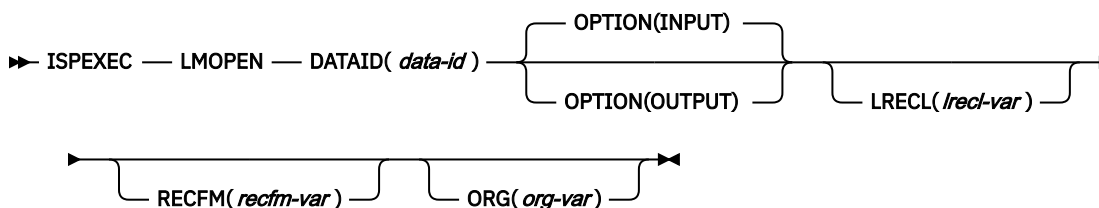
The LMOPEN service opens the data set associated with a given data ID so the data set can be either read from, using LMGET, or written to, using LMPUT. The LMINIT service must be completed before LMOPEN can be used.

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). See 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 LMOOPEN 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







**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 these:

- 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 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.



## LMPUT

**4**

Either:

- 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**

Either:

- 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 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 this format:

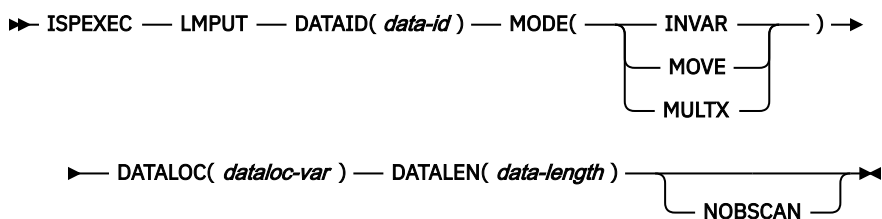
```

+-----+-----+-----+-----+-----+-----+
|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 ---->

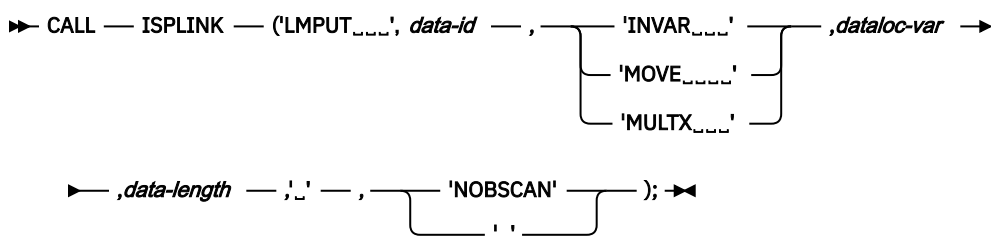
```

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



## Call invocation format



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

In INVAR mode and MULTX mode, the data-location parameter variable contains the data itself. In MOVE mode, 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 or MULTX 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|MULTX specification. 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**

These 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**

Either:

- 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 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 by setting them to '. For example, if x is the name of a variable: x = ' '

## 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 32 000 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')'

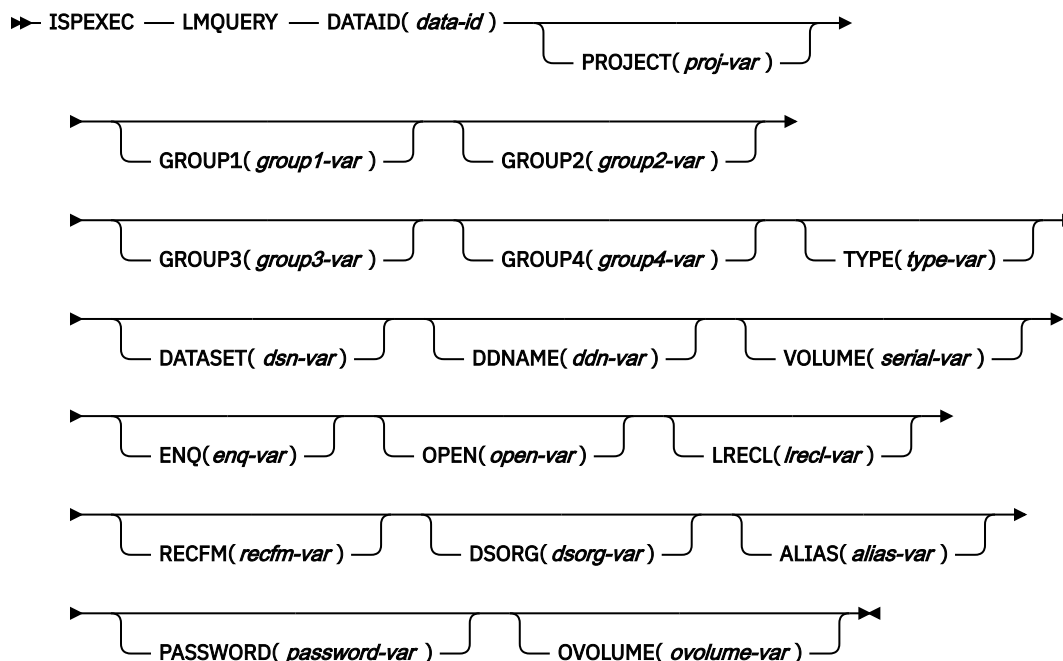
```

## LMQUERY—give a dialog information about a data set

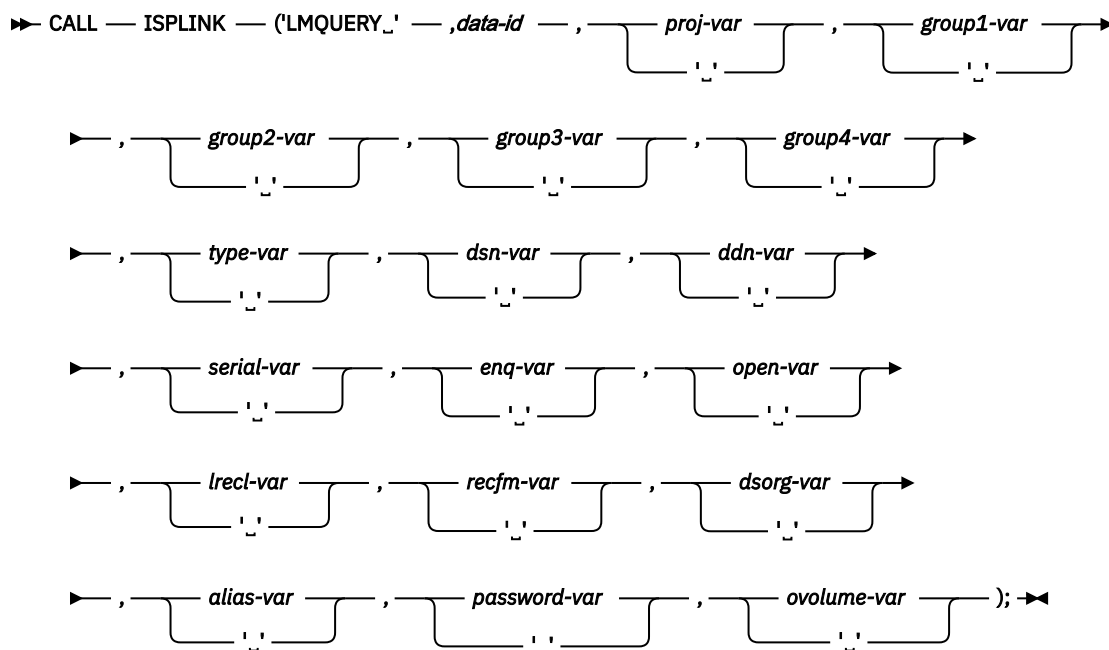
The LMQUERY service returns values specified for the LMINIT service parameters that are associated with a given data ID. In this way, LMQUERY provides the dialog with selected information about a data set by showing how the LMINIT parameters were set up when the data ID of that data set was generated.

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.

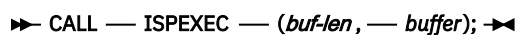
## Command invocation format



## Call invocation format



or



## 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 46-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 set has been allocated will be placed. If a DDNAME was specified on the LMINIT service, it will be returned. If no DDNAME was specified, the DDNAME generated by ISPF will be returned.

**volume-var**

The name of a 6-character variable into which the value of the VOLUME parameter specified on the LMINIT service will be placed.

**enq-var**

The name of a 5-character variable into which the value of the ENQ parameter specified on the LMINIT service will be placed.

**open-var**

The name of an 8-character variable into which an indicator will be placed to indicate whether the data set was opened for INPUT, OUTPUT, or UPDATE. If no LMOPEN has been done, blanks will be returned.

**lrecl-var**

The name of an 8-character variable into which the character representation of the logical record length will be placed. If no LMOPEN has been done, blanks will be returned.

**recfm-var**

The name of a 4-character variable into which the record format will be placed. If no LMOPEN has been done, blanks will be returned. These characters may appear in the record format value:

**F**

Fixed-length records

**V**

Variable-length records

**U**

Undefined-length records

**B**

Blocked records

**T**

Track overflow

## LMQUERY

### **M**

Machine control characters

### **A**

ANSI control characters

### **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

These 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 these 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.



## LMRENAME

### 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

These return codes are possible:

**0**

Normal completion.

**4**

New name already exists.

**8**

One of these:

- 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 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

These 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 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 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.



**action**

A single letter action to replace the "S" line command, such as E (edit) or B (browse). If not specified, then the system attempts to invoke an external command called "S".

**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**

These 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 these:

- 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**

This 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.

Here is the command invocation:

```
ISPEXEC MEMLIST DATAID(&ID) CONFIRM(YES)
```

Here is the call invocation:

```
CALL ISPLINK ('MEMLIST ',ID,'YES ');
```

Alternatively, you could:

1. Set the program variable BUFFER to contain

```
BUFFER='MEMLIST DATAID(&ID) CONFIRM(YES) ';
```

2. Set program variable BUFLen to the length of the variable BUFFER.
3. Issue the command

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





## 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

These 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 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, ISPMMLIB, ISPSLIB, ISPTLIB, ISPLLILB, ISPTABL, ISPFIL, 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

These 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**.

Here is the command invocation

```
ISPEXEC QBASELIB ISPMLIB ID(IDV)
```

Here is the call invocation:

```
CALL ISPLINK ('QBASELIB','ISPMLIB ','IDV ');
```

Or alternatively, you could:

1. Set the program variable BUFFER to contain

```
QBASELIB ISPMLIB ID(IDV)
```

2. Set program variable BUFLen to the length of the variable BUFFER.
3. Issue the command

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

## 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, ISPFIL, or a generic name. The values that may be specified on a LIBDEF service may be specified on a QLIBDEF service. The image library associated with ddname ISPILIB is no longer used by ISPF.

### 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

These 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 command:

```
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 constructed by appending a sequence number to the prefix. The total number of variables created is returned in a variable constructed by appending '0' (zero) to the prefix.

### Return codes

These return codes are possible:

**0**

Normal completion.

**4**

List incomplete. There was insufficient space to construct 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.

**Note:** QUERYENQ may not return all enqueue data when there are many requestors with the same QNAME/RNAME combination. Only 84 requestors can be returned under these circumstances.

## Command invocation format

```

➤ ISPEXEC — QUERYENQ — TABLE(table-name) — QNAME(qname) — RNAME(rname) →
      └──────────────────┬──────────────────┘
      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 with TBEND.

### qname

A variable name that can contain a name or a prefix. A prefix must end in an asterisk. The default is '\*' (all qnames). Maximum length is 8 characters and must be fully padded if called from a compiled program because embedded blanks are allowed.

### rname

A variable name that can contain a name or a prefix. A prefix must end in an asterisk. The default is '\*' (all rnames). Its length is 255 characters and must be fully padded or VDEFINED to a shorter length if called from a compiled program because embedded blanks are allowed.

### pattern

Used to limit the ENQ search to specific requestors. The pattern can contain asterisks which will match zero or more characters, and percent signs which will match one character. The value of *pattern* is the actual pattern, and not a variable name.

If the variable value is not a prefix (does not end in an asterisk before any trailing blanks), it must be the exact length of the RNAME being requested. For compiled programs, this can be controlled on the VDEFINE or VREPLACE statement. The exceptions to this rule are for QNAMEs SPFEDIT and SPFUSER. For SPFUSER requests, the variable name is padded or truncated to 8 characters. For SPFEDIT requests, variables less than 45 characters in length are padded with blanks to 44 and treated as a prefix. Variables longer than 44 characters are padded to 52 and not treated as a prefix. Variables that are passed in as a prefix are not changed.

### WAIT

Indicates that all waiting ENQs are returned. This shows all ENQ contention known to the local system. RNAME and QNAME are ignored for WAIT.

### limit

The maximum size of the table. The default is 5000. Zero (0) indicates no limit.

### list-id

An 8-character data set name qualifier, used to create a data set named [prefix.userid].list-id.ENQLIST according to standard ISPF naming conventions. The data set is a VB 332 data set, containing the same data as would be returned in the table. The order is: Owner, System, Disposition, Hold, Scope,

Global, QNAME, and RNAME. RNAME is last because trailing blanks are removed to reduce the size of the data set. A space is added between each field.

### XSYS

Indicates that the XSYS=YES parameter should be used on the GQSCAN macro. The default is to use XSYS=NO. This means that some ENQs on other systems may not be returned. Use of the XSYS keyword may have significant performance implications. See the documentation for the GQSCAN macro in *z/OS MVS Programming: Authorized Assembler Services Reference EDT-IXG* for more information.

### 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.

## Variables returned in each row of the table

Table 9. Variables Returned in Each Row of the Table		
Name	Size	Description
ZENJOB	8	Job or address space name holding or requesting the ENQ
ZENQNAME	8	Qname portion of the ENQ
ZENRNAME	255	Rname portion of the ENQ
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

These 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); ➤
```

or

```
➤ 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

These 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.

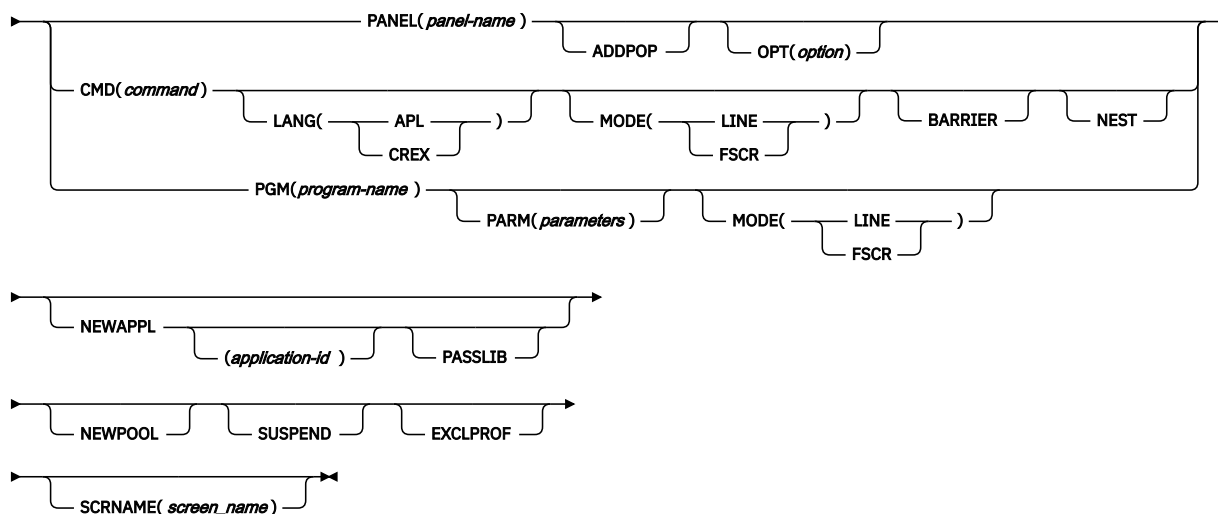
Within a dialog function, a program can invoke another program using standard CALL or LINK conventions. These nested programs are transparent to the dialog manager. However, when the invoked program is a new dialog function, SELECT must be used.

## SELECT

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 →

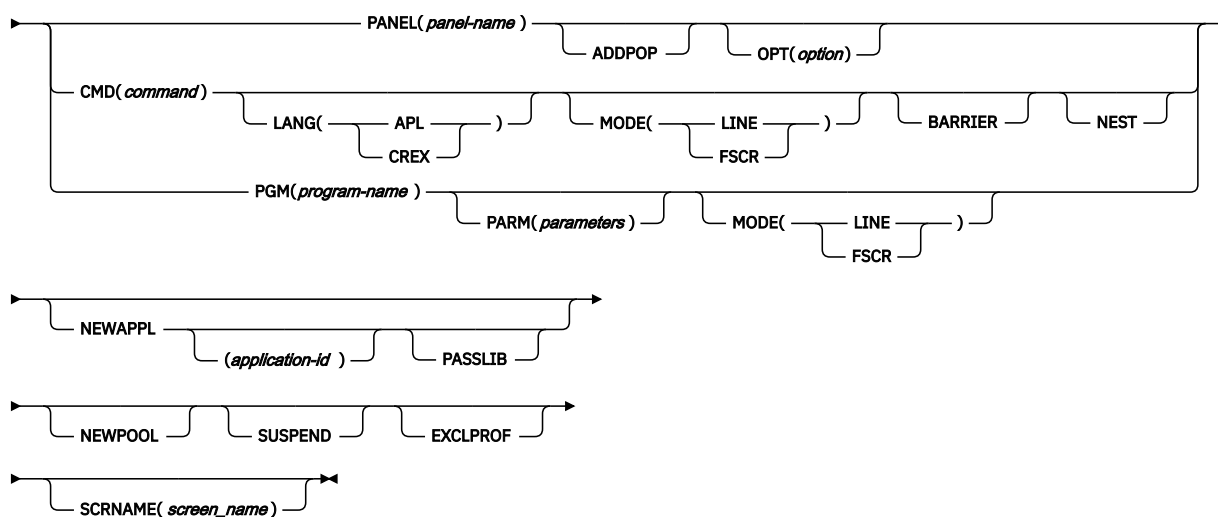


## Call invocation format

➡ CALL — ISPEXEC — (*buf-len*, — *buffer*); →

or

➡ CALL — ISPLINK — ('SELECT\_...' — , — *length*, →



## 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.

### ADDDPOP

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).
- Ensure 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.

If using quotes(') as part of the parameter list to the SELECT command, you must ensure that each quote is matched. Quotes used in SELECT service command parameters and the TSO EXEC primary command must have an even number of quotes, whereas this is not needed when executing a command from the TSO Ready prompt.

### 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 [z/OS 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.

## SELECT

### 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 "ISPxxxx" and "ISRxxxx".

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 [z/OS 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 runtime option. See the publications supporting the language for specific requirements.

### 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.

This example shows 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 this 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.

## SELECT

### 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.

### EXCLPROF

Specifies that ISPF is to disable the multi-logon profile sharing support for this service call. The parameter is ignored if the ISPF multi-logon support is not enabled, either by means of the ISPF Configuration options, or by specifying SHRPROF on the ISPSTART command. The parameter is optional.

### 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 example shown, 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.

### Note:

1. 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.
2. The CONTROL ERRORS mode set in the dialog function that issued the SELECT service call does not apply to return codes being passed from the command or program, but it does apply to return codes set by the SELECT service.
3. 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.

4. 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

These 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.

## Examples

See:

- [“Example 1” on page 207](#)
- [“Example 2” on page 207](#)
- [“Example 3” on page 207](#)
- [“Example 4” on page 208](#)
- [“Example 5” on page 208](#)
- [“Example 6” on page 208](#)

### 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 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 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.

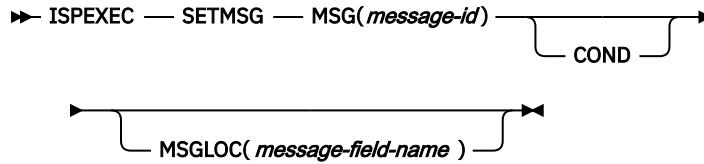
A message that has been set with SETMSG is displayed the next time any full-screen output is sent to the display, regardless of whether that output is a panel, table display, Browse data, or Edit data. The SETMSG service executed in the batch environment causes the message to be written to the log at 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.



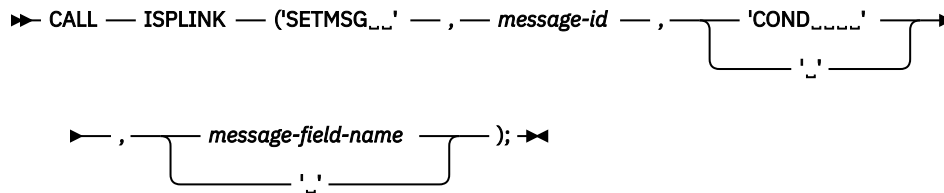
## Command invocation format



## Call invocation format

➤ CALL — ISPEXEC — (*buf-len* , — *buffer*); ➤

or



## 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.

### 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

These 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. Enter the command:

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

or alternately

```
CALL ISPLINK ('SETMSG ', 'ABCX015 ');
```

## Example 2

This 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 this message definition for TSTA110

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

Results in:

```
PANEL A

FIELD==> FLD1
  ┌──────────┐
  │ ENTER NUMERIC DATA │
  └──────────┘
  <<<
```

## 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.

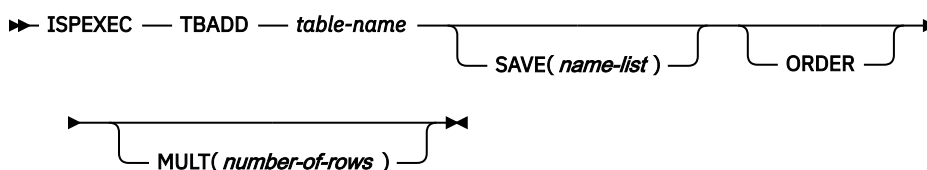
To improve performance when you add several rows to a table, you can specify the MULT keyword with the *number-of-rows* parameter. By specifying the estimated number of rows you expect to add to the table, you supply ISPF the information it needs to more efficiently obtain the necessary storage for all 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



## 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**

These 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. Enter the command:

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

or alternately

```
CALL ISPLINK ('TBADD      ','TELBOOK ',' ','ORDER      ');
```

## Example 2

Add multiple rows to table TELBOOK.

ISPEXEC TBADD TELBOOK MULT(&amp;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);
```

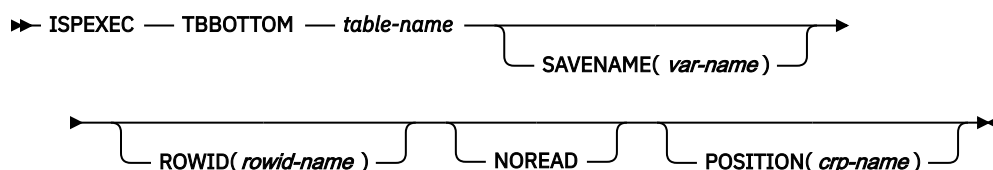
where 8 indicates the number of rows to be added.

## 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



## 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. The variable must be an 8-byte character field.

**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

These 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 command:

```
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.

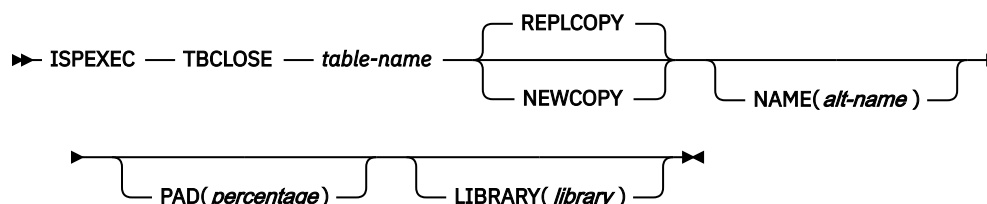
Table output can be directed to a table output library other than the default library specified on the table output ISPTABL DD statement. The library to be used must be allocated before table services receives control. Thus, an application can update a specific table library. This is particularly useful for applications that need to maintain a common set of tables containing their data.

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.

When TBCLOSE is used with the LIBRARY parameter, ISPF does not immediately close the data set allocated to the LIBRARY ddname or LIBDEF DATASET. The data set remains open until an ISPF table service is used with a different LIBRARY. If the LIBRARY is a LIBDEF lib-type, it will be closed when the LIBDEF is cleared. If the LIBRARY is a ddname, it might be closed by using a table service with any other LIBRARY name, even with a dummy table and LIBDEF name, such as TBSTATS DUMMYT LIBRARY(DUMMYL).

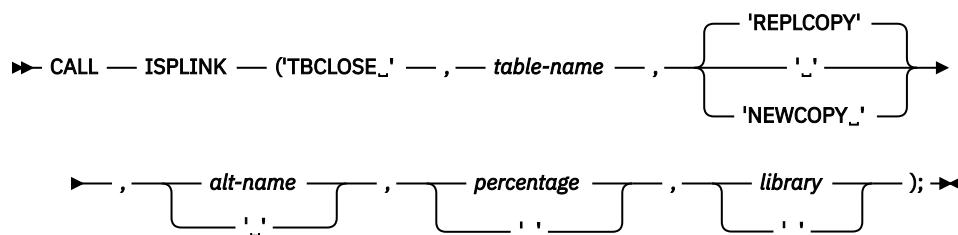
### Command invocation format



### Call invocation format

>> CALL — ISPEXEC — (buf-len, — buffer); <<

or



## 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) ddname 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

These 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 command:

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

or alternately

```
CALL ISPLINK ('TBCLOSE ', 'TELBOOK ');
```

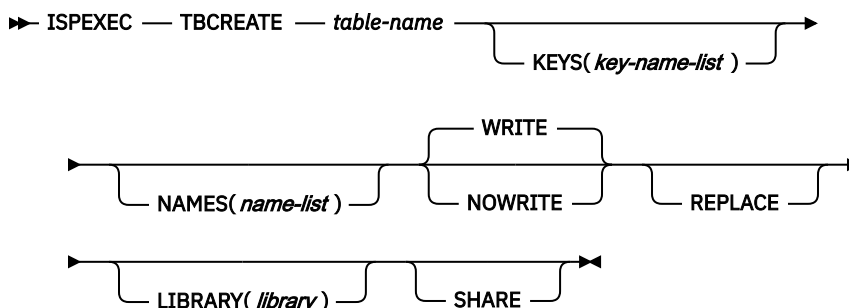
## TBCREATE—create a new table

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.

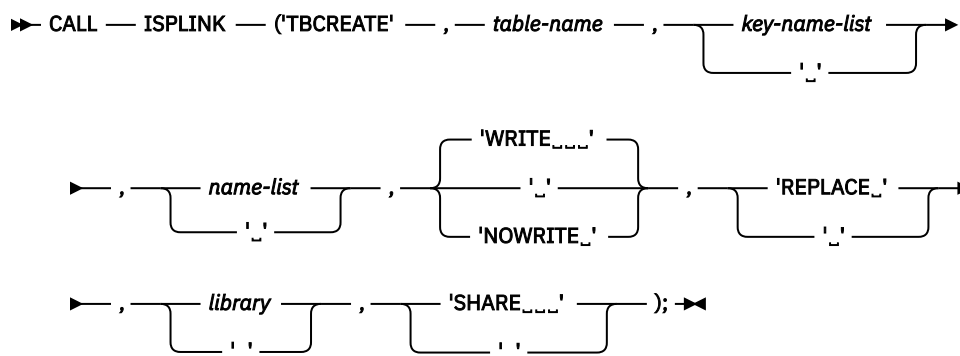
### Command invocation format



### Call invocation format

```
>> CALL — ISPEXEC — (buf-len, — buffer); <<
```

or



## 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](#) 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) ddname 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

These 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.

## Examples

See:

- [“Example 1” on page 219](#)
- [“Example 2” on page 219](#)
- [“Example 3” on page 220](#)

### 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 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

These 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 command:

```
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 these 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**

These 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:

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) and the variable values are unpredictable. 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 ZTDELS contains the number of rows that were selected. The value of ZTDELS 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 ZDTOP. 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 ZDTOP before reinvoking the TBDISPL service with the panel name specified. This is not necessary if the panel name is not specified.

## ZDTOP and ZTDELS variables

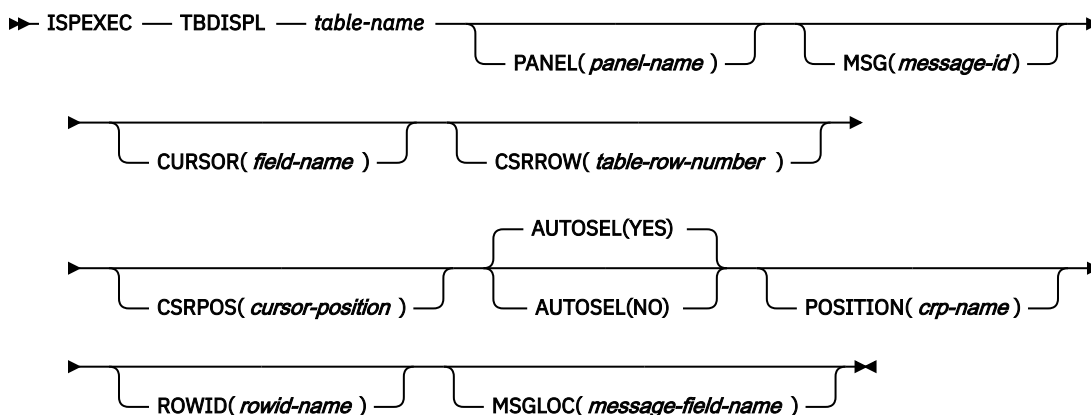
ZDTOP and ZTDELS are variables in the function pool. A command procedure can access them directly. A program can access them through use of the VDEFINE or VCOPY service. If a program function uses the VCOPY service to access the variable, the value will be in character string format. It will not be in fixed binary format.

If the application user selected more than one row in a single interaction, the variable ZTDELS 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 previously, 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.

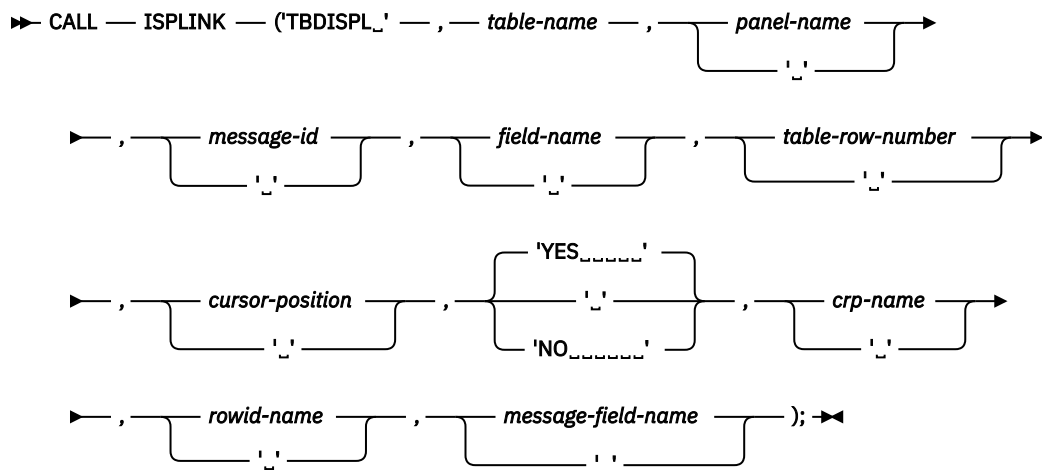
## Command invocation format



## Call invocation format

➤ CALL — ISPEXEC — (*buf-len*, — *buffer*); ➤

or



## 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. The variable must be an 8-byte character field.

**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.

**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. They are processed 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: 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

These 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 these 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 these:
  - 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 these 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 TBDISPL 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 command:

```
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, this assignment could be made in the )INIT section of a table display panel:

```
&ZTDMARK = '----> End of Data <----'
```

ZTDMARK can be blank. That is, this assignment is valid:

```
&ZTDMARK = ' '
```

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 *z/OS 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.

### ZTDVROWS

Returns the number of visible rows available from the most recent table display. Only complete model sets are counted, so if a model set spans multiple lines and not all of the lines are visible, it is not counted. ZTDVROWS is set to zero if no complete model sets are visible. ZTDVROWS resides in the function pool. It is a six character output pool variable and will have leading zeros if necessary.

System variables ZTDRET, ZTDADD, ZTDSCR, 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 these 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 242](#).

### 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” on page 229](#).

### 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 the 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

These 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 these 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 previous three statements shown could be on one line. For example:

```
If (.msg=' ') &Select=' ' Refresh(Select)
```

## 4. Using Auto-Selection

Consider this 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. This 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.
- The variable is retrieved from the function pool only once for each TBDISPL with a nonblank panel name. This means that the same model line, including attribute settings, will be in effect for all rows displayed by the TBDISPL.
- 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.
- If the SFIHDR keyword is specified on the )MODEL header statement, the first variable model line is assumed to define scroll indicator fields for scrollable fields that are defined on subsequent variable model lines.

## Example—panel using variable model lines

Figure 7 on page 233 is the panel definition for a panel named VARMOD. Figure 8 on page 234 and Figure 9 on page 234 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 7. 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      Lee, Kelly
''   KC10002      Smith, Rich
''   KC10003      Holmes, David
''   KC10004      Jones, Bob
''   KC10005      Xu, Bill
''   KC10006      Miller, Chris
''   KC10007      Stephens, Matthew
''   KC10007      Morris, Theresa
***** BOTTOM OF DATA *****
```

Figure 8. 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      Lee, Kelly
                        253 Main St
                        Junction City      KS
=====
''   KC10002      Smith, Rich
                        2810 Curtis Lane
                        Long Beach        CA
=====
''   KC10003      Holmes, David
                        3600 Chestnut St
                        Hyannis           MA
=====
''   KC10004      Jones, Bob
                        212 Fallon Ave
                        North Hudson      NY
=====
''   KC10005      Xu, Bill
                        180 Berthold Ave
                        Baton Rouge       LA
```

Figure 9. (Part 1 of 2). Variable Model Lines: Display 1

```

=====
''  KC10006      Miller, Chris
                        South Mountain Pass                Ashland                NH
=====
''  KC10007      Stephens, Matthew
                        42 Dragonica Way                    Newark                DE
=====
''  KC10008      Morris, Theresa
                        67 Waimea Blvd                      Naalehu                HI
=====
***** BOTTOM OF DATA *****

```

Figure 10. (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.

### Example—scroll indicator field in first variable model line

Figure 11 on page 236 shows the panel definition for panel SFIMOD, which is similar to the previous example but uses variable model lines and the SFIHDR keyword on the )MODEL statement to define scrollable fields and scroll field indicators in the model section.

```

)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(//)
%--/-- Publication List --/--
%Command ==>_cmdfld / / +Scroll ==>_amt +
+
+ Allow Update? ==>_QUP+(Yes or No)
+ Display format ==>_Z+ 1. Name|Doc. Number|Title
+ 2. Doc. Number|Name|Title
+ 3. Title|Name|Doc. Number
+
%Select
%Code &HDG
)Model sfihdr
&MDSI
&MDL1
)Init
&amt=page
.ZVARS='(QFM)'
If (&QUP=' ') &QUP=NO
If (&QFM=' ') &QFM=1
If (&QUP='YES') &TYPE='Input'
If (&QUP='NO') &TYPE='Output'
If (&QFM=' ') &TYPE='Output'
ver(&QFM Range,1,3)
If (&QFM=1)
&HDG='Name Doc. Number Title'
If (&QFM=2)
&HDG='Doc. Number Name Title'
If (&QFM=3)
&HDG='Title Name Doc. Number'
)Proc
&QUP = Trans(Trunc(&QUP,1) Y,YES N,NO ' ',NO *,*)
Ver(&QUP,List,YES,NO)
ver(&QFM Range,1,3)
If (&QFM=1)
&HDG='Name Doc. Number Title'
&MDSI=' $Ttlsind +'
&MDL1='!SCODE+øName +$Docnum +$Title +'
If (&QFM=2)
&HDG='Doc. Number Name Title'
&MDSI=' $Ttlsind +'
&MDL1='!SCODE+$Docnum +øName +$Title +'
If (&QFM=3)
&HDG='Title Name Doc. Number'
&MDSI=' $Ttlsind +
&MDL1='!SCODE+$Title +øName +$Docnum +'
)Field
FIELD(TITLE) SIND(TTLSIND)
)End

```

Figure 11. SFIHDR Keyword in Variable Model Lines: Panel Definition

With the SFIHDR keyword specified on the )MODEL statement, the variable &MDSI is assumed to define scroll indicator fields for scrollable fields defined in the variable &MDL1.

Before panel SFIMOD is displayed, the dialog function must initialize the variable model lines as follows:

```

&MDSI=' $Ttlsind +'
&MDL1='!SCODE+øName +$Docnum +$Title +'

```

The Title field in the model section is defined as a scrollable field with a separator indicator displayed in panel variable &TTLSIND (refer to the field section of the panel).

With this panel, the user can alter the order in which the fields in the model are displayed using the "Display format" field (QFM). For example, if the user enters 3 in this field the &MDL1 variable is modified so that the fields are displayed in the order Title, Name and Doc. Number, and the &MDSI variable is modified so that the separator indicator field is displayed above the Title field.

Figure 12 on page 237 shows the panel display when &QFM equals 1.

```

----- Publication List ----- Row 1 to 11 of 11
Command ==>                      Scroll ==> PAGE

Allow Update? ==> NO (Yes or No)
Display format ==> 1  1. Name|Doc. Number|Title
                     2. Doc. Number|Name|Title
                     3. Title|Name|Doc. Number

Select
Code   Name      Doc. Number  Title
----->
ISPZDG20 SC34-4821-02 z/OS V1R5.0 ISPF Dialog Developer's
ISPZDT20 SC34-4824-02 z/OS V1R5.0 ISPF Dialog Tag Languag
ISPZEM20 SC34-4820-02 z/OS V1R5.0 ISPF Edit and Edit Macr
ISPZMC20 SC34-4815-02 z/OS V1R5.0 ISPF Messages and Codes
ISPZPC20 GC34-4814-02 z/OS V1R5.0 ISPF Planning and Custo
ISPZRS20 SC34-4816-02 z/OS V1R5.0 ISPF Reference Summary
ISPZSC20 SC34-4817-02 z/OS V1R5.0 ISPF Software Configura
ISPZSG20 SC34-4819-02 z/OS V1R5.0 ISPF Services Guide
ISPZSR20 SC34-4818-02 z/OS V1R5.0 ISPF Software Configura
ISPZUG20 SC34-4822-02 z/OS V1R5.0 ISPF User's Guide Volum
ISPZU220 SC34-4823-02 z/OS V1R5.0 ISPF User's Guide Volum
***** Bottom of data *****

```

Figure 12. SFIHDR Keyword in Variable Model Lines: Panel Example 1

Figure 13 on page 237 shows the panel display when &QFM equals 3.

```

----- Publication List ----- Row 1 to 11 of 11
Command ==>                      Scroll ==> PAGE

Allow Update? ==> NO (Yes or No)
Display format ==> 3  1. Name|Doc. Number|Title
                     2. Doc. Number|Name|Title
                     3. Title|Name|Doc. Number

Select
Code   Title                                     Name      Doc. Number
----->
z/OS V1R5.0 ISPF Dialog Developer's  ISPZDG20  SC34-4821-02
z/OS V1R5.0 ISPF Dialog Tag Languag  ISPZDT20  SC34-4824-02
z/OS V1R5.0 ISPF Edit and Edit Macr   ISPZEM20  SC34-4820-02
z/OS V1R5.0 ISPF Messages and Codes   ISPZMC20  SC34-4815-02
z/OS V1R5.0 ISPF Planning and Custo   ISPZPC20  GC34-4814-02
z/OS V1R5.0 ISPF Reference Summary    ISPZRS20  SC34-4816-02
z/OS V1R5.0 ISPF Services Guide       ISPZSG20  SC34-4819-02
z/OS V1R5.0 ISPF Software Configura   ISPZSC20  SC34-4817-02
z/OS V1R5.0 ISPF Software Configura   ISPZSR20  SC34-4818-02
z/OS V1R5.0 ISPF User's Guide Volum   ISPZUG20  SC34-4822-02
z/OS V1R5.0 ISPF User's Guide Volum   ISPZU220  SC34-4823-02
***** Bottom of data *****

```

Figure 13. SFIHDR Keyword in Variable Model Lines: Panel Example 2

## Example—using the TBDISPL and TBPOT services

This topic describes the use of the TBDISPL and TBPOT 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 TBPOT 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 TBPOT 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. Here are the contents of table TAB1:

EMP SER	LNAME	FNAME	I	PHA	PHNUM
-----	-----	-----	--	----	-----
598304	Robert	Richard	P	301	555-1224
172397	Smith	Susan	A	301	555-8465
813058	Lowe	Charles	L	202	555-9557
395733	Adams	John	Q	202	555-1776
502774	Hsu	Ann	A	914	555-4156

Figure 14. 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 15 on page 239. Format of the display is controlled by a panel definition named PAN1, shown in Figure 16 on page 240. 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
Robert    Richard    P      301   555-1224      598304
Smith     Susan     A      301   555-8465      172397
Lowe      Charles   L      202   555-9557      813058
Adams     John      Q      202   555-1776      395733
Hsu       Ann       A      914   555-4156      503774
***** BOTTOM OF DATA *****

```

Figure 15. 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      _EMP SER
)Init
  &AMT = PAGE
)Proc
  VPUT (Notes) Profile
)End

```

Figure 16. Table Display Panel Definition PAN1

Control will be returned to the dialog function when the user performs one of these actions:

- 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 these 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 16 on page 240](#), 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 performs one of these actions:

- 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. These 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 16 on page 240](#), 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 TBPUR TAB1

Process the scrollable portion input. Here, the current selected row is processed. In this example, the TBPUR 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 to 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:

<i>Service</i>	<i>Required Parameter</i>	<i>Optional Parameters</i>	<i>Categories</i>
TBDISPL	table-name	[PANEL(panel-name)] [MSG(message-id)] [CURSOR(field-name)] [CSRROW(table-row-number)] [CSRPOS(cursor-position)] [AUTOSEL(YES NO)] [POSITION(cip-name)] [ROWID(rowid-name)]	in name in name in name in name in number in number in key out number 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. These 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. These 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. These items can appear in the )MODEL section of a table display panel definition:

- Text
- Variable model lines
- Input fields
- Output fields.

12. These items cannot appear in the )MODEL section of a table display panel definition:

- Variables within text
- Dynamic areas
- Graphic areas.

13. During TBDISPL display, these user actions return control to the dialog function:

- Pressing the Enter key. See item 6 in [“TBDISPL summary” on page 242](#) 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, these 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 these 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. These 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 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.

When TBEND is issued for a table opened with the LIBRARY parameter, ISPF does not immediately close the data set allocated to the LIBRARY ddname or LIBDEF DATASET. The data set remains open until an ISPF table service is used with a different LIBRARY. If the LIBRARY is a LIBDEF lib-type, it will be closed when the LIBDEF is cleared. If the LIBRARY is a ddname, it might be closed by using a table service with any other LIBRARY name, even with a dummy table and LIBDEF name, such as TBSTATS DUMMYT LIBRARY(DUMMYL).

## 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

These 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 this command:

```
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

These 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. Enter the command:

```
CALL ISPEXEC (BUFLN, 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

These 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 command:

```
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.

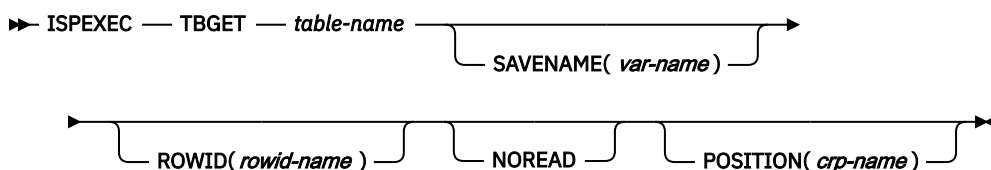
For tables with keys, the table is searched for the row to be fetched. The current contents of the key variables, dialog variables that correspond to keys in the table, are used as the search argument.

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.

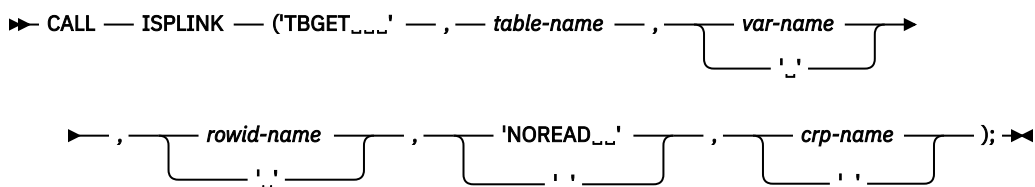
### Command invocation format



### Call invocation format

```
>> CALL — ISPEXEC — ( buf-len , — buffer ); <<
```

or



## 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. The variable must be an 8-byte character field.

### **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

These 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 BUFLN to the length of the variable BUFFER. Issue the command:

```
CALL ISPEXEC (BUFLN, 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. This 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**

These 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 command:

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

or alternately

```
CALL ISPLINK ('TBMOD ', 'TELBOOK ');
```

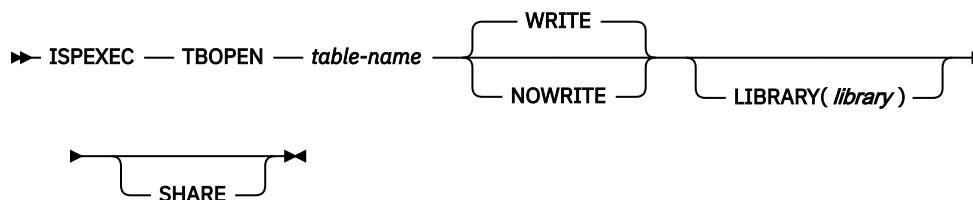
**TBOPEN—open a table**

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.

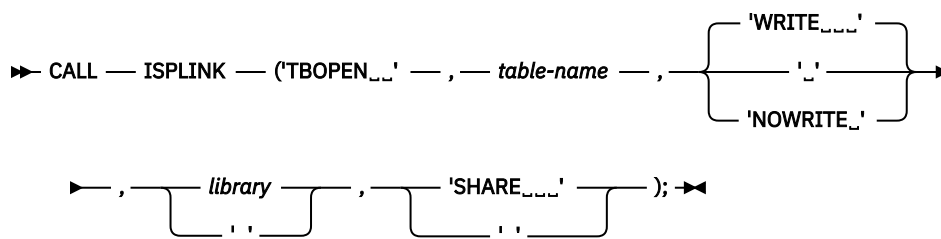
## Command invocation format



## Call invocation format

>> CALL — ISPEXEC — (*buf-len*, — *buffer*); <<

or



## 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) ddname 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**

These 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 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.

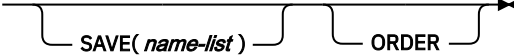
For tables with keys, the current contents of the key variables, dialog variables that correspond to keys in the table, must match the key of the current row, pointed to by the current row pointer (CRP). Otherwise, the update is not performed.

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.

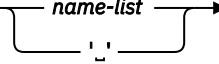
## Command invocation format

➤ ISPEXEC — TBPOT — *table-name* —  ➤

## Call invocation format

➤ CALL — ISPEXEC — (*buf-len* , — *buffer*); ➤

or

➤ CALL — ISPLINK — ('TBPOT\_...' — , — *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

These 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 TBPOT TELBOOK
```

Set the program variable BUFFER to contain:

```
TBPOT TELBOOK
```

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

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

or alternately

```
CALL ISPLINK ('TBPOT', 'TELBOOK');
```

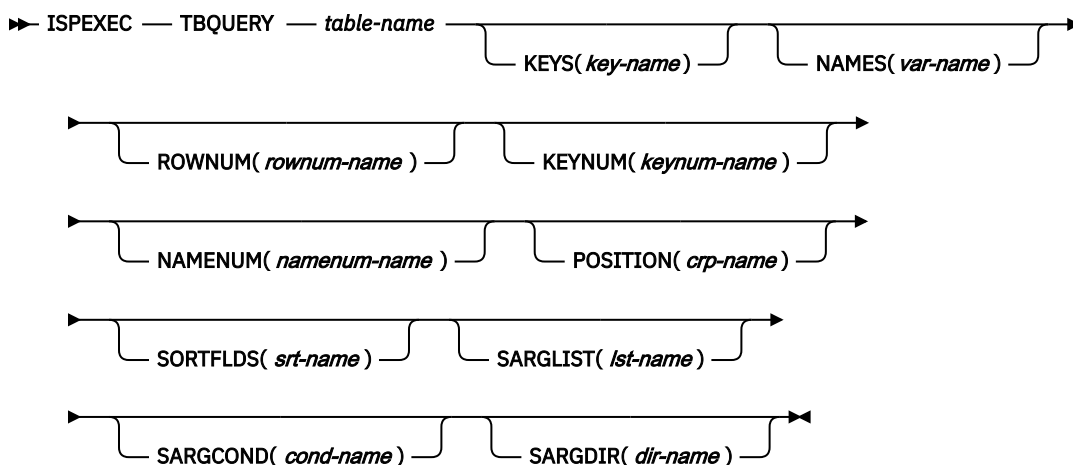
## TBQUERY—obtain table information

The TBQUERY service returns information about a specified table, which must have been opened (TBPOT) by the current user before invoking this service. This 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



### 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**

These 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 command:

```
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 TDBISPL services. When TBSARG is used in conjunction with TDBISPL, the panel definition referred to by the TDBISPL 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 TDBISPL 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 TDBISPL 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, this 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]*
```

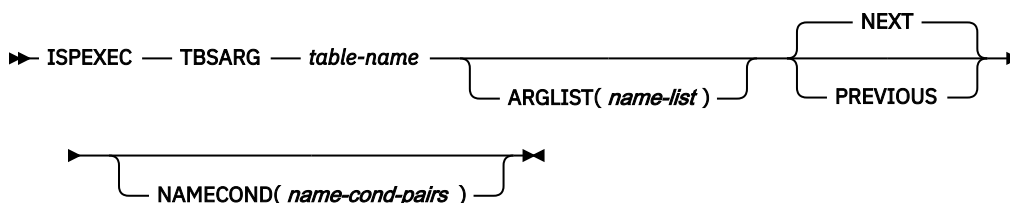
where eeee represents a 4-character EBCDIC string, DBDBDBDBDB represents a 5-character DBCS string, [ and ] represent shift-out and shift-in characters, and \* is an asterisk in single-byte format.

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.

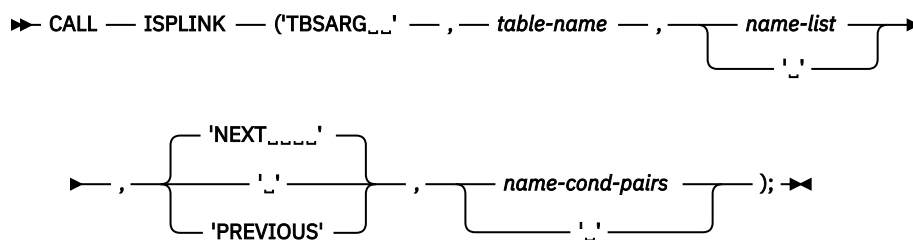
## Command invocation format



## Call invocation format

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

or



## 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 be immediately followed by a date indicator. The date indicator is Yn, 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. These 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

These 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 this 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 command:

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

or alternately

```
CALL ISPLINK ('TBSARG ', 'TELBOOK ', ' ', 'NEXT ',  
              '(LNAME,EQ,ZIPCODE,GT)');
```

## TBSAVE

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 this TBSARG request:

```
ISPEXEC TBSARG DATETBL NEXT NAMECOND(DATE1,GTY1)
```

## TBSAVE—save a table

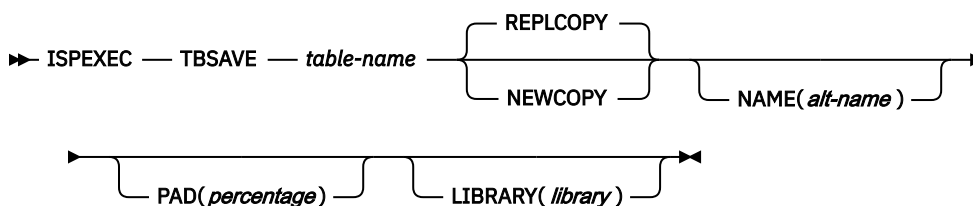
The TBSAVE service writes the specified table from virtual storage to the table output library. The table output library must be allocated to a ddname of ISPTABL, or specified by using the LIBDEF service before invoking this service. The table must be open in WRITE mode.

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.

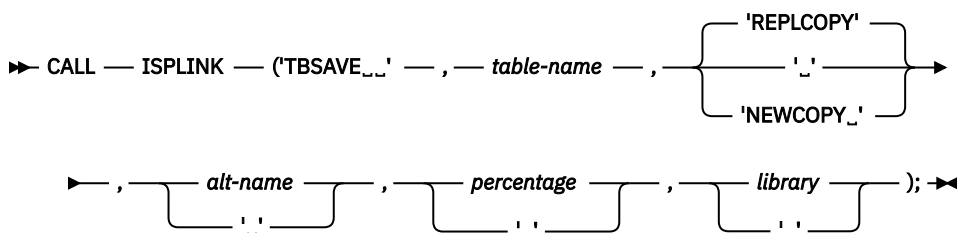
## Command invocation format



## Call invocation format

➤ CALL — ISPEXEC — (*buf-len*, — *buffer*); ➤

or



## 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 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 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) ddname 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

These 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 command:

```
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. A forward search starts with the row after the current row and continues to the end of the table. A backward search 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 up to the "\*" are compared. This is called a generic search argument. A generic search argument is specified by placing an asterisk in the last 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, this 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

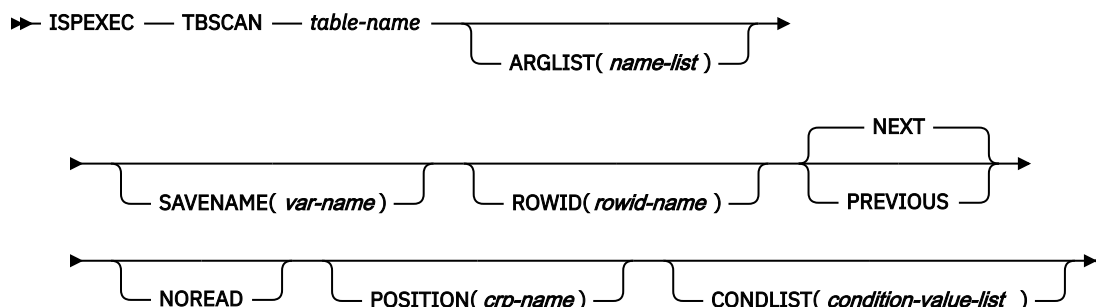
```
eeee[DBDBDBDBDB]*
```



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.

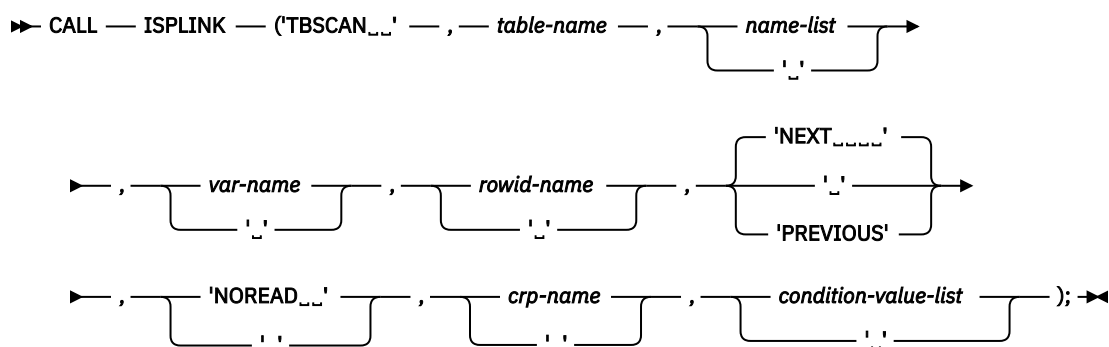
## Command invocation format



## Call invocation format

➤ CALL — ISPEXEC — (buf-len, — buffer); ➤

or



## 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. The variable must be an 8-byte character field.

**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 *Yn*, 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.

These 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**

These 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.

**Examples**

See:

- [“Example 1” on page 267](#)
- [“Example 2” on page 267](#)
- [“Example 3” on page 268](#)

**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. 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 BUFLN to the length of the variable BUFFER. Issue the command:

```
CALL ISPEXEC (BUFLN, 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 this 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 command:

```
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 these requests:

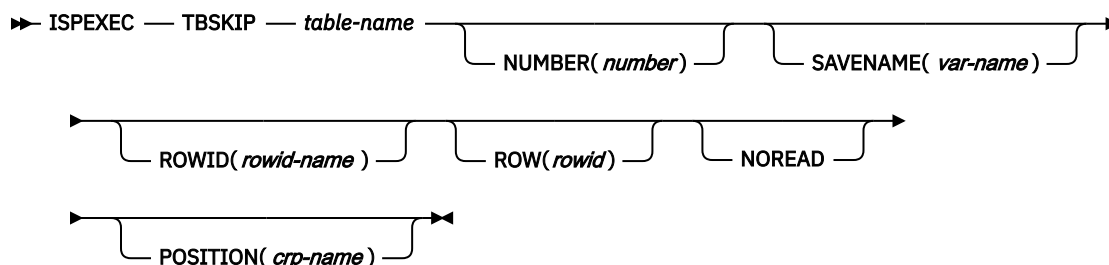
```
ISPEXEC TBSARG DATETBL NEXT NAMECOND(DATE1,GT1)
ISPEXEC TBSCAN DATETBL
```

## 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



### Call invocation format

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

or



**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 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 ISPF configuration table field YEAR\_2000\_SLIDING\_RULE 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.

### 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.

## 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 'Y*n*' 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 'Y*n*' 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. To sort a table into a valid ascending or descending date sequence, the date field must have a format with the most significant part (the year) at the start, the least significant part at the end, and a sort field type designator of Y1. Some examples of valid formats are:

```
YY/MM/DD
YYDDD
YY-MM
```

Internally, TBSORT expands the year to a 4-byte year using the ISPF configuration table field YEAR\_2000\_SLIDING\_RULE to calculate 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 languages, the comma is used in place of a decimal point. To accommodate different usages, three numeric representations are supported: period, comma, and French representations.

Table 10. 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.

These 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 2 147 483 647.
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

These return codes are possible:

**0**

Normal completion.

**12**

Table is not open.

**16**

Numeric convert error.

**20**

Severe error.

## 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 command:



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

or alternately

```
CALL ISPLINK ('TBSORT ', 'TELBOOK ', 'LASTNAME');
```

## Example 2

Perform a sort on table MODSIZES. Sort on NAME, a character field, in ascending sequence. Then sort on SIZE, 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 BUFLN to the length of the variable BUFFER. Issue the command:

```
CALL ISPEXEC (BUFLN, 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.

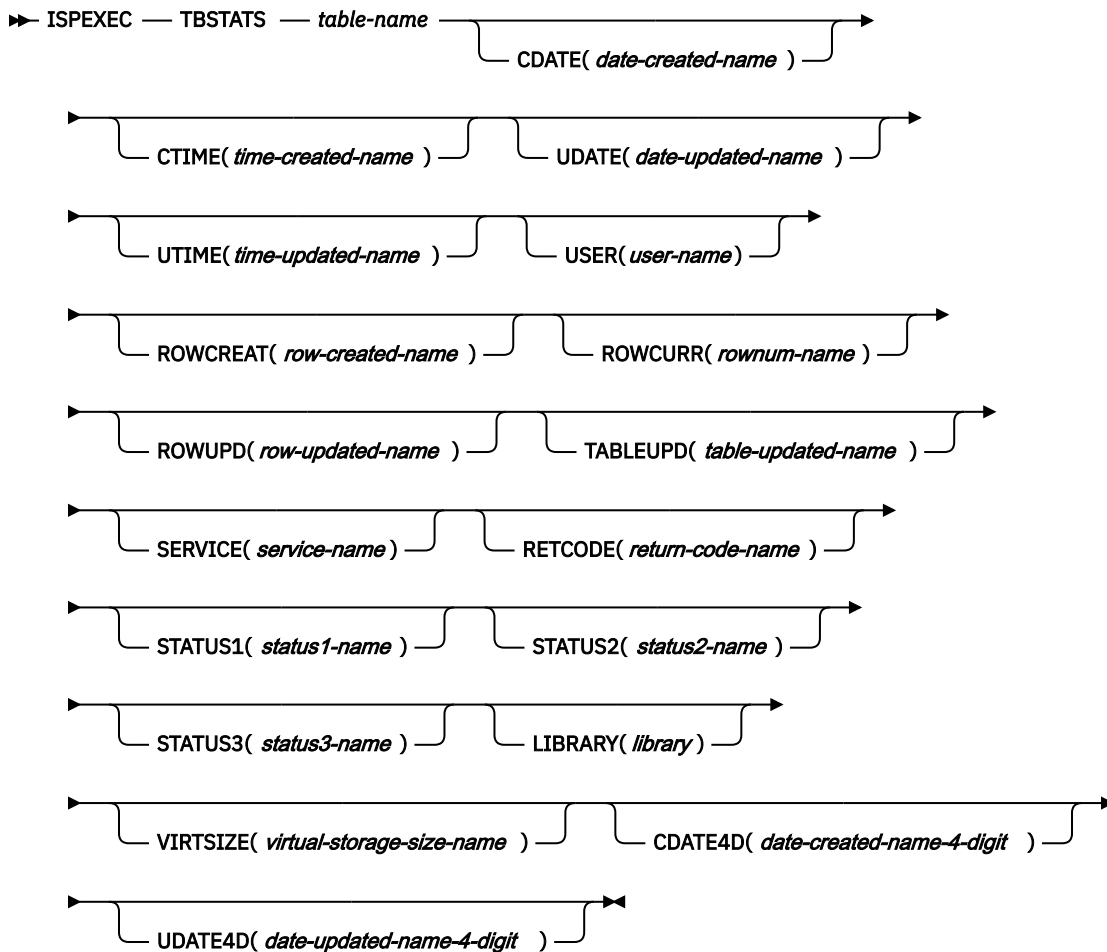
The existence of a table can be checked by the value in the STATUS1 field. If the table does not exist, no other processing takes place.

This statistical information is available:

- Date and time the table was created
- Date and 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
- Number of bytes of virtual storage required by the table

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



## 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 — , — virtual-storage-size-name —
                                ' ' —
➤ , — 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 TBPOT or TBMOT. 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.

**virtual-storage-size-name**

Specifies the name of a variable where the number of bytes of virtual storage required by the table is to be stored.

**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

These 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 BUFLN to the length of the variable BUFFER. Issue the command:

```
CALL ISPEXEC (BUFLN, 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); ➤
```

## TBVCLEAR

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

These 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 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

These 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 command:

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

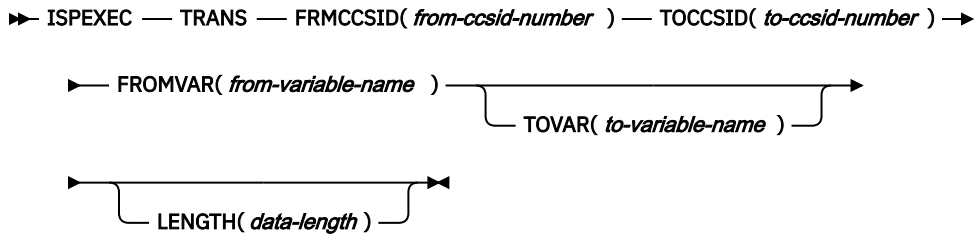
or alternately

```
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 [z/OS ISPF Dialog Developer's Guide and Reference](#).

## Command invocation format



## 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.

### 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

- 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 [z/OS 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 309 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

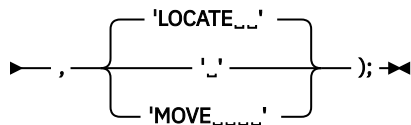
```
ISPEXEC *This service does not apply to APL2 or command
        procedures*
```

## Call invocation format

```
CALL ISPEXEC *This service cannot be used with this interface*
```

or

➔ CALL — ISPLINK — ('VCOPY\_...' — , — *name-list* , — *length-array* , — *value-array* ➔



## 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

These 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, QROWSDATA, '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, 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, 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.

## **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.

**Note:** If the variable being defined is smaller than the retrieved value, the service request terminates and returns to the caller. In this case, if you have specified the LIST option, the remainder of the list is not processed.

## **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

These return codes are possible:

- 0**  
Normal completion.
- 8**  
Variable not found.
- 16**  
Data truncation occurred.
- 20**  
Severe error.

## Examples

See:

- [“Example 1: Error message variable” on page 287](#)
- [“Example 2: Different data formats” on page 287](#)
- [“Example 3: Variables in a structure” on page 288](#)
- [“Example 4: Character data variables” on page 288](#)

### Example 1: Error message variable

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: Different data formats

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

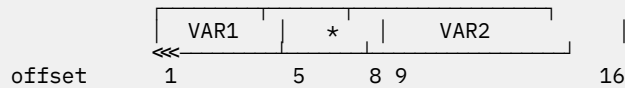
1  VARS,
   3  FVAR      FIXED BIN(31),
   3  CVAR      CHAR(5),
   3  DVAR      CHAR(20),
   FARR(3) CHAR(8),
   LARR(3) FIXED BIN(31);

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: Variables in a structure

Define two dialog variables, VAR1 and VAR2, contained in a structure. The structure is named STRCVARS. It 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: Character data variables

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 BINST, 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 this 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, 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

These 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,SPFDLEN * 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) *
```

The VDELETE service removes variable names, previously defined by the program module, from the function pool. This service is the opposite of VDEFINE.

```
ISPEXEC  *This service does not apply to APL2 or
          command procedures*
```

**➤ 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

These return codes are possible:

**0**

Normal completion.

**8**

At least one variable not found.

**20**

Severe error.

## Example

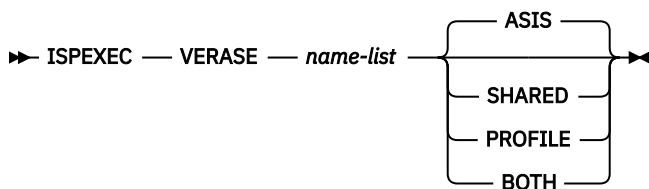
Remove ISPF accessibility to a PL/I program variable that was previously established by VDEFINE to be accessible using dialog variable name MSGNAME.

```
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.

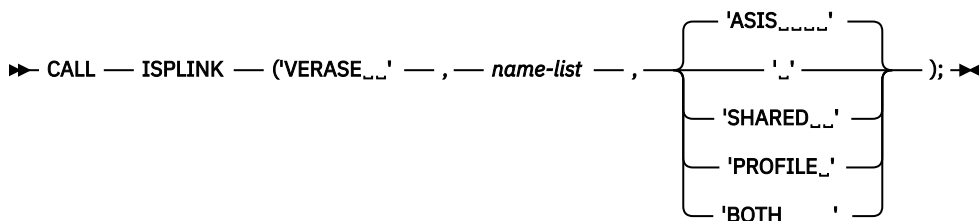
## Command invocation format



## Call invocation format

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

or



## 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

These return codes are possible:

**0**

Normal completion.

**8**

At least one variable not found.

**20**

Severe error.

### **Note:**

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 BUFLN to the length of variable BUFFER and issue the command:

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

or alternately

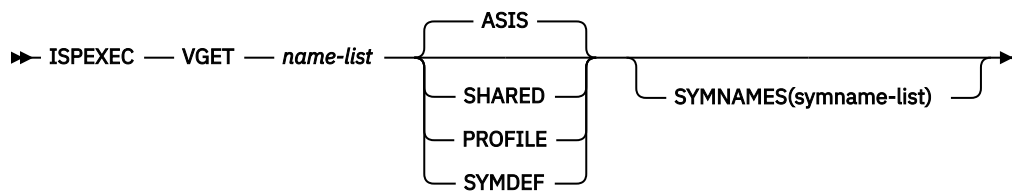
```
CALL ISPLINK ('VERASE ' , '(NAME PHONE ADDRESS)' , 'BOTH ' );
```

## VGET—retrieve variables from a pool or profile or system symbol

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.

The VGET service can also copy values from system symbols. By default each function variable is created with the same name as the corresponding system symbol, but you can specify a different name.

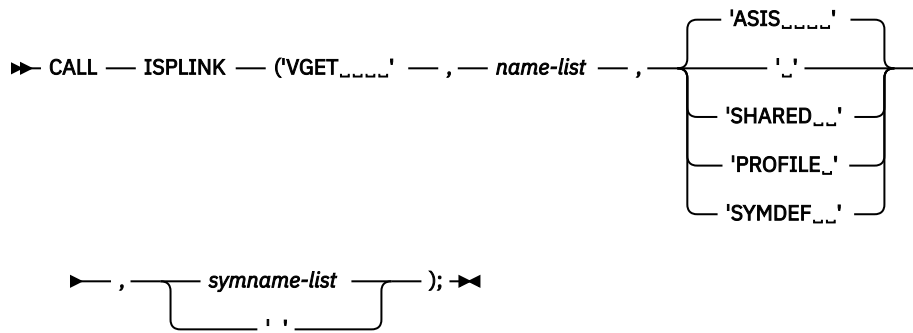
### Command invocation format



### Call invocation format

```
►► CALL — ISPEXEC — (buf-len , — buffer); ◄◄
```

or



### 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.

#### SYMDEF

The values for the variables defined by *name-list* are to be obtained from the system symbols.

**SYMNAMES(symname-list)**

*symname-list* lists the names of one or more system symbols that are to be obtained. It is specified in the same format as the *name-list* parameter. Where *symname-list* is omitted, the system symbols obtained are the same as those specified on the *name-list* parameter.

One reason why you might use the SYMNAMES parameter is that some system symbols may have the same name as a reserved or read-only dialog variable. In this case you must specify a different variable name in *name-list* and specify the actual symbol name in *symname-list*. For example, SYSCONE is a read-only dialog variable in a CLIST. Therefore, this command would work within a REXX exec, but it would fail in a CLIST:

```
VGET (SYSCONE) SYMDEF
```

Instead, you could specify the command to obtain the current value for the static symbol SYSCONE and store it in a variable named CLONE:

```
ISPEXEC VGET (CLONE) SYMDEF SYMNAMES(SYSCONE)
```

If there are fewer symbol names in *symname-list* than names in the *name-list*, then the symbol names are used from the *symname-list* until there are no more corresponding symbol names, then the remaining names in the *name-list* are used. In other words, if there are five names in *name-list* and only three symbol names, the symbol names are used for the first three symbols and the last two names in the *name-list* are used for the remaining symbols.

If the number of symbol names in *symname-list* exceeds the number of names in *name-list*, a severe error occurs.

This is an optional parameter. It is only valid when the SYMDEF parameter is also specified.

**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**

These return codes are possible:

- 0** Normal completion.
- 8** Variable not found. If the SYMDEF parameter was specified: system symbol not found.
- 12** Validation failed.
- 16** Translation error or truncation occurred during data movement.
- 20** Severe error. If the SYMDEF parameter was specified: the number of symbol names in *symname-list* exceeds the number of names in *name-list*.

**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).

## Examples

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 command:

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

or alternately

```
CALL ISPLINK ('VGET ' ,VARLIST,'SHARED ');
```

In a CLIST, obtain the current value for the dynamic system variable LHHMMSS:

```
ISPEXEC VGET (LHHMMSS) SYMDEF
```

In a REXX exec, obtain the current values for the static symbols SYSNAME and SYSR1:

```
'VGET (SYSNAME SYSR1) SYMDEF'
```

In a REXX exec, obtain the current values for the dynamic symbols HHMMSS and LHHMMSS. Also obtain the current value for the static symbol SYSClONE and store it in a variable named cl:

```
'VGET (c1 hhmmss lhhmmss) SYMDEF SYMNames(sysclone)'
```

## 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 these 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 partitioned data sets, sequential data sets, z/OS UNIX files, and data sets 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.

**Note:**

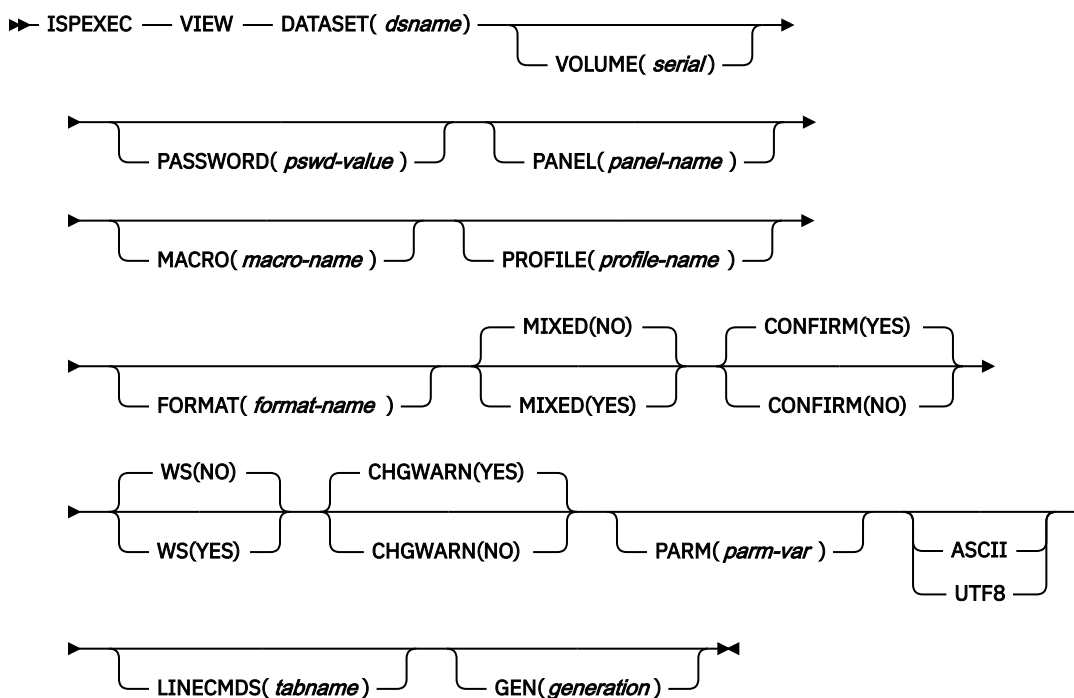


1. Dialogs that invoke the VIEW service may invoke the EDREC service first to start view recovery, because the VIEW service does not do view recovery.
2. The VIEW service might alter the DISPLAY environment. Do not expect the DISPLAY environment that existed before invoking the VIEW service to remain 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.

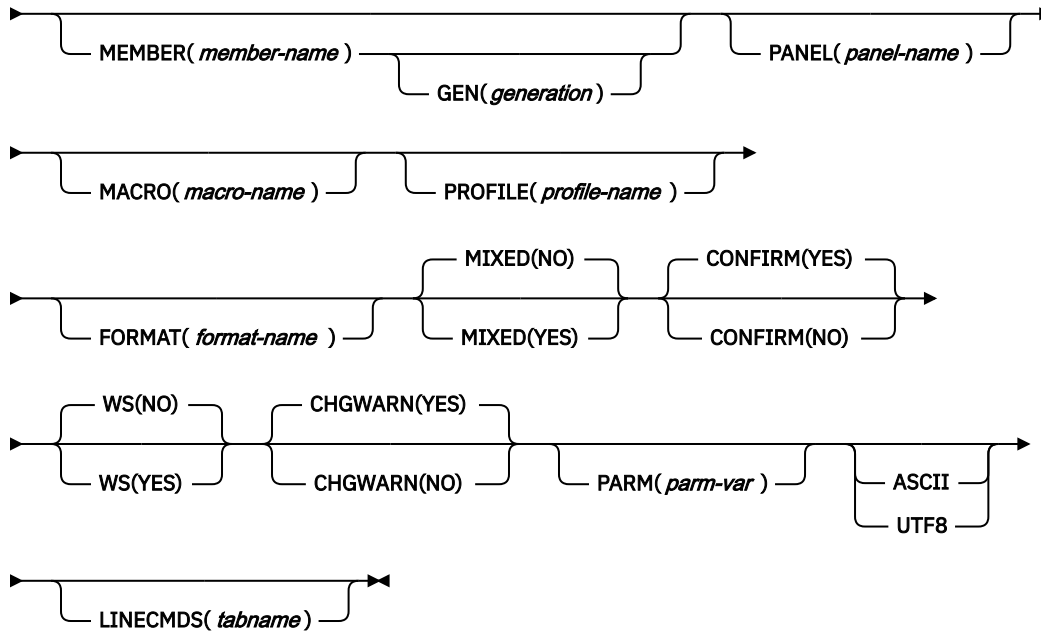
A dialog using VIEW can place data into the ZEDUSER dialog variable in the shared pool. The data in ZEDUSER is saved in the edit recovery table as an extension variable when the recovery data set is initialized. This is done if RECOVERY is ON when first entering view or after using the CREATE or REPLACE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

## Command invocation format



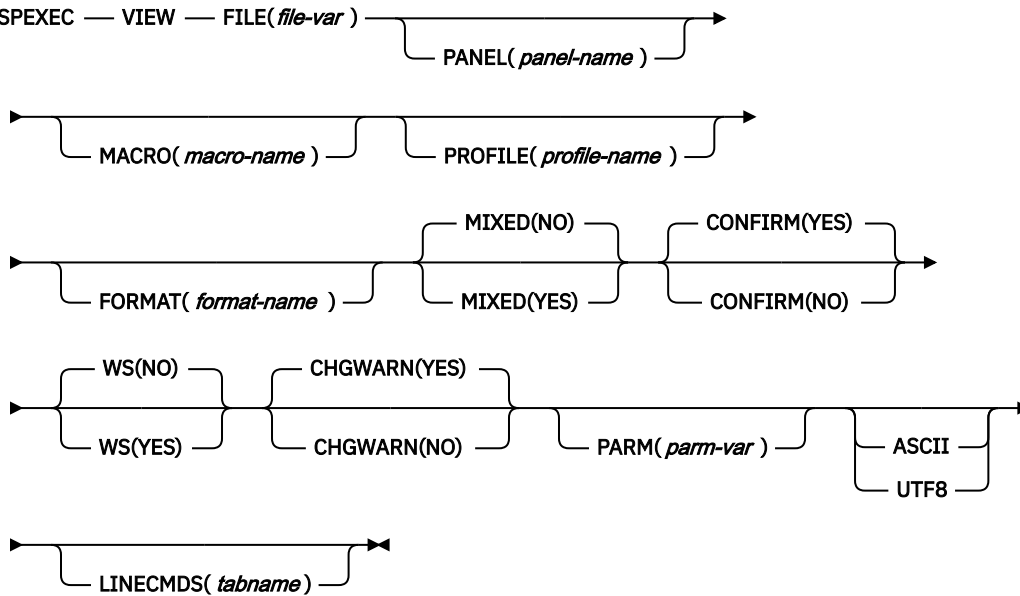
or

►► ISPEXEC — VIEW — DATAID( *data-id* ) —►

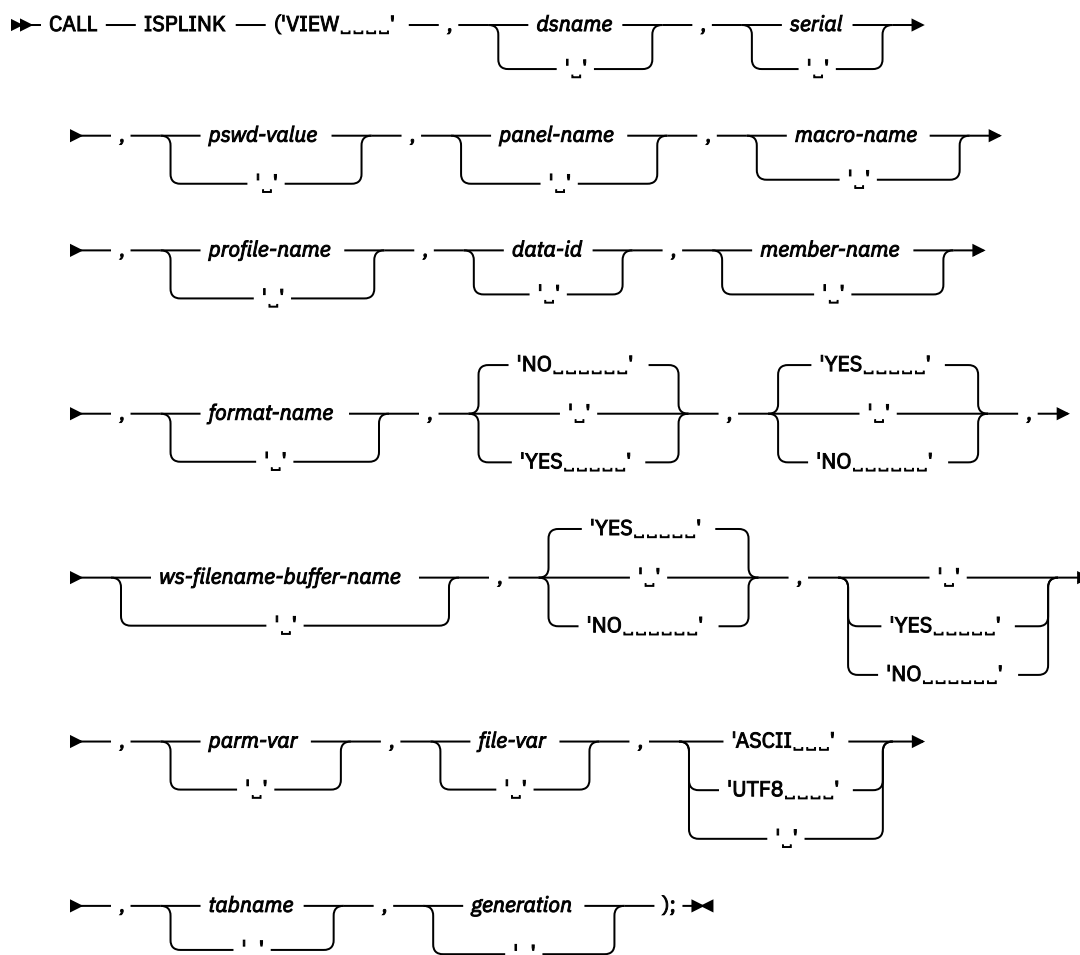


or

►► ISPEXEC — VIEW — FILE( *file-var* ) —►



## Call invocation format



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 *z/OS ISPF User's Guide Vol 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-protected data sets.

**panel-name**

The name of a customized view panel, created by you, to be used when displaying the data. See [z/OS 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. See [z/OS 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 [z/OS 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 (MIXED)**

For the MIXED parameter, if YES is specified, the VIEW service treats the data as mixed-mode DBCS data. If NO 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 (CONFIRM)**

For the CONFIRM parameter, if you specify YES 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 canceled, 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 ENTER. 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 (WS)**

The WS parameter is no longer used, but it is left in for compatibility purposes. If specified, the view service will ignore it.

**YES|NO (CHGWARN)**

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 [z/OS ISPF User's Guide Vol I](#).

**generation**

A fullword fixed integer containing the relative or absolute generation of the member to be viewed. If the value is negative, it is a relative generation. If the value is positive, it is an absolute generation that the caller has determined to be valid. The value 0 (zero) indicates the current generation and is equivalent to not specifying the parameter. This parameter is valid only when the specified member is in a PDSE version 2 data set that is configured for member generations.

**ws-filename-buffer-name**

The *ws-filename-buffer-name* parameter is no longer accepted, but it is left in for compatibility purposes. If specified, the view service will fail with return code 12.

**ASCII|UTF8**

This parameter can be specified when invoking VIEW to view data encoded in ASCII (or UTF-8) and the file is not tagged with a CCSID of 819 (or 1208).

When ASCII is specified or the file is tagged with CCSID 819, the editor renders the ASCII data readable by converting it to the CCSID of the terminal. Also, if set for a z/OS UNIX file, the editor breaks up the data into records using the ASCII linefeed character (X'0A') and the ASCII carriage return character (X'0D') as the record delimiter. For z/OS UNIX files, the linefeed and carriage return characters are removed from the data loaded into the editor but written back to the file when the data is saved.

When UTF8 is specified, or the file is tagged with CCSID 1208, the equivalent actions happen, except for UTF-8 instead of ASCII.

**tabname**

The name of a user line command table to be provided by the service caller.

**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.

**file-var**

The name of an ISPF variable containing the path name for a z/OS UNIX regular file or directory. An absolute path name or a path name relative to the current working directory can be specified. Absolute path names begin with '/'. Relative path names begin with '..'. If the path name is for a directory, a directory selection list 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

These 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.

**10**

Member or generation (if specified) not found.

**12**

VIEW has been disabled through the ISPF configuration table or the ws-filename-buffer-name parameter was specified.

**14**

Member, sequential data set, or z/OS UNIX file in use.

**16**

Either:

- 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.

## Examples

See:

- [“Example 1: ” on page 302](#)
- [“Example 2: ” on page 302](#)

### Example 1:

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

#### **Command invocation**

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

or

```
ISPEXEC LMINIT DATAID(EDT) DATASET('ISPFPROJ.FTOUTPUT')
ISPEXEC VIEW DATAID(&EDT) MEMBER(TELOUT)
```

#### **Call invocation**

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

or

Set the program variable BUFFER to contain:

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

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

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

### Example 2:

This example invokes the VIEW service for z/OS UNIX file /u/user1/filea.

#### **Command invocation**

```
FILEVAR='/u/user1/filea'
ISPEXEC VIEW FILE(FILEVAR)
```

## Call invocation

```
FILEVAR='/u/user1/filea';
CALL ISPLINK('VIEW',,,'FILEVAR');
```

## 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 (and CREATE and REPLACE commands when a write routine is specified). If this routine is not specified, ISPF processes these commands.
- Write the records selected for the CREATE and REPLACE primary commands, accepting one record from Edit on each invocation.

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 ZEUSER dialog variable in the shared pool. When the system initializes the recovery data set, the system also saves the data in ZEUSER 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 ZEUSER 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_<u>      </u>' — , <u>data-name</u> , <u>profile-name</u> — , <u>rec-format</u> — ➤

➤➤ , <u>rec-len</u> — , <u>read-routine</u> — , <u>cmd-routine</u> , <u>dialog-data</u> ➤

➤➤ , <u>edit-len</u> , <u>panel-name</u> , <u>macro-name</u> ➤

➤➤ , <u>format-name</u> , { <u>'NO_<u>      </u>'</u> , <u>'NO_<u>      </u>'</u> } , { <u>'YES_<u>      </u>'</u> , <u>'YES_<u>      </u>'</u> } ➤

➤➤ , <u>parm-var</u> , <u>write-routine</u> , { <u>'YES_<u>      </u>'</u> , <u>'NO_<u>      </u>'</u> } ➤

➤➤ , <u>tabname</u> ); ➤➤

```

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 this format for VIIF, specifying YES for the recovery-request parameter:

```

➤➤ CALL — ISPLINK — ('VIIF_<u>      </u>' — , <u>data-name</u> , <u>'<u>      </u>'</u> — , <u>rec-format</u> , ➤

➤➤ <u>rec-len</u> , <u>read-routine</u> — , <u>cmd-routine</u> , <u>dialog-data</u> ➤

➤➤ , <u>'<u>      </u>'</u> — , <u>'<u>      </u>'</u> — , <u>'<u>      </u>'</u> — , <u>'<u>      </u>'</u> — , <u>'YES_<u>      </u>'</u> — , <u>'<u>      </u>'</u> — , <u>write-routine</u> , <u>'<u>      </u>'</u> ➤

➤➤ , <u>tabname</u> ); ➤➤

```

## 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 is 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 is 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 is 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). It is recommended that the high-order bit of this value be set ON. See [“Read routine” on page 306](#) 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. This routine also processes the CREATE and REPLACE primary commands when the address of a write-routine is specified as a parameter on the VIIF call. It is recommended that the high-order bit of this value be set ON. See [“Command routine” on page 307](#) 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 bytes from (edit-len + 1) to rec-len are not displayed. That means the inaccessible record data is at the end of the record.

The edit-len parameter is optional when recovery-request is 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 is NO (or is not specified); otherwise, it is not allowed. The default is the standard View data display panel. See [z/OS 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 is NO (or is not specified); otherwise, it is not allowed. The default is no initial macro. See [z/OS 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 is 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 is 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 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.

**write-routine**

A fullword address indicating the entry point of a dialog- supplied write routine used to handle the writing of records for the CREATE and REPLACE primary commands. It is recommended that the high-order bit of this value be set ON. If a write-routine is not supplied, ISPF handles the writing of records for the CREATE and REPLACE primary commands. See [“Write routine” on page 307](#) for more information about this parameter.

**NO|YES (change warning)**

Specifies whether a warning message is issued on the first change of data. If you specify YES, the VIIF 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.

**tabname**

The name of a user line command table to be used for the view session. The value must be 8 characters, blank padded.

## 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 or write routine must have an assembler interface to be used in a call to VIIF.

The dialog-supplied 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:**

1. The dialog-supplied routines can be written in languages that use the Language Environment runtime environment. However, a mixture of Language Environment-conforming main dialog code and service routine code is not supported. Dialogs and service routines must either all be Language Environment-conforming or all be Language Environment-nonconforming.
2. Language Environment applications that use the ISPF EDIF, VIIF, or BRIF service must use the Language Environment OS\_UPSTACK option. For ISPF to invoke the user routines with a valid LE dynamic save area, the Language Environment application must issue a CONTROL LE ON service request before each EDIF, VIIF, or BRIF service request and a CONTROL LE OFF service request after each EDIF, VIIF, or BRIF service request.

## 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 these 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. If the address of a write-routine is specified as a parameter on the VIIF call, the command routine also processes the CREATE and REPLACE primary commands. The command routine is invoked with these parameters:

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

```
1n Move
2n Copy
3n Create
4n Replace
5n 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, COPY, CREATE, or REPLACE, the command routine initiates the processing for the requested function. When the return code from the command routine is zero, VIIF 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 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.

## Write routine

VIIF 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 called repeatedly to write the data records selected for the CREATE or REPLACE command. Flags are passed to the write routine to indicate the source and change status for each record.

The dialog-supplied write routine is invoked with these 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).

- 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 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.

## Write routine return codes

**0**

Normal completion.

**16**

Output error, return to View mode.

**20**

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

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);
```

## 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 TDBISPL. 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

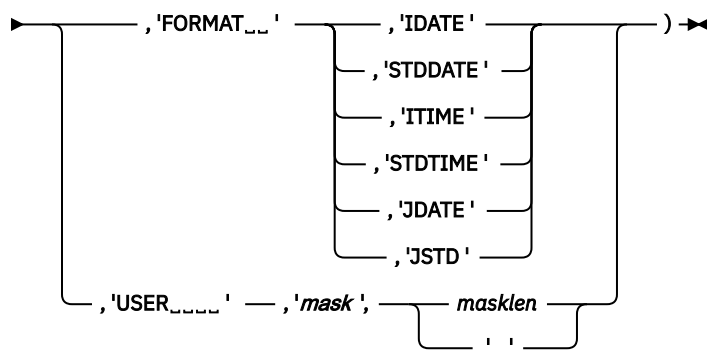
The mask is associated with the A definition of VAR1, not the B definition.

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* →



## 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, these 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, these parameters must be defined:

#### **mask**

Identifies the mask pattern associated with the dialog variable.

A mask pattern can consist of 20 characters. These are 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

These 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.

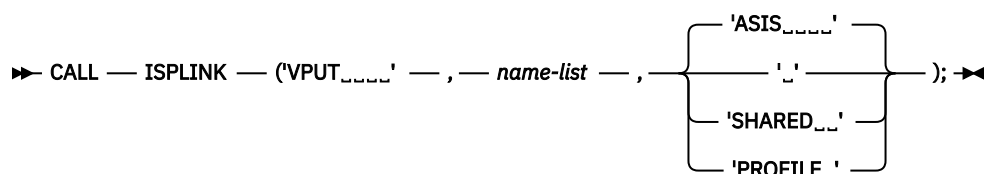
### Command invocation format



### Call invocation format

>> CALL — ISPEXEC — (buf-len, — buffer); >>

or



### 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

These return codes are possible:

0

Normal completion.

## VREPLACE

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. Enter the command:

```
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

These 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

These return codes are possible:

**0**

Normal completion.

20

Severe error.

## Example

Remove ISPF accessibility to all PL/I program variables.

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

## VSYM—resolve system symbols

The VSYM service updates the value of dialog variables found in the function pool by resolving the values of any system symbols. This includes all system static symbols and dynamic symbols and any user-defined static symbols. The *z/OS MVS Initialization and Tuning Reference* has details on system static and dynamic symbols. Consult your system programmer for any locally defined user symbols as these are system and installation dependent. If the named dialog variable is not found in the function pool, it is not created.

## Command invocation format

➤ ISPEXEC — VSYM — *name-list* ➤

## Call invocation format

➤ CALL — ISPEXEC — (*buf-len*, — *buffer*); ➤

or

➤ CALL — ISPLINK — ('VSYM\_....' — , — *name-list*); ➤

## Parameters

### name-list

Specifies the names of one or more dialog variables whose values in the function pool are to be processed to resolve system symbols. The names are passed in the standard name-list format.

### 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

These return codes are possible:

**0**

Normal completion.

**4**

One or more symbol names not substituted (no corresponding system symbol was found).

**8**

Variable not found in function pool.

**12**

Validation failed.

**16**

Truncation occurred resolving system symbols.

**20**

Severe error.

## Example

In a CLIST, define the variable DSNLVL to contain a data set prefix of SYS2, and the second qualifier to that of the sysplex on which the command is executed.

```
SET DSNLVL = SYS2.&&SYSPLEX  
ISPEXEC VSYM (DSNLVL)
```

When executed on a system that is a member of a sysplex named SYSPLEX1, the resulting value of DSNLVL is SYS2.SYSPLEX1.

The same example in REXX is:

```
DSNLVL = 'SYS2.&SYSPLEX'  
address "ISPEXEC"  
"VSYM (DSNLVL)"
```



## Appendix A. JSON API

### JSON data structures and variables used to communicate between ISPF and a client

This section describes the format of the JSON (Javascript Object Notation) data structures that can be exchanged between ISPF and a client to allow the client to process ISPF panels. These JSON data structures were developed for the implementation of ISPF application processing in z/OSMF (z/OS System Management Facility). With z/OS 1.13, z/OSMF is able to display in a web browser the panels for an ISPF application.

ISPF creates and sends to the client a JSON data structure that describes the layout of an ISPF panel. The client handles the display processing for the panel and is responsible for creating and sending to ISPF a JSON data structure that describes the user response for the panel.

JSON data structures are also used by TSO to communicate any TSO messages to the client. For TSO messages that require a user response, the client is required to return a JSON data structure that describes the response.

A z/OS UNIX message queue is used to handle the exchange of JSON data structures between ISPF and the client. The TSO launcher component of CEA (Common Event Adapter) must be invoked to create the TSO address space used to run ISPF because the launcher handles the creation of the z/OS UNIX message queue and passing of the message queue identifier to TSO and ISPF. Message type identifiers are used to identify the different messages sent from ISPF, TSO and the client.

#### Message type Description

- |          |  |
|----------|--|
| <b>2</b> | JSON data structure sent from TSO to client  |
| <b>3</b> | JSON data structure sent from ISPF to client |
| <b>7</b> | JSON data structure sent from client to TSO  |
| <b>8</b> | JSON data structure sent from client to ISPF |

The following sections provide JSON schemas to describe the JSON data structures used to communicate between TSO/ISPF and the client.

### JSON data structures sent from TSO to client (message type 2)

#### TSO Message JSON

The following schema describes the JSON data structure for a TSO message:

```
{
  "TSO MESSAGE":{
    "description":"TSO message for the client",
    "type":"object",
    "properties":{
      "VER":{
        "description":"TSO message JSON version identifier",
        "type":"string",
        "maxLength":4,
        "required":true
      },

```

## JSON data structures sent from ISPF to client (message type 3)

```
    "DATA":{
      "description":"message text",
      "type":"string",
      "maxLength":32767,
      "required":true
    }
  }
}
```

### TSO message JSON example

Here is an example of the JSON for the message generated when the user enters the TSO command LISTCAT LVL:

```
{
  "TSO MESSAGE":{
    "VERSION":"0100",
    "DATA":"ENTER LEVEL NAME -   "
  }
}
```

### TSO prompt JSON

The following schema describes the JSON data structure generated when TSO requires a response from the user:

```
{
  "TSO PROMPT":{
    "description":"TSO prompt request for the client",
    "type":"object",
    "properties":{
      "VER":{
        "description":"TSO prompt JSON version identifier",
        "type":"string",
        "maxLength":4,
        "required":true
      },
      "HIDDEN":{
        "description":"Specifies if the response should be hidden",
        "type":"string",
        "maxLength":5,
        "enum":["TRUE","FALSE"],
        "required":true
      }
    }
  }
}
```

### TSO prompt JSON example

Here is an example of the JSON when the user enters a command like LISTCAT LVL and TSO requires the user to provide a response:

```
{
  "TSO PROMPT":{
    "VERSION":"0100",
    "HIDDEN":"FALSE"
  }
}
```

## JSON data structures sent from ISPF to client (message type 3)



## ISPF panel display JSON

The following schema describes the JSON data structure for an ISPF panel display:

```
{
  "PNL":{
    "description":"ISPF panel display data and format",
    "type":"object",
    "properties":{
      "VER":{
        "description":"ISPF panel JSON version identifier",
        "type":"string",
        "maxLength":4,
        "required":true
      },
      "NME":{
        "description":"Panel name",
        "type":"string",
        "maxLength":8,
        "required":true
      },
      "SCR":{
        "description":"ISPF logical screen identifier",
        "type":"string",
        "maxLength":1,
        "required":true
      },
      "SCN":{
        "description":"Screen name defined for this panel",
        "type":"string",
        "maxLength":8
      },
      "HDL":{
        "description":"Handle value - used internally by ISPF",
        "type":"string",
        "maxLength":12,
        "required":true
      },
      "RWS":{
        "description":"Panel height - number of rows",
        "type":"integer",
        "maximum":204,
        "required":true
      },
      "CLS":{
        "description":"Panel width - number of columns",
        "type":"integer",
        "maximum":160,
        "required":true
      },
      "TLE":{
        "description":"Panel title",
        "type":"string"
      },
      "EDT":{
        "description":"Edit data display indicator",
        "type":"string",
        "enum":["TRUE"]
      },
      "CML":{
        "description":"Command line location",
        "type":"string",
        "enum":["BOTTOM","ASIS","NONE"],
        "required":true
      },
      "ALA":{
        "description":"Indicates an alarm should sound when the panel is displayed",
        "type":"string",
        "enum":["TRUE"]
      },
      "CUR":{
        "description":"The cursor location on the panel",
        "type":"object",
        "required":true,
        "properties":{
          "ROW":{
            "description":"The number of the panel row the cursor is on",
            "type":"integer",
            "maximum":204,
            "required":true
          }
        }
      }
    }
  }
}
```

## JSON data structures sent from ISPF to client (message type 3)

```
{
  "CLM":{
    "description":"The number of the panel column the cursor is on",
    "type":"integer",
    "maximum":160,
    "required":true
  },
  "SUP":{
    "description":"Indicates the current panel display can be scrolled up",
    "type":"string",
    "enum":["TRUE"]
  },
  "SDN":{
    "description":"Indicates the current panel display can be scrolled down",
    "type":"string",
    "enum":["TRUE"]
  },
  "SLF":{
    "description":"Indicates the current panel display can be scrolled left",
    "type":"string",
    "enum":["TRUE"]
  },
  "SRG":{
    "description":"Indicates the current panel display can be scrolled right",
    "type":"string",
    "enum":["TRUE"]
  },
  "IFY":{
    "description":"Identifiers currently active for this panel display",
    "type":"object",
    "required":true,
    "properties":{
      "SYS":{
        "description":"The system ID currently displayed with the panel",
        "type":"string",
        "maxLength":8,
        "required":true
      },
      "UID":{
        "description":"The user ID currently displayed with the panel",
        "type":"string",
        "maxLength":8,
        "required":true
      },
      "PNL":{
        "description":"Indicates that the panel name is currently displayed with the panel",
        "type":"string",
        "enum":["TRUE"]
      },
      "SCR":{
        "description":"Indicates that the screen ID is currently displayed with the panel",
        "type":"string",
        "enum":["TRUE"]
      }
    }
  },
  "MSG":{
    "description":"Details of a message displayed with the panel",
    "type":"object",
    "properties":{
      "ID":{
        "description":"The message identifier",
        "type":"string",
        "maxLength":8
      },
      "TYP":{
        "description":"The message type as defined on the TYPE keyword for the message",
        "type":"string",
        "maxLength":1,
        "enum":["N","W","A","C"]
      },
      "HLP":{
        "description":"Indicates there is no help associated with this message  
(i.e. no help panel or long message for a short message)",
        "type":"string"
      },
      "LEN":{
        "description":"Length of the message text",
        "type":"integer"
      }
    }
  },
}
```

```

    "TXT":{
      "description":"The text for the message",
      "type":"string",
      "maxLength":512
    }
  },
  "ARE":{
    "description":"The scrollable, dynamic, and table model areas defined for the panel",
    "type":"array",
    "items":{
      "NME":{
        "description":"The name used to define to dynamic or scrollable area.
For a table model area this is the table name.",
        "type":"string",
        "maxLength":8
      },
      "TYP":{
        "description":"The type of area - D=dynamic, S=scrollable, T=table model",
        "type":"string",
        "maxLength":1,
        "enum":["D","S","T"]
      },
      "TOP":{
        "description":"The panel row where the top of this area is located",
        "type":"integer"
      },
      "BOT":{
        "description":"The panel row where the bottom of this area is located",
        "type":"integer"
      },
      "LFT":{
        "description":"The panel row where the left side of this area is located",
        "type":"integer"
      },
      "RGT":{
        "description":"The panel row where the right side of this area is located",
        "type":"integer"
      }
    }
  },
  "CHA":{
    "description":"The color and highlighting defined for the character level attributes
to be used in the dynamic areas for the panel",
    "type":"array",
    "items":{
      "COL":{
        "description":"The color assigned for this character level attribute",
        "type":"string",
        "enum":["AQUA","BLUE","GREEN","PINK","RED","WHITE","YELLOW"]
      },
      "HIL":{
        "description":"The highlighting assigned for this character level attribute",
        "type":"string",
        "enum":["BLINK","RVIDEO","USCORE"]
      }
    }
  },
  "MNU":{
    "description":"The action bar choices and pull-down menus for the panel",
    "type":"array",
    "items":{
      "PUL":{
        "description":"Pull-down menu description text",
        "type":"string"
      },
      "CHS":{
        "description":"The choices for the pull-down menu",
        "type":"array",
        "items":{
          "N":{
            "description":"Pull-down choice description text",
            "type":"string"
          },
          "I":{
            "description":"An identifier used to indicate when the pull-down choice has
been selected",
            "type":"string"
          }
        }
      }
    }
  }
}

```

```

},
"FLD":{
  "description":"The input, output, point-and-shoot, and text fields displayed on the panel",
  "type":"array",
  "items":{
    "T":{
      "description":"The field type - I=input, O=output, P=point-and-shoot, T=text",
      "type":"string",
      "enum":["I","O","P","T"]
    },
    "P":{
      "description":"Indicates this input field is also a point-and-shoot field",
      "type":"string",
      "enum":["TRUE"]
    },
    "Z":{
      "description":"Indicates this is the command field for the panel",
      "type":"string",
      "enum":["TRUE"]
    }
  },
  "A":{
    "description":"The name of the scrollable, dynamic, or table model area where this field is located",
    "type":"string",
    "maxLength":8
  },
  "Y":{
    "description":"The panel row where the field is located",
    "type":"integer",
    "maximum":204
  },
  "X":{
    "description":"The panel column where the field is located",
    "type":"integer",
    "maximum":160
  },
  "C":{
    "description":"The color of the field",
    "type":"string",
    "enum":["AQUA","BLUE","GREEN","PINK","RED","WHITE","YELLOW"]
  },
  "I":{
    "description":"The intensity of the field",
    "type":"string",
    "enum":["HIGH","LOW","NON"]
  },
  "H":{
    "description":"The highlighting for the field",
    "type":"string",
    "enum":["BLINK","RVIDEO","USCORE"]
  },
  "L":{
    "description":"The length of the field",
    "type":"integer"
  },
  "LX":{
    "description":"The total length for a scrollable field",
    "type":"integer",
    "maximum":32767
  },
  "N":{
    "description":"The name associated with the field",
    "type":"string",
    "maxLength":14
  },
  "D":{
    "description":"The data for the field",
    "type":"string",
    "maxLength":32767
  },
  "SL":{
    "description":"Identifies a selection field defined as a check box or radio button",
    "type":"object",
    "properties":{
      "T":{
        "description":"Type of selection field - CB=check box, RD=radio button",
        "type":"string",
        "enum":["CB","RD"]
      },
      "V":{
        "description":"The data value associated with selecting this field",
        "type":"string"
      }
    }
  }
}

```

```

    },
    "N": {
      "description": "For a radio button selection, the name of the input field to
be updated with the associated value",
      "type": "string",
      "maxLength": 8
    },
    "G": {
      "description": "For a radio button selection, the group number used to
identify related selection fields",
      "type": "integer",
      "maximum": 99
    }
  },
  "SHS": {
    "description": "For a field in a dynamic area, identifies the sections of the
field that are highlighted using character level (shadow) attributes",
    "type": "array",
    "items": {
      "STA": {
        "description": "The starting position within the field for a set of
characters highlighted by a character level attribute",
        "type": "integer"
      },
      "LEN": {
        "description": "The number of characters from the starting position that are
highlighted by the character level attribute",
        "type": "integer"
      },
      "ATT": {
        "description": "The number identifying the entry in the \"CHA\" array (see
above) for the character level attribute used to highlight this section of the field",
        "type": "integer"
      }
    }
  },
  "KEY": {
    "description": "The function keys defined for display with this panel",
    "type": "array",
    "items": {
      "K": {
        "description": "The function key identifier",
        "type": "string",
        "enum": ["ENTER", "1", "2", "3", "4", "5", "6", "7", "8", "9", "11", "12", "13", "14", "15",
"16", "17", "18", "19", "20", "21", "22", "23", "24"]
      },
      "N": {
        "description": "The label defined for the function key",
        "type": "string"
      }
    }
  }
}

```

## ISPF panel display JSON examples

Here is an example of the JSON for a display of the ISPF primary options menu:

```

{
  "PNL": {
    "VER": "0100",
    "NME": "ISR@PRIM",
    "SCR": "1",
    "HDL": "000162984204",
    "RWS": 24,
    "CLS": 80,
    "TLE": "ISPF Primary Option Menu",
    "CML": "BOTTOM",
    "CUR": {
      "ROW": 24,
      "CLM": 14
    },
    "IFY": {
      "SYS": "",

```

```

    "UID": ""
  },
  "MSG": {
    "ID": "ISRL0999",
    "TYP": "N",
    "HLP": "FALSE",
    "LEN": 423,
    "TXT": "Licensed Materials - Property of IBM
5650-ZOS      Copyright IBM Corp. 1980, 2012.
All rights reserved.
US Government Users Restricted Rights -
Use, duplication or disclosure restricted
by GSA ADP Schedule Contract with IBM Corp."
  },
  "ARE": [ {
    "NME": "TMPROWS",
    "TYP": "D",
    "TOP": 5,
    "BOT": 15,
    "LFT": 58,
    "RGT": 78
  },
  {
    "NME": "SAREA39",
    "TYP": "S",
    "TOP": 4,
    "BOT": 17,
    "LFT": 1,
    "RGT": 56
  }
],
  "CHA": [ {
    "COL": "WHITE"
  },
  {
    "COL": "RED"
  },
  {
    "COL": "BLUE"
  },
  {
    "COL": "GREEN"
  },
  {
    "COL": "PINK"
  },
  {
    "COL": "YELLOW"
  },
  {
    "COL": "AQUA"
  },
  {
    "COL": "WHITE",
    "HIL": "RVIDEO"
  },
  {
    "COL": "RED",
    "HIL": "RVIDEO"
  },
  {
    "COL": "BLUE",
    "HIL": "RVIDEO"
  },
  {
    "COL": "GREEN",
    "HIL": "RVIDEO"
  },
  {
    "COL": "PINK",
    "HIL": "RVIDEO"
  },
  {
    "COL": "YELLOW",
    "HIL": "RVIDEO"
  },
  {
    "COL": "AQUA",
    "HIL": "RVIDEO"
  }
],
  "MNU": [ {

```

```

    "PUL": " Menu",
    "CHS": [
      {
        "N": "Settings",
        "I": "1-1"
      },
      {
        "N": "View",
        "I": "1-2"
      },
      {
        "N": "Edit",
        "I": "1-3"
      },
      {
        "N": "ISPF Command Shell",
        "I": "1-4"
      },
      {
        "N": "Dialog Test...",
        "I": "1-5"
      },
      {
        "N": "Other IBM Products...",
        "I": "1-6"
      },
      {
        "N": "SCLM",
        "I": "1-7"
      },
      {
        "N": "ISPF Workplace",
        "I": "1-8"
      },
      {
        "N": "Status Area...",
        "I": "1-9"
      },
      {
        "N": "Exit",
        "I": "1-10"
      }
    ]
  },
  {
    "PUL": " Utilities",
    "CHS": [
      {
        "N": "Library",
        "I": "2-1"
      },
      {
        "N": "Data set",
        "I": "2-2"
      },
      {
        "N": "Move/Copy",
        "I": "2-3"
      },
      {
        "N": "Data Set List",
        "I": "2-4"
      },
      {
        "N": "Reset Statistics",
        "I": "2-5"
      },
      {
        "N": "Hardcopy",
        "I": "2-6"
      },
      {
        "N": "Reserved",
        "I": "2-7",
        "D": "TRUE"
      },
      {
        "N": "Outlist",
        "I": "2-8"
      },
      {
        "N": "Commands...",
        "I": "2-9"
      }
    ]
  }
]

```

```

    {
      "N": "Reserved",
      "I": "2-10",
      "D": "TRUE"
    },
    {
      "N": "Format",
      "I": "2-11"
    },
    {
      "N": "SuperC",
      "I": "2-12"
    },
    {
      "N": "SuperCE",
      "I": "2-13"
    },
    {
      "N": "Search-For",
      "I": "2-14"
    },
    {
      "N": "Search-ForE",
      "I": "2-15"
    },
    {
      "N": "Table Utility",
      "I": "2-16"
    },
    {
      "N": "Directory List",
      "I": "2-17"
    }
  ]
},
{
  "PUL": "Compilers",
  "CHS": [
    {
      "N": "Foreground Compilers",
      "I": "3-1"
    },
    {
      "N": "Background Compilers",
      "I": "3-2"
    },
    {
      "N": "ISPPREP Panel Utility...",
      "I": "3-3"
    },
    {
      "N": "DTL Compiler...",
      "I": "3-4"
    }
  ]
},
{
  "PUL": "Options",
  "CHS": [
    {
      "N": "General Settings",
      "I": "4-1"
    },
    {
      "N": "CUA Attributes...",
      "I": "4-2"
    },
    {
      "N": "Keylists...",
      "I": "4-3"
    },
    {
      "N": "Point-and-Shoot...",
      "I": "4-4"
    },
    {
      "N": "Colors...",
      "I": "4-5"
    },
    {
      "N": "Dialog Test appl ID...",
      "I": "4-6"
    }
  ]
}

```



```
]
"PUL": "Status",
"CHS": [{
    "N": "Session",
    "I": "5-1",
    "D": "TRUE"
},
{
    "N": "Function keys",
    "I": "5-2"
},
{
    "N": "Calendar",
    "I": "5-3"
},
{
    "N": "User status",
    "I": "5-4"
},
{
    "N": "User point and shoot",
    "I": "5-5"
},
{
    "N": "None",
    "I": "5-6"
}
]

"PUL": "Help",
"CHS": [{
    "N": "General",
    "I": "6-1"
},
{
    "N": "Settings",
    "I": "6-2"
},
{
    "N": "View",
    "I": "6-3"
},
{
    "N": "Edit",
    "I": "6-4"
},
{
    "N": "Utilities",
    "I": "6-5"
},
{
    "N": "Foreground",
    "I": "6-6"
},
{
    "N": "Batch",
    "I": "6-7"
},
{
    "N": "Command",
    "I": "6-8"
},
{
    "N": "Dialog Test",
    "I": "6-9"
},
{
    "N": "IBM Products",
    "I": "6-10"
},
{
    "N": "SCLM",
    "I": "6-11"
},
{
    "N": "Workplace",
    "I": "6-12"
}
```

```

        "N": "Exit",
        "I": "6-13"
    },
    {
        "N": "Status Area",
        "I": "6-14"
    },
    {
        "N": "About...",
        "I": "6-15"
    },
    {
        "N": "Changes for this Release",
        "I": "6-16"
    },
    {
        "N": "Tutorial",
        "I": "6-17"
    },
    {
        "N": "Appendices",
        "I": "6-18"
    },
    {
        "N": "Index",
        "I": "6-19"
    }
]
},
"FLD": [{
    "T": "T",
    "Y": 3,
    "X": 29,
    "C": "BLUE",
    "I": "LOW",
    "L": 24,
    "D": "ISPF Primary Option Menu"
},
{
    "T": "T",
    "Y": 5,
    "X": 2,
    "C": "WHITE",
    "I": "LOW",
    "L": 1,
    "D": "0"
},
{
    "T": "P",
    "Y": 5,
    "X": 5,
    "C": "AQUA",
    "I": "HIGH",
    "L": 13,
    "N": "0PS-6-5",
    "D": "Settings      "
},
{
    "T": "T",
    "Y": 5,
    "X": 19,
    "C": "GREEN",
    "I": "LOW",
    "L": 28,
    "D": "Terminal and user parameters"
},
{
    "T": "T",
    "Y": 5,
    "X": 58,
    "C": "GREEN",
    "I": "LOW",
    "L": 12,
    "D": " User ID . ."
},
{
    "T": "T",
    "A": "TMPROWS ",
    "Y": 5,
    "X": 71,
    "C": "AQUA",

```

```

    "I": "LOW",
    "L": 7,
    "D": "PVANDYK"
  },
  {
    "T": "T",
    "Y": 6,
    "X": 2,
    "C": "WHITE",
    "I": "LOW",
    "L": 1,
    "D": "1"
  },
  {
    "T": "P",
    "Y": 6,
    "X": 5,
    "C": "AQUA",
    "I": "HIGH",
    "L": 13,
    "N": "ØPS-7-5",
    "D": "View"
  },
  {
    "T": "T",
    "Y": 6,
    "X": 19,
    "C": "GREEN",
    "I": "LOW",
    "L": 31,
    "D": "Display source data or listings"
  },
  {
    "T": "T",
    "Y": 6,
    "X": 58,
    "C": "GREEN",
    "I": "LOW",
    "L": 12,
    "D": " Time. . . ."
  },
  {
    "T": "T",
    "A": "TMPROWS ",
    "Y": 6,
    "X": 71,
    "C": "AQUA",
    "I": "LOW",
    "L": 5,
    "D": "12:38"
  },
  {
    "T": "T",
    "Y": 7,
    "X": 2,
    "C": "WHITE",
    "I": "LOW",
    "L": 1,
    "D": "2"
  },
  {
    "T": "P",
    "Y": 7,
    "X": 5,
    "C": "AQUA",
    "I": "HIGH",
    "L": 13,
    "N": "ØPS-8-5",
    "D": "Edit"
  },
  {
    "T": "T",
    "Y": 7,
    "X": 19,
    "C": "GREEN",
    "I": "LOW",
    "L": 28,
    "D": "Create or change source data"
  },
  {
    "T": "T",
    "Y": 7,

```

```

    "X":58,
    "C":"GREEN",
    "I":"LOW",
    "L":12,
    "D":" Terminal. : "
  },
  {
    "T":"T",
    "A":"TMPROWS ",
    "Y":7,
    "X":71,
    "C":"AQUA",
    "I":"LOW",
    "L":4,
    "D":"3278"
  },
  {
    "T":"T",
    "Y":8,
    "X":2,
    "C":"WHITE",
    "I":"LOW",
    "L":1,
    "D":"3"
  },
  {
    "T":"P",
    "Y":8,
    "X":5,
    "C":"AQUA",
    "I":"HIGH",
    "L":13,
    "N":"0PS-9-5",
    "D":"Utilities      "
  },
  {
    "T":"T",
    "Y":8,
    "X":19,
    "C":"GREEN",
    "I":"LOW",
    "L":25,
    "D":"Perform utility functions"
  },
  {
    "T":"T",
    "Y":8,
    "X":58,
    "C":"GREEN",
    "I":"LOW",
    "L":12,
    "D":" Screen. . : "
  },
  {
    "T":"T",
    "A":"TMPROWS ",
    "Y":8,
    "X":71,
    "C":"AQUA",
    "I":"LOW",
    "L":1,
    "D":"1"
  },
  {
    "T":"T",
    "Y":9,
    "X":2,
    "C":"WHITE",
    "I":"LOW",
    "L":1,
    "D":"4"
  },
  {
    "T":"P",
    "Y":9,
    "X":5,
    "C":"AQUA",
    "I":"HIGH",
    "L":13,
    "N":"0PS-10-5",
    "D":"Foreground      "
  },
  }

```

```

{
  "T": "T",
  "Y": 9,
  "X": 19,
  "C": "GREEN",
  "I": "LOW",
  "L": 31,
  "D": "Interactive language processing"
},
{
  "T": "T",
  "Y": 9,
  "X": 58,
  "C": "GREEN",
  "I": "LOW",
  "L": 12,
  "D": " Language. ."
},
{
  "T": "T",
  "A": "TMPROWS ",
  "Y": 9,
  "X": 71,
  "C": "AQUA",
  "I": "LOW",
  "L": 7,
  "D": "ENGLISH"
},
{
  "T": "T",
  "Y": 10,
  "X": 2,
  "C": "WHITE",
  "I": "LOW",
  "L": 1,
  "D": "5"
},
{
  "T": "P",
  "Y": 10,
  "X": 5,
  "C": "AQUA",
  "I": "HIGH",
  "L": 13,
  "N": "ØPS-11-5",
  "D": "Batch"
},
{
  "T": "T",
  "Y": 10,
  "X": 19,
  "C": "GREEN",
  "I": "LOW",
  "L": 34,
  "D": "Submit job for language processing"
},
{
  "T": "T",
  "Y": 10,
  "X": 58,
  "C": "GREEN",
  "I": "LOW",
  "L": 12,
  "D": " Appl ID . ."
},
{
  "T": "T",
  "A": "TMPROWS ",
  "Y": 10,
  "X": 71,
  "C": "AQUA",
  "I": "LOW",
  "L": 3,
  "D": "ISR"
},
{
  "T": "T",
  "Y": 11,
  "X": 2,
  "C": "WHITE",
  "I": "LOW",
  "L": 1,

```

```

    {
      "D": "6"
    },
    {
      "T": "P",
      "Y": 11,
      "X": 5,
      "C": "AQUA",
      "I": "HIGH",
      "L": 13,
      "N": "0PS-12-5",
      "D": "Command"
    },
    {
      "T": "T",
      "Y": 11,
      "X": 19,
      "C": "GREEN",
      "I": "LOW",
      "L": 33,
      "D": "Enter TSO commands"
    },
    {
      "T": "T",
      "Y": 11,
      "X": 58,
      "C": "GREEN",
      "I": "LOW",
      "L": 12,
      "D": " TSO logon :"
    },
    {
      "T": "T",
      "A": "TMPROWS ",
      "Y": 11,
      "X": 71,
      "C": "AQUA",
      "I": "LOW",
      "L": 5,
      "D": "STEP1"
    },
    {
      "T": "T",
      "Y": 12,
      "X": 2,
      "C": "WHITE",
      "I": "LOW",
      "L": 1,
      "D": "7"
    },
    {
      "T": "P",
      "Y": 12,
      "X": 5,
      "C": "AQUA",
      "I": "HIGH",
      "L": 13,
      "N": "0PS-13-5",
      "D": "Dialog Test"
    },
    {
      "T": "T",
      "Y": 12,
      "X": 19,
      "C": "GREEN",
      "I": "LOW",
      "L": 22,
      "D": "Perform dialog testing"
    },
    {
      "T": "T",
      "Y": 12,
      "X": 58,
      "C": "GREEN",
      "I": "LOW",
      "L": 12,
      "D": " TSO prefix:"
    },
    {
      "T": "T",
      "A": "TMPROWS ",
      "Y": 12,
      "X": 71,

```

```

    "C": "AQUA",
    "I": "LOW",
    "L": 7,
    "D": "PVANDYK"
  },
  {
    "T": "T",
    "Y": 13,
    "X": 2,
    "C": "WHITE",
    "I": "LOW",
    "L": 1,
    "D": "9"
  },
  {
    "T": "P",
    "Y": 13,
    "X": 5,
    "C": "AQUA",
    "I": "HIGH",
    "L": 13,
    "N": "0PS-14-5",
    "D": "IBM Products "
  },
  {
    "T": "T",
    "Y": 13,
    "X": 19,
    "C": "GREEN",
    "I": "LOW",
    "L": 32,
    "D": "IBM program development products"
  },
  {
    "T": "P",
    "A": "TMPROWS ",
    "Y": 13,
    "X": 59,
    "C": "AQUA",
    "I": "HIGH",
    "L": 20,
    "N": "0PS-14-59",
    "D": "System ID : ISA2   "
  },
  {
    "T": "T",
    "Y": 14,
    "X": 2,
    "C": "WHITE",
    "I": "LOW",
    "L": 2,
    "D": "10"
  },
  {
    "T": "P",
    "Y": 14,
    "X": 5,
    "C": "AQUA",
    "I": "HIGH",
    "L": 13,
    "N": "0PS-15-5",
    "D": "SCLM      "
  },
  {
    "T": "T",
    "Y": 14,
    "X": 19,
    "C": "GREEN",
    "I": "LOW",
    "L": 32,
    "D": "SW Configuration Library Manager"
  },
  {
    "T": "T",
    "Y": 14,
    "X": 58,
    "C": "GREEN",
    "I": "LOW",
    "L": 12,
    "D": " MVS acct. ."
  }
]

```

```

      "T": "T",
      "A": "TMPROWS ",
      "Y": 14,
      "X": 71,
      "C": "AQUA",
      "I": "LOW",
      "L": 8,
      "D": "***NONE**"
    },
    {
      "T": "T",
      "Y": 15,
      "X": 2,
      "C": "WHITE",
      "I": "LOW",
      "L": 2,
      "D": "11"
    },
    {
      "T": "P",
      "Y": 15,
      "X": 5,
      "C": "AQUA",
      "I": "HIGH",
      "L": 13,
      "N": "0PS-16-5",
      "D": "Workplace "
    },
    {
      "T": "T",
      "Y": 15,
      "X": 19,
      "C": "GREEN",
      "I": "LOW",
      "L": 28,
      "D": "ISPF Object/Action Workplace"
    },
    {
      "T": "P",
      "A": "TMPROWS ",
      "Y": 15,
      "X": 59,
      "C": "AQUA",
      "I": "HIGH",
      "L": 20,
      "N": "0PS-16-59",
      "D": "Release . : ISPF 7.1"
    },
    {
      "T": "T",
      "Y": 16,
      "X": 2,
      "C": "WHITE",
      "I": "LOW",
      "L": 2,
      "D": "12"
    },
    {
      "T": "P",
      "Y": 16,
      "X": 5,
      "C": "AQUA",
      "I": "HIGH",
      "L": 13,
      "N": "0PS-17-5",
      "D": "z/OS System "
    },
    {
      "T": "T",
      "Y": 16,
      "X": 19,
      "C": "GREEN",
      "I": "LOW",
      "L": 35,
      "D": "z/OS system programmer applications"
    },
    {
      "T": "T",
      "Y": 17,
      "X": 2,
      "C": "WHITE",
      "I": "LOW",

```



```

    "L":2,
    "D":"13"
  },
  {
    "T":"P",
    "Y":17,
    "X":5,
    "C":"AQUA",
    "I":"HIGH",
    "L":13,
    "N":"0PS-18-5",
    "D":"z/OS User"
  },
  {
    "T":"T",
    "Y":17,
    "X":19,
    "C":"GREEN",
    "I":"LOW",
    "L":22,
    "D":"z/OS user applications"
  },
  {
    "T":"O",
    "Y":18,
    "X":2,
    "C":"AQUA",
    "I":"LOW",
    "L":4,
    "N":"ZEXI",
    "D":""
  },
  {
    "T":"T",
    "Y":19,
    "X":7,
    "C":"GREEN",
    "I":"LOW",
    "L":5,
    "D":"Enter"
  },
  {
    "T":"P",
    "Y":19,
    "X":13,
    "C":"AQUA",
    "I":"HIGH",
    "L":1,
    "N":"0PS-20-13",
    "D":"X"
  },
  {
    "T":"T",
    "Y":19,
    "X":15,
    "C":"GREEN",
    "I":"LOW",
    "L":36,
    "D":"to Terminate using log/list defaults"
  },
  {
    "T":"T",
    "Y":24,
    "X":2,
    "C":"GREEN",
    "I":"LOW",
    "L":11,
    "D":"Option ==>"
  },
  {
    "T":"I",
    "Z":"TRUE",
    "Y":24,
    "X":14,
    "C":"AQUA",
    "I":"LOW",
    "H":"USCORE",
    "L":66,
    "N":"ZCMD",
    "D":""
  }
],

```

## JSON data structures sent from ISPF to client (message type 3)

```
    "KEY": [{
      {
        "K": "ENTER",
        "N": "ENTER"
      },
      {
        "K": "1",
        "N": "Help"
      },
      {
        "K": "2",
        "N": "Split"
      },
      {
        "K": "3",
        "N": "Exit"
      },
      {
        "K": "7",
        "N": "Backward"
      },
      {
        "K": "8",
        "N": "Forward"
      },
      {
        "K": "9",
        "N": "Swap"
      },
      {
        "K": "10",
        "N": "Actions"
      },
      {
        "K": "12",
        "N": "Cancel"
      }
    ]
  }
}
```

Here is an example of the JSON for a display of the View entry panel (ISPF option 1):

```
{
  "PNL": {
    "VER": "0100",
    "NME": "ISRBR001",
    "SCR": "1",
    "SCN": "VIEW",
    "HDL": "001115829516",
    "RWS": 24,
    "CLS": 80,
    "TLE": "View Entry Panel",
    "CML": "BOTTOM",
    "CUR": {
      "ROW": 6,
      "CLM": 19
    },
    "IFY": {
      "SYS": "",
      "UID": ""
    },
    "ARE": [{
      {
        "NME": "SAREA39",
        "TYP": "S",
        "TOP": 4,
        "BOT": 23,
        "LFT": 1,
        "RGT": 80
      }
    ]
  },
  "MNU": [{
    {
      "PUL": " Menu",
      "CHS": [{
        {
          "N": "Settings",
          "I": "1-1"
        },
        {
          "N": "View",
          "I": "1-2",
          "D": "TRUE"
        }
      ]
    }
  ]
}
```

```

    },
    {
      "N": "Edit",
      "I": "1-3"
    },
    {
      "N": "ISPF Command Shell",
      "I": "1-4"
    },
    {
      "N": "Dialog Test...",
      "I": "1-5"
    },
    {
      "N": "Other IBM Products...",
      "I": "1-6"
    },
    {
      "N": "SCLM",
      "I": "1-7"
    },
    {
      "N": "ISPF Workplace",
      "I": "1-8"
    },
    {
      "N": "Status Area...",
      "I": "1-9"
    },
    {
      "N": "Exit",
      "I": "1-10"
    }
  ]
},
{
  "PUL": " RefList",
  "CHS": [
    {
      "N": "Current Data Set List (REFLIST)",
      "I": "2-1"
    },
    {
      "N": "Current Library List (REFLIST)",
      "I": "2-2"
    },
    {
      "N": "List of Personal Data Set Lists",
      "I": "2-3"
    },
    {
      "N": "List of Personal Library Lists",
      "I": "2-4"
    }
  ]
},
{
  "PUL": " RefMode",
  "CHS": [
    {
      "N": "List Execute",
      "I": "3-1"
    },
    {
      "N": "List Retrieve",
      "I": "3-2",
      "D": "TRUE"
    }
  ]
},
{
  "PUL": " Utilities",
  "CHS": [
    {
      "N": "Library",
      "I": "4-1"
    },
    {
      "N": "Data set",
      "I": "4-2"
    },
    {
      "N": "Move/Copy",
      "I": "4-3"
    }
  ]
},

```

## JSON data structures sent from ISPF to client (message type 3)

```
{
  "N": "Data Set List",
  "I": "4-4"
},
{
  "N": "Reset Statistics",
  "I": "4-5"
},
{
  "N": "Hardcopy",
  "I": "4-6"
},
{
  "N": "Reserved",
  "I": "4-7",
  "D": "TRUE"
},
{
  "N": "Outlist",
  "I": "4-8"
},
{
  "N": "Commands...",
  "I": "4-9"
},
{
  "N": "Reserved",
  "I": "4-10",
  "D": "TRUE"
},
{
  "N": "Format",
  "I": "4-11"
},
{
  "N": "SuperC",
  "I": "4-12"
},
{
  "N": "SuperCE",
  "I": "4-13"
},
{
  "N": "Search-For",
  "I": "4-14"
},
{
  "N": "Search-ForE",
  "I": "4-15"
},
{
  "N": "Table Utility",
  "I": "4-16"
},
{
  "N": "Directory List",
  "I": "4-17"
}
],
{
  "PUL": " Help",
  "CHS": [{
    "N": "General",
    "I": "6-1"
  },
  {
    "N": "View entry panel",
    "I": "6-2"
  },
  {
    "N": "Member selection list",
    "I": "6-3"
  },
  {
    "N": "Scrolling data",
    "I": "6-4"
  },
  {
    "N": "Types of Data Sets - View",
    "I": "6-5"
  }
]
```

```

        "I": "6-5"
      },
      {
        "N": "View display screen format",
        "I": "6-6"
      },
      {
        "N": "Sequence numbering - View",
        "I": "6-7"
      },
      {
        "N": "Display modes - View",
        "I": "6-8"
      },
      {
        "N": "Tabbing - View",
        "I": "6-9"
      },
      {
        "N": "Automatic recovery - View",
        "I": "6-10"
      },
      {
        "N": "Edit profiles",
        "I": "6-11"
      },
      {
        "N": "Edit line commands",
        "I": "6-12"
      },
      {
        "N": "Edit primary commands",
        "I": "6-13"
      },
      {
        "N": "Labels and line ranges - View",
        "I": "6-14"
      },
      {
        "N": "Ending a view session",
        "I": "6-15"
      },
      {
        "N": "Types of Data Sets - Browse",
        "I": "6-16"
      },
      {
        "N": "Browse display screen format",
        "I": "6-17"
      },
      {
        "N": "Assigning Browse labels",
        "I": "6-18"
      },
      {
        "N": "Browse commands",
        "I": "6-19"
      },
      {
        "N": "Terminating Browse function",
        "I": "6-20"
      },
      {
        "N": "Index",
        "I": "6-21"
      }
    ]
  },
  {
    "FLD": [
      {
        "T": "T",
        "Y": 3,
        "X": 33,
        "C": "BLUE",
        "I": "LOW",
        "L": 16,
        "D": "View Entry Panel"
      },
      {
        "T": "T",
        "Y": 5,
        "X": 2,

```

```

    "C": "BLUE",
    "I": "HIGH",
    "L": 13,
    "D": "ISPF Library:"
  },
  {
    "T": "T",
    "Y": 6,
    "X": 5,
    "C": "GREEN",
    "I": "LOW",
    "L": 13,
    "D": "Project . . ."
  },
  {
    "T": "I",
    "A": "SAREA39 ",
    "Y": 6,
    "X": 19,
    "C": "AQUA",
    "I": "LOW",
    "H": "USCORE",
    "L": 8,
    "N": "PRJ1",
    "D": ""
  },
  {
    "T": "T",
    "Y": 7,
    "X": 5,
    "C": "GREEN",
    "I": "LOW",
    "L": 13,
    "D": "Group . . . ."
  },
  {
    "T": "I",
    "A": "SAREA39 ",
    "Y": 7,
    "X": 19,
    "C": "AQUA",
    "I": "LOW",
    "H": "USCORE",
    "L": 8,
    "N": "LIB1",
    "D": ""
  },
  {
    "T": "T",
    "Y": 7,
    "X": 28,
    "C": "GREEN",
    "I": "LOW",
    "L": 5,
    "D": ". . ."
  },
  {
    "T": "I",
    "A": "SAREA39 ",
    "Y": 7,
    "X": 34,
    "C": "AQUA",
    "I": "LOW",
    "H": "USCORE",
    "L": 8,
    "N": "LIB2",
    "D": ""
  },
  {
    "T": "T",
    "Y": 7,
    "X": 43,
    "C": "GREEN",
    "I": "LOW",
    "L": 5,
    "D": ". . ."
  },
  {
    "T": "I",
    "A": "SAREA39 ",
    "Y": 7,
    "X": 49,

```

```

    "C": "AQUA",
    "I": "LOW",
    "H": "USCORE",
    "L": 8,
    "N": "LIB3",
    "D": ""
  },
  {
    "T": "T",
    "Y": 7,
    "X": 58,
    "C": "GREEN",
    "I": "LOW",
    "L": 5,
    "D": ". . ."
  },
  {
    "T": "I",
    "A": "SAREA39 ",
    "Y": 7,
    "X": 64,
    "C": "AQUA",
    "I": "LOW",
    "H": "USCORE",
    "L": 8,
    "N": "LIB4",
    "D": ""
  },
  {
    "T": "T",
    "Y": 8,
    "X": 5,
    "C": "GREEN",
    "I": "LOW",
    "L": 13,
    "D": "Type . . ."
  },
  {
    "T": "I",
    "A": "SAREA39 ",
    "Y": 8,
    "X": 19,
    "C": "AQUA",
    "I": "LOW",
    "H": "USCORE",
    "L": 8,
    "N": "TYP1",
    "D": ""
  },
  {
    "T": "T",
    "Y": 9,
    "X": 5,
    "C": "GREEN",
    "I": "LOW",
    "L": 13,
    "D": "Member . . ."
  },
  {
    "T": "I",
    "A": "SAREA39 ",
    "Y": 9,
    "X": 19,
    "C": "AQUA",
    "I": "LOW",
    "H": "USCORE",
    "L": 8,
    "N": "ZMEM",
    "D": ""
  },
  {
    "T": "T",
    "Y": 9,
    "X": 35,
    "C": "GREEN",
    "I": "LOW",
    "L": 44,
    "D": "(Blank or pattern for member selection list)"
  },
  {
    "T": "T",
    "Y": 11,

```

```

    "X":2,
    "C":"BLUE",
    "I":"HIGH",
    "L":66,
    "D":"Other Partitioned, Sequential or VSAM Data Set, or z/OS UNIX file:"
  },
  {
    "T":"T",
    "Y":12,
    "X":5,
    "C":"GREEN",
    "I":"LOW",
    "L":14,
    "D":"Name . . . ."
  },
  {
    "T":"I",
    "A":"SAREA39 ",
    "Y":12,
    "X":20,
    "C":"AQUA",
    "I":"LOW",
    "H":"USCORE",
    "L":56,
    "LX":1023,
    "N":"ZODSN",
    "D":""
  },
  {
    "T":"O",
    "A":"SAREA39 ",
    "Y":12,
    "X":78,
    "C":"WHITE",
    "I":"LOW",
    "L":2,
    "N":"ZODSIND",
    "D":" +"
  },
  {
    "T":"T",
    "Y":13,
    "X":5,
    "C":"GREEN",
    "I":"LOW",
    "L":17,
    "D":"Volume Serial . ."
  },
  {
    "T":"I",
    "A":"SAREA39 ",
    "Y":13,
    "X":23,
    "C":"AQUA",
    "I":"LOW",
    "H":"USCORE",
    "L":6,
    "N":"ZVOL",
    "D":""
  },
  {
    "T":"T",
    "Y":13,
    "X":33,
    "C":"GREEN",
    "I":"LOW",
    "L":18,
    "D":"(If not cataloged)"
  },
  {
    "T":"T",
    "A":"SAREA39 ",
    "Y":17,
    "X":42,
    "C":"BLUE",
    "I":"HIGH",
    "L":7,
    "D":"Options"
  },
  {
    "T":"T",
    "Y":18,

```



```

    "X":2,
    "C":"GREEN",
    "I":"LOW",
    "L":22,
    "D":"Initial Macro . . . ."
  },
  {
    "T":"I",
    "A":"SAREA39 ",
    "Y":18,
    "X":25,
    "C":"AQUA",
    "I":"LOW",
    "H":"USCORE",
    "L":8,
    "N":"ZVIMAC",
    "D":""
  },
  {
    "T":"I",
    "A":"SAREA39 ",
    "Y":18,
    "X":42,
    "C":"AQUA",
    "I":"LOW",
    "H":"USCORE",
    "L":1,
    "N":"ZPCFMCN",
    "D":"/",
    "SL":{
      "T":"CB",
      "V":"/"
    }
  },
  {
    "T":"T",
    "Y":18,
    "X":45,
    "C":"WHITE",
    "I":"LOW",
    "L":27,
    "D":"Confirm Cancel/Move/Replace"
  },
  {
    "T":"T",
    "Y":19,
    "X":2,
    "C":"GREEN",
    "I":"LOW",
    "L":22,
    "D":"Profile Name . . . . ."
  },
  {
    "T":"I",
    "A":"SAREA39 ",
    "Y":19,
    "X":25,
    "C":"AQUA",
    "I":"LOW",
    "H":"USCORE",
    "L":8,
    "N":"ZVPROF",
    "D":""
  },
  {
    "T":"I",
    "A":"SAREA39 ",
    "Y":19,
    "X":42,
    "C":"AQUA",
    "I":"LOW",
    "H":"USCORE",
    "L":1,
    "N":"VIEWM",
    "D":"",
    "SL":{
      "T":"CB",
      "V":"/"
    }
  },
  {
    "T":"T",

```

```

      "Y":19,
      "X":45,
      "C":"WHITE",
      "I":"LOW",
      "L":11,
      "D":"Browse Mode"
    },
    {
      "T":"T",
      "Y":20,
      "X":2,
      "C":"GREEN",
      "I":"LOW",
      "L":22,
      "D":"Format Name . . . ."
    },
    {
      "T":"I",
      "A":"SAREA39 ",
      "Y":20,
      "X":25,
      "C":"AQUA",
      "I":"LOW",
      "H":"USCORE",
      "L":8,
      "N":"FNAM",
      "D":""
    },
    {
      "T":"T",
      "Y":21,
      "X":2,
      "C":"GREEN",
      "I":"LOW",
      "L":22,
      "D":"Data Set Password . ."
    },
    {
      "T":"I",
      "A":"SAREA39 ",
      "Y":21,
      "X":25,
      "C":"AQUA",
      "I":"NON",
      "H":"USCORE",
      "L":8,
      "N":"PSWD",
      "D":""
    },
    {
      "T":"I",
      "A":"SAREA39 ",
      "Y":21,
      "X":42,
      "C":"AQUA",
      "I":"LOW",
      "H":"USCORE",
      "L":1,
      "N":"ZVWARN",
      "D":"/",
      "SL":{
        "T":"CB",
        "V":"/"
      }
    },
    {
      "T":"T",
      "Y":21,
      "X":45,
      "C":"WHITE",
      "I":"LOW",
      "L":25,
      "D":"Warn on First Data Change"
    },
    {
      "T":"T",
      "Y":22,
      "X":2,
      "C":"GREEN",
      "I":"LOW",
      "L":22,
      "D":"Record Length . . . ."
    }
  ]
}

```

```

    },
    {
      "T": "I",
      "A": "SAREA39 ",
      "Y": 22,
      "X": 25,
      "C": "AQUA",
      "I": "LOW",
      "H": "USCORE",
      "L": 5,
      "N": "ZBRECL",
      "D": ""
    },
    {
      "T": "I",
      "A": "SAREA39 ",
      "Y": 22,
      "X": 42,
      "C": "AQUA",
      "I": "LOW",
      "H": "USCORE",
      "L": 1,
      "N": "MIXM",
      "D": "",
      "SL": {
        "T": "CB",
        "V": "/"
      }
    },
    {
      "T": "T",
      "Y": 22,
      "X": 45,
      "C": "WHITE",
      "I": "LOW",
      "L": 10,
      "D": "Mixed Mode"
    },
    {
      "T": "T",
      "Y": 23,
      "X": 2,
      "C": "GREEN",
      "I": "LOW",
      "L": 22,
      "D": "Line Command Table . ."
    },
    {
      "T": "I",
      "A": "SAREA39 ",
      "Y": 23,
      "X": 25,
      "C": "AQUA",
      "I": "LOW",
      "H": "USCORE",
      "L": 8,
      "N": "ZLMAC",
      "D": ""
    },
    {
      "T": "I",
      "A": "SAREA39 ",
      "Y": 23,
      "X": 42,
      "C": "AQUA",
      "I": "LOW",
      "H": "USCORE",
      "L": 1,
      "N": "ZEDASC",
      "D": "",
      "SL": {
        "T": "CB",
        "V": "/"
      }
    },
    {
      "T": "T",
      "Y": 23,
      "X": 45,
      "C": "WHITE",
      "I": "LOW",
      "L": 15,

```

## JSON data structures sent from ISPF to client (message type 3)

```
{
  "D": "View ASCII data"
},
{
  "T": "T",
  "Y": 24,
  "X": 2,
  "C": "GREEN",
  "I": "LOW",
  "L": 12,
  "D": "Command ==>"
},
{
  "T": "I",
  "Z": "TRUE",
  "Y": 24,
  "X": 15,
  "C": "AQUA",
  "I": "LOW",
  "H": "USCORE",
  "L": 65,
  "N": "ZCMD",
  "D": ""
}
],
"KEY": [{
  "K": "ENTER",
  "N": "ENTER"
},
{
  "K": "1",
  "N": "Help"
},
{
  "K": "2",
  "N": "Split"
},
{
  "K": "3",
  "N": "Exit"
},
{
  "K": "7",
  "N": "Backward"
},
{
  "K": "8",
  "N": "Forward"
},
{
  "K": "9",
  "N": "Swap"
},
{
  "K": "10",
  "N": "Actions"
},
{
  "K": "12",
  "N": "Cancel"
}
]
}
```

Here is an example of the JSON for a view display of member ACB in data set 'SYS1.MACLIB'. Language sensitive coloring is enabled through the edit HILITE command:

```
{
  "PNL": {
    "VER": "0100",
    "NME": "ISREDDE2",
    "SCR": "1",
    "SCN": "VIEW",
    "HDL": "001806467788",
    "RWS": 24,
    "CLS": 80,
    "EDT": "TRUE",
    "CML": "BOTTOM",
    "CUR": {
      "ROW": 24,

```

```

      "CLM":15
    },
    "SUP":"TRUE",
    "SDN":"TRUE",
    "SRG":"TRUE",
    "IFY":{
      "SYS":"","
      "UID":""
    },
    "ARE":[{
      "NME":"ZDATA",
      "TYP":"D",
      "TOP":4,
      "BOT":23,
      "LFT":1,
      "RGT":80
    }
  ],
  "CHA":[{
    "COL":"PINK",
    "HIL":"RVIDEO"
  },
  {
    "COL":"RED"
  },
  {
    "COL":"GREEN"
  },
  {
    "COL":"BLUE"
  },
  {
    "COL":"WHITE"
  },
  {
    "COL":"PINK"
  },
  {
    "COL":"YELLOW"
  },
  {
    "COL":"AQUA"
  },
  {
    "COL":"RED"
  },
  {
    "HIL":"USCORE"
  },
  {
    "COL":"RED"
  },
  {
    "COL":"YELLOW"
  },
  {
    "COL":"WHITE"
  },
  {
    "COL":"AQUA"
  },
  {
    "COL":"GREEN"
  },
  {
    "COL":"GREEN"
  },
  {
    "COL":"WHITE",
    "HIL":"RVIDEO"
  },
  {
    "COL":"WHITE"
  }
],
  "MNU":[{
    "PUL":" File",
    "CHS":[{
      "N":"Save",
      "I":"1-1",
      "D":"TRUE"
    }
  ]
},

```

```

        {
            "N": "Cancel",
            "I": "1-2"
        },
        {
            "N": "Exit",
            "I": "1-3"
        }
    ]
},
{
    "PUL": " Edit",
    "CHS": [
        {
            "N": "Reset",
            "I": "2-1"
        },
        {
            "N": "Undo",
            "I": "2-2"
        },
        {
            "N": "Hilite",
            "I": "2-3"
        },
        {
            "N": "Cut",
            "I": "2-4"
        },
        {
            "N": "Paste",
            "I": "2-5"
        }
    ]
},
{
    "PUL": " Edit_Settings",
    "CHS": [
        {
            "N": "Edit settings",
            "I": "3-1"
        }
    ]
},
{
    "PUL": " Menu",
    "CHS": [
        {
            "N": "Settings",
            "I": "4-1"
        },
        {
            "N": "View",
            "I": "4-2"
        },
        {
            "N": "Edit",
            "I": "4-3"
        },
        {
            "N": "ISPF Command Shell",
            "I": "4-4"
        },
        {
            "N": "Dialog Test...",
            "I": "4-5"
        },
        {
            "N": "Other IBM Products...",
            "I": "4-6"
        },
        {
            "N": "SCLM",
            "I": "4-7"
        },
        {
            "N": "ISPF Workplace",
            "I": "4-8"
        },
        {
            "N": "Status Area...",
            "I": "4-9"
        },
        {
            "N": "Exit",

```

```

        "I": "4-10"
      }
    ],
    "PUL": "Utilities",
    "CHS": [
      {
        "N": "Library",
        "I": "5-1"
      },
      {
        "N": "Data set",
        "I": "5-2"
      },
      {
        "N": "Move/Copy",
        "I": "5-3"
      },
      {
        "N": "Data Set List",
        "I": "5-4"
      },
      {
        "N": "Reset Statistics",
        "I": "5-5"
      },
      {
        "N": "Hardcopy",
        "I": "5-6"
      },
      {
        "N": "Reserved",
        "I": "5-7",
        "D": "TRUE"
      },
      {
        "N": "Outlist",
        "I": "5-8"
      },
      {
        "N": "Commands...",
        "I": "5-9"
      },
      {
        "N": "Reserved",
        "I": "5-10",
        "D": "TRUE"
      },
      {
        "N": "Format",
        "I": "5-11"
      },
      {
        "N": "SuperC",
        "I": "5-12"
      },
      {
        "N": "SuperCE",
        "I": "5-13"
      },
      {
        "N": "Search-For",
        "I": "5-14"
      },
      {
        "N": "Search-ForE",
        "I": "5-15"
      },
      {
        "N": "Table Utility",
        "I": "5-16"
      },
      {
        "N": "Directory List",
        "I": "5-17"
      }
    ]
  },
  "PUL": "Compilers",
  "CHS": [
    {
      "N": "Foreground Compilers",

```

```

        "I": "6-1"
      },
      {
        "N": "Background Compilers",
        "I": "6-2"
      },
      {
        "N": "ISPPREP Panel Utility...",
        "I": "6-3"
      },
      {
        "N": "DTL Compiler...",
        "I": "6-4"
      }
    ]
  },
  {
    "PUL": " Test",
    "CHS": [
      {
        "N": "Functions...",
        "I": "7-1"
      },
      {
        "N": "Panels...",
        "I": "7-2"
      },
      {
        "N": "Variables...",
        "I": "7-3"
      },
      {
        "N": "Tables...",
        "I": "7-4"
      },
      {
        "N": "Log",
        "I": "7-5"
      },
      {
        "N": "Services...",
        "I": "7-6"
      },
      {
        "N": "Traces...",
        "I": "7-7"
      },
      {
        "N": "Break Points...",
        "I": "7-8"
      },
      {
        "N": "Dialog Test...",
        "I": "7-9"
      },
      {
        "N": "Dialog Test appl ID...",
        "I": "7-10"
      }
    ]
  },
  {
    "PUL": " Help",
    "CHS": [
      {
        "N": "General",
        "I": "8-1"
      },
      {
        "N": "Display screen format",
        "I": "8-2"
      },
      {
        "N": "Scrolling data",
        "I": "8-3"
      },
      {
        "N": "Sequence numbering",
        "I": "8-4"
      },
      {
        "N": "Display modes",
        "I": "8-5"
      }
    ]
  }
]

```



```

    {
      "N": "Tabbing",
      "I": "8-6"
    },
    {
      "N": "Automatic recovery",
      "I": "8-7"
    },
    {
      "N": "Edit profiles",
      "I": "8-8"
    },
    {
      "N": "Edit line commands",
      "I": "8-9"
    },
    {
      "N": "Edit primary commands",
      "I": "8-10"
    },
    {
      "N": "Labels and line ranges",
      "I": "8-11"
    },
    {
      "N": "Ending an edit session",
      "I": "8-12"
    },
    {
      "N": "Appendices",
      "I": "8-13"
    },
    {
      "N": "Index",
      "I": "8-14"
    }
  ]
},
"FLD": [
  {
    "T": "0",
    "Y": 3,
    "X": 2,
    "C": "AQUA",
    "I": "LOW",
    "L": 10,
    "N": "ZVMODET",
    "D": "VIEW"
  },
  {
    "T": "0",
    "Y": 3,
    "X": 13,
    "C": "AQUA",
    "I": "LOW",
    "L": 47,
    "N": "ZTITLE",
    "D": "SYS1.MACLIB(ACB) - 01.00"
  },
  {
    "T": "T",
    "Y": 3,
    "X": 61,
    "C": "GREEN",
    "I": "LOW",
    "L": 7,
    "D": "Columns"
  },
  {
    "T": "0",
    "Y": 3,
    "X": 69,
    "C": "AQUA",
    "I": "LOW",
    "L": 5,
    "N": "ZCL",
    "D": "00001"
  },
  {
    "T": "0",
    "Y": 3,
    "X": 75,

```

## JSON data structures sent from ISPF to client (message type 3)

```

    "C": "AQUA",
    "I": "LOW",
    "L": 5,
    "N": "ZCR",
    "D": "00072"
  },
  {
    "T": "I",
    "A": "ZDATA  ",
    "Y": 4,
    "X": 2,
    "C": "BLUE",
    "I": "LOW",
    "L": 6,
    "N": "ZDATA_0001",
    "D": "010833"
  },
  {
    "T": "I",
    "A": "ZDATA  ",
    "Y": 4,
    "X": 9,
    "C": "YELLOW",
    "I": "LOW",
    "L": 72,
    "N": "ZDATA_0002",
    "D": ". * $LQ=DFSMS,HDZ11C0,043093,SJPLRG: VSAM RLS @LQA ",
    "SHS": [
      {
        "STA": 1,
        "LEN": 2,
        "ATT": 14
      },
      {
        "STA": 4,
        "LEN": 32,
        "ATT": 14
      },
      {
        "STA": 37,
        "LEN": 4,
        "ATT": 14
      },
      {
        "STA": 42,
        "LEN": 3,
        "ATT": 14
      },
      {
        "STA": 68,
        "LEN": 4,
        "ATT": 14
      }
    ]
  },
  {
    "T": "I",
    "A": "ZDATA  ",
    "Y": 5,
    "X": 2,
    "C": "BLUE",
    "I": "LOW",
    "L": 6,
    "N": "ZDATA_0003",
    "D": "010999"
  },
  {
    "T": "I",
    "A": "ZDATA  ",
    "Y": 5,
    "X": 9,
    "C": "YELLOW",
    "I": "LOW",
    "L": 72,
    "N": "ZDATA_0004",
    "D": ". * $VC= DFSMS HDZ11F0 10/26/99 SJPLJRB : REMOVE VSAM CTLG SUPP @VCA ",
    "SHS": [
      {
        "STA": 1,
        "LEN": 2,
        "ATT": 14
      },
      {
        "STA": 4,

```

```

        "LEN":4,
        "ATT":14
      },
      {
        "STA":9,
        "LEN":5,
        "ATT":14
      },
      {
        "STA":16,
        "LEN":7,
        "ATT":14
      },
      {
        "STA":24,
        "LEN":8,
        "ATT":14
      },
      {
        "STA":33,
        "LEN":7,
        "ATT":14
      },
      {
        "STA":42,
        "LEN":1,
        "ATT":14
      },
      {
        "STA":44,
        "LEN":6,
        "ATT":14
      },
      {
        "STA":51,
        "LEN":4,
        "ATT":14
      },
      {
        "STA":56,
        "LEN":4,
        "ATT":14
      },
      {
        "STA":61,
        "LEN":4,
        "ATT":14
      },
      {
        "STA":68,
        "LEN":4,
        "ATT":14
      }
    ]
  },
  {
    "T":"I",
    "A":"ZDATA  ",
    "Y":6,
    "X":2,
    "C":"BLUE",
    "I":"LOW",
    "L":6,
    "N":"ZDATA_0005",
    "D":"011166"
  },
  {
    "T":"I",
    "A":"ZDATA  ",
    "Y":6,
    "X":9,
    "C":"YELLOW",
    "I":"LOW",
    "L":72,
    "N":"ZDATA_0006",
    "D":"*****@LQC  ",
    "SHS":[{
      "STA":1,
      "LEN":71,
      "ATT":14
    }
  ]
}

```

```

    },
    {
      "T": "I",
      "A": "ZDATA  ",
      "Y": 7,
      "X": 2,
      "C": "BLUE",
      "I": "LOW",
      "L": 6,
      "N": "ZDATA_0007",
      "D": "011500"
    },
    {
      "T": "I",
      "A": "ZDATA  ",
      "Y": 7,
      "X": 9,
      "C": "YELLOW",
      "I": "LOW",
      "L": 72,
      "N": "ZDATA_0008",
      "D": "AIF ('&AM' EQ 'VTAM').VTACB IS IT VTAM",
      "SHS": [
        {
          "STA": 10,
          "LEN": 3,
          "ATT": 11
        },
        {
          "STA": 16,
          "LEN": 1,
          "ATT": 16
        },
        {
          "STA": 17,
          "LEN": 5,
          "ATT": 13
        },
        {
          "STA": 23,
          "LEN": 2,
          "ATT": 16
        },
        {
          "STA": 26,
          "LEN": 6,
          "ATT": 13
        },
        {
          "STA": 32,
          "LEN": 7,
          "ATT": 16
        },
        {
          "STA": 41,
          "LEN": 2,
          "ATT": 14
        },
        {
          "STA": 44,
          "LEN": 2,
          "ATT": 14
        },
        {
          "STA": 47,
          "LEN": 4,
          "ATT": 14
        }
      ]
    },
    {
      "T": "I",
      "A": "ZDATA  ",
      "Y": 8,
      "X": 2,
      "C": "BLUE",
      "I": "LOW",
      "L": 6,
      "N": "ZDATA_0009",
      "D": "012000"
    },
    {
      "T": "I",

```

```

    "A": "ZDATA  ",
    "Y": 8,
    "X": 9,
    "C": "YELLOW",
    "I": "LOW",
    "L": 72,
    "N": "ZDATA_0010",
    "D": "          AIF      ('&AM' EQ 'VSAM').VSACB  IS IT VSAM      ",
    "SHS": [{
      "STA": 10,
      "LEN": 3,
      "ATT": 11
    },
    {
      "STA": 16,
      "LEN": 1,
      "ATT": 16
    },
    {
      "STA": 17,
      "LEN": 5,
      "ATT": 13
    },
    {
      "STA": 23,
      "LEN": 2,
      "ATT": 16
    },
    {
      "STA": 26,
      "LEN": 6,
      "ATT": 13
    },
    {
      "STA": 32,
      "LEN": 7,
      "ATT": 16
    },
    {
      "STA": 41,
      "LEN": 2,
      "ATT": 14
    },
    {
      "STA": 44,
      "LEN": 2,
      "ATT": 14
    },
    {
      "STA": 47,
      "LEN": 4,
      "ATT": 14
    }
    ]
  },
  {
    "T": "I",
    "A": "ZDATA  ",
    "Y": 9,
    "X": 2,
    "C": "BLUE",
    "I": "LOW",
    "L": 6,
    "N": "ZDATA_0011",
    "D": "012500"
  },
  {
    "T": "I",
    "A": "ZDATA  ",
    "Y": 9,
    "X": 9,
    "C": "YELLOW",
    "I": "LOW",
    "L": 72,
    "N": "ZDATA_0012",
    "D": "          AIF      ('&AM' EQ ' ').VSACB      IF NULL, DEFAULT VSAM      ",
    "SHS": [{
      "STA": 10,
      "LEN": 3,
      "ATT": 11
    },
    {

```

## JSON data structures sent from ISPF to client (message type 3)

```
        "STA":16,
        "LEN":1,
        "ATT":16
      },
      {
        "STA":17,
        "LEN":5,
        "ATT":13
      },
      {
        "STA":23,
        "LEN":2,
        "ATT":16
      },
      {
        "STA":26,
        "LEN":2,
        "ATT":13
      },
      {
        "STA":28,
        "LEN":7,
        "ATT":16
      },
      {
        "STA":41,
        "LEN":2,
        "ATT":14
      },
      {
        "STA":44,
        "LEN":5,
        "ATT":14
      },
      {
        "STA":50,
        "LEN":7,
        "ATT":14
      },
      {
        "STA":58,
        "LEN":4,
        "ATT":14
      }
    ]
  },
  {
    "T":"I",
    "A":"ZDATA  ",
    "Y":10,
    "X":2,
    "C":"BLUE",
    "I":"LOW",
    "L":6,
    "N":"ZDATA_0013",
    "D":"013000"
  },
  {
    "T":"I",
    "A":"ZDATA  ",
    "Y":10,
    "X":9,
    "C":"YELLOW",
    "I":"LOW",
    "L":72,
    "N":"ZDATA_0014",
    "D":".*",
    "SHS":[{
      "STA":1,
      "LEN":2,
      "ATT":14
    }
  ]
},
{
  "T":"I",
  "A":"ZDATA  ",
  "Y":11,
  "X":2,
  "C":"BLUE",
  "I":"LOW",
  "L":6,
```

```

    "N": "ZDATA_0015",
    "D": "013500"
  },
  {
    "T": "I",
    "A": "ZDATA  ",
    "Y": 11,
    "X": 9,
    "C": "YELLOW",
    "I": "LOW",
    "L": 72,
    "N": "ZDATA_0016",
    "D": "          IDAERMAC  3,AM,&AM          ISSUE ERROR MSG          @L9C  ",
    "SHS": [
      {
        "STA": 10,
        "LEN": 8,
        "ATT": 11
      },
      {
        "STA": 20,
        "LEN": 1,
        "ATT": 16
      },
      {
        "STA": 22,
        "LEN": 2,
        "ATT": 16
      },
      {
        "STA": 26,
        "LEN": 2,
        "ATT": 16
      },
      {
        "STA": 41,
        "LEN": 5,
        "ATT": 14
      },
      {
        "STA": 47,
        "LEN": 5,
        "ATT": 14
      },
      {
        "STA": 53,
        "LEN": 3,
        "ATT": 14
      },
      {
        "STA": 68,
        "LEN": 4,
        "ATT": 14
      }
    ]
  },
  {
    "T": "I",
    "A": "ZDATA  ",
    "Y": 12,
    "X": 2,
    "C": "BLUE",
    "I": "LOW",
    "L": 6,
    "N": "ZDATA_0017",
    "D": "014000"
  },
  {
    "T": "I",
    "A": "ZDATA  ",
    "Y": 12,
    "X": 9,
    "C": "YELLOW",
    "I": "LOW",
    "L": 72,
    "N": "ZDATA_0018",
    "D": "          MEXIT          ",
    "SHS": [
      {
        "STA": 10,
        "LEN": 5,
        "ATT": 11
      }
    ]
  }
]

```

## JSON data structures sent from ISPF to client (message type 3)

```

    },
    {
      "T": "I",
      "A": "ZDATA  ",
      "Y": 13,
      "X": 2,
      "C": "BLUE",
      "I": "LOW",
      "L": 6,
      "N": "ZDATA_0019",
      "D": "014500"
    },
    {
      "T": "I",
      "A": "ZDATA  ",
      "Y": 13,
      "X": 9,
      "C": "YELLOW",
      "I": "LOW",
      "L": 72,
      "N": "ZDATA_0020",
      "D": ".*",
      "SHS": [
        {
          "STA": 1,
          "LEN": 2,
          "ATT": 14
        }
      ]
    },
    {
      "T": "I",
      "A": "ZDATA  ",
      "Y": 14,
      "X": 2,
      "C": "BLUE",
      "I": "LOW",
      "L": 6,
      "N": "ZDATA_0021",
      "D": "015000"
    },
    {
      "T": "I",
      "A": "ZDATA  ",
      "Y": 14,
      "X": 9,
      "C": "YELLOW",
      "I": "LOW",
      "L": 72,
      "N": "ZDATA_0022",
      "D": ".VTACB  ANOP",
      "SHS": [
        {
          "STA": 1,
          "LEN": 6,
          "ATT": 16
        },
        {
          "STA": 10,
          "LEN": 4,
          "ATT": 11
        }
      ]
    },
    {
      "T": "I",
      "A": "ZDATA  ",
      "Y": 15,
      "X": 2,
      "C": "BLUE",
      "I": "LOW",
      "L": 6,
      "N": "ZDATA_0023",
      "D": "015500"
    },
    {
      "T": "I",
      "A": "ZDATA  ",
      "Y": 15,
      "X": 9,
      "C": "YELLOW",
      "I": "LOW",
      "L": 72,
      "N": "ZDATA_0024",

```



```

"D": "&:NAME      ISTACB1 DDNAME=&:DDNAME,EXLST=&:EXLST,          - ",
"SHS": [{
  "STA": 1,
  "LEN": 5,
  "ATT": 16
},
{
  "STA": 10,
  "LEN": 7,
  "ATT": 11
},
{
  "STA": 18,
  "LEN": 6,
  "ATT": 16
},
{
  "STA": 26,
  "LEN": 6,
  "ATT": 16
},
{
  "STA": 33,
  "LEN": 5,
  "ATT": 16
},
{
  "STA": 40,
  "LEN": 5,
  "ATT": 16
},
{
  "STA": 72,
  "LEN": 1,
  "ATT": 14
}
]
},
{
  "T": "I",
  "A": "ZDATA  ",
  "Y": 16,
  "X": 2,
  "C": "BLUE",
  "I": "LOW",
  "L": 6,
  "N": "ZDATA_0025",
  "D": "016000"
},
{
  "T": "I",
  "A": "ZDATA  ",
  "Y": 16,
  "X": 9,
  "C": "YELLOW",
  "I": "LOW",
  "L": 72,
  "N": "ZDATA_0026",
  "D": "MACRF=&:MACRF,JFCB=&:JFCB,BUFND=&:BUFND,          - ",
  "SHS": [{
    "STA": 16,
    "LEN": 5,
    "ATT": 16
  },
  {
    "STA": 23,
    "LEN": 5,
    "ATT": 16
  },
  {
    "STA": 29,
    "LEN": 4,
    "ATT": 16
  },
  {
    "STA": 35,
    "LEN": 4,
    "ATT": 16
  },
  {
    "STA": 40,
    "LEN": 5,
  }
}

```

```

        "ATT":16
      },
      {
        "STA":47,
        "LEN":5,
        "ATT":16
      },
      {
        "STA":72,
        "LEN":1,
        "ATT":14
      }
    ]
  },
  {
    "T":"I",
    "A":"ZDATA  ",
    "Y":17,
    "X":2,
    "C":"BLUE",
    "I":"LOW",
    "L":6,
    "N":"ZDATA_0027",
    "D":"016500"
  },
  {
    "T":"I",
    "A":"ZDATA  ",
    "Y":17,
    "X":9,
    "C":"YELLOW",
    "I":"LOW",
    "L":72,
    "N":"ZDATA_0028",
    "D":"",
    "SHS":[{
      "STA":16,
      "LEN":5,
      "ATT":16
    },
    {
      "STA":23,
      "LEN":5,
      "ATT":16
    },
    {
      "STA":29,
      "LEN":6,
      "ATT":16
    },
    {
      "STA":37,
      "LEN":6,
      "ATT":16
    },
    {
      "STA":72,
      "LEN":1,
      "ATT":14
    }
  ]
},
{
  "T":"I",
  "A":"ZDATA  ",
  "Y":18,
  "X":2,
  "C":"BLUE",
  "I":"LOW",
  "L":6,
  "N":"ZDATA_0029",
  "D":"017000"
},
{
  "T":"I",
  "A":"ZDATA  ",
  "Y":18,
  "X":9,
  "C":"YELLOW",
  "I":"LOW",
  "L":72,
  "N":"ZDATA_0030",
  "D":"",
  "SHS":&: BUFNI= &: BUFNI , PASSWD= &: PASSWD,
  "I":&:
}

```

```

"D": "
"SHS": [{
  "STA": 16,
  "LEN": 7,
  "ATT": 16
},
{
  "STA": 25,
  "LEN": 7,
  "ATT": 16
},
{
  "STA": 33,
  "LEN": 5,
  "ATT": 16
},
{
  "STA": 40,
  "LEN": 5,
  "ATT": 16
},
{
  "STA": 72,
  "LEN": 1,
  "ATT": 14
}
]
},
{
  "T": "I",
  "A": "ZDATA ",
  "Y": 19,
  "X": 2,
  "C": "BLUE",
  "I": "LOW",
  "L": 6,
  "N": "ZDATA_0031",
  "D": "017500"
},
{
  "T": "I",
  "A": "ZDATA ",
  "Y": 19,
  "X": 9,
  "C": "YELLOW",
  "I": "LOW",
  "L": 72,
  "N": "ZDATA_0032",
  "D": "
"SHS": [{
  "STA": 16,
  "LEN": 5,
  "ATT": 16
},
{
  "STA": 23,
  "LEN": 5,
  "ATT": 16
},
{
  "STA": 29,
  "LEN": 6,
  "ATT": 16
},
{
  "STA": 37,
  "LEN": 6,
  "ATT": 16
},
{
  "STA": 72,
  "LEN": 1,
  "ATT": 14
}
]
},
{
  "T": "I",
  "A": "ZDATA ",
  "Y": 20,
  "X": 2,
  "C": "BLUE",

```

## JSON data structures sent from ISPF to client (message type 3)

```

    "I": "LOW",
    "L": 6,
    "N": "ZDATA_0033",
    "D": "018000"
  },
  {
    "T": "I",
    "A": "ZDATA  ",
    "Y": 20,
    "X": 9,
    "C": "YELLOW",
    "I": "LOW",
    "L": 72,
    "N": "ZDATA_0034",
    "D": "STRNO=&:STRNO,CATALOG=&:CATALOG, - ",
    "SHS": [
      {
        "STA": 16,
        "LEN": 5,
        "ATT": 16
      },
      {
        "STA": 23,
        "LEN": 5,
        "ATT": 16
      },
      {
        "STA": 29,
        "LEN": 7,
        "ATT": 16
      },
      {
        "STA": 38,
        "LEN": 7,
        "ATT": 16
      },
      {
        "STA": 72,
        "LEN": 1,
        "ATT": 14
      }
    ]
  },
  {
    "T": "I",
    "A": "ZDATA  ",
    "Y": 21,
    "X": 2,
    "C": "BLUE",
    "I": "LOW",
    "L": 6,
    "N": "ZDATA_0035",
    "D": "018500"
  },
  {
    "T": "I",
    "A": "ZDATA  ",
    "Y": 21,
    "X": 9,
    "C": "YELLOW",
    "I": "LOW",
    "L": 72,
    "N": "ZDATA_0036",
    "D": "BSTRNO=&:BSTRNO,MAREA=&:MAREA,MLEN=&:MLEN, - ",
    "SHS": [
      {
        "STA": 16,
        "LEN": 6,
        "ATT": 16
      },
      {
        "STA": 24,
        "LEN": 6,
        "ATT": 16
      },
      {
        "STA": 31,
        "LEN": 5,
        "ATT": 16
      },
      {
        "STA": 38,
        "LEN": 5,
        "ATT": 16
      }
    ]
  }
]

```

```

    },
    {
      "STA": 44,
      "LEN": 4,
      "ATT": 16
    },
    {
      "STA": 50,
      "LEN": 4,
      "ATT": 16
    },
    {
      "STA": 72,
      "LEN": 1,
      "ATT": 14
    }
  ]
},
{
  "T": "I",
  "A": "ZDATA  ",
  "Y": 22,
  "X": 2,
  "C": "BLUE",
  "I": "LOW",
  "L": 6,
  "N": "ZDATA_0037",
  "D": "019000"
},
{
  "T": "I",
  "A": "ZDATA  ",
  "Y": 22,
  "X": 9,
  "C": "YELLOW",
  "I": "LOW",
  "L": 72,
  "N": "ZDATA_0038",
  "D": "",
  "SHS": [
    {
      "STA": 16,
      "LEN": 3,
      "ATT": 16
    },
    {
      "STA": 21,
      "LEN": 3,
      "ATT": 16
    },
    {
      "STA": 25,
      "LEN": 2,
      "ATT": 16
    },
    {
      "STA": 29,
      "LEN": 2,
      "ATT": 16
    },
    {
      "STA": 32,
      "LEN": 6,
      "ATT": 16
    },
    {
      "STA": 40,
      "LEN": 6,
      "ATT": 16
    },
    {
      "STA": 72,
      "LEN": 1,
      "ATT": 14
    }
  ]
},
{
  "T": "I",
  "A": "ZDATA  ",
  "Y": 23,
  "X": 2,
  "C": "BLUE",

```

```

    "I": "LOW",
    "L": 6,
    "N": "ZDATA_0039",
    "D": "019500"
  },
  {
    "T": "I",
    "A": "ZDATA  ",
    "Y": 23,
    "X": 9,
    "C": "YELLOW",
    "I": "LOW",
    "L": 72,
    "N": "ZDATA_0040",
    "D": "          USERPTR=&:USERPTR,          - ",
    "SHS": [
      {
        "STA": 16,
        "LEN": 7,
        "ATT": 16
      },
      {
        "STA": 25,
        "LEN": 7,
        "ATT": 16
      },
      {
        "STA": 72,
        "LEN": 1,
        "ATT": 14
      }
    ]
  },
  {
    "T": "T",
    "Y": 24,
    "X": 2,
    "C": "GREEN",
    "I": "LOW",
    "L": 12,
    "D": "Command ==>"
  },
  {
    "T": "I",
    "Z": "TRUE",
    "Y": 24,
    "X": 15,
    "C": "AQUA",
    "I": "LOW",
    "H": "USCORE",
    "L": 48,
    "N": "ZCMD",
    "D": ""
  },
  {
    "T": "T",
    "Y": 24,
    "X": 64,
    "C": "GREEN",
    "I": "LOW",
    "L": 11,
    "D": "Scroll ==>"
  },
  {
    "T": "I",
    "Y": 24,
    "X": 76,
    "C": "AQUA",
    "I": "LOW",
    "H": "USCORE",
    "L": 4,
    "N": "ZSCED",
    "D": "PAGE"
  }
],
"KEY": [
  {
    "K": "ENTER",
    "N": "ENTER"
  },
  {
    "K": "1",
    "N": "Help"
  }
]

```

```

{
  "K": "2",
  "N": "Split"
},
{
  "K": "3",
  "N": "Exit"
},
{
  "K": "5",
  "N": "Rfind"
},
{
  "K": "6",
  "N": "Rchange"
},
{
  "K": "7",
  "N": "Up"
},
{
  "K": "8",
  "N": "Down"
},
{
  "K": "9",
  "N": "Swap"
},
{
  "K": "10",
  "N": "Left"
},
{
  "K": "11",
  "N": "Right"
},
{
  "K": "12",
  "N": "Cancel"
}
]
}

```

Here is an example of the JSON for a data set list display when the attribute view is selected and SYS1.DGT\* is specified for the Dsname Level:

```

{
  "PNL": {
    "VER": "0100",
    "NME": "ISRUDSL0",
    "SCR": "1",
    "SCN": "DSLST",
    "HDL": "000422895556",
    "RWS": 24,
    "CLS": 80,
    "CML": "BOTTOM",
    "CUR": {
      "ROW": 24,
      "CLM": 15
    },
    "SLF": "TRUE",
    "SRG": "TRUE",
    "IFY": {
      "SYS": "",
      "UID": ""
    }
  },
  "ARE": [
    {
      "NME": "ZDATA",
      "TYP": "D",
      "TOP": 5,
      "BOT": 23,
      "LFT": 1,
      "RGT": 80
    }
  ],
  "MNU": [
    {
      "PUL": " Menu",
      "CHS": [
        {
          "N": "Settings",

```

```

        "I": "1-1"
      },
      {
        "N": "View",
        "I": "1-2"
      },
      {
        "N": "Edit",
        "I": "1-3"
      },
      {
        "N": "ISPF Command Shell",
        "I": "1-4"
      },
      {
        "N": "Dialog Test...",
        "I": "1-5"
      },
      {
        "N": "Other IBM Products...",
        "I": "1-6"
      },
      {
        "N": "SCLM",
        "I": "1-7"
      },
      {
        "N": "ISPF Workplace",
        "I": "1-8"
      },
      {
        "N": "Status Area...",
        "I": "1-9"
      },
      {
        "N": "Exit",
        "I": "1-10"
      }
    ]
  },
  {
    "PUL": " Options",
    "CHS": [
      {
        "N": "DSLST Settings...",
        "I": "2-1"
      },
      {
        "N": "Refresh List",
        "I": "2-2"
      },
      {
        "N": "Append to List...",
        "I": "2-3"
      },
      {
        "N": "Save List",
        "I": "2-4"
      },
      {
        "N": "Reset",
        "I": "2-5"
      }
    ]
  },
  {
    "PUL": " View",
    "CHS": [
      {
        "N": "Volume",
        "I": "3-1"
      },
      {
        "N": "Space",
        "I": "3-2"
      },
      {
        "N": "Attributes",
        "I": "3-3",
        "D": "TRUE"
      },
      {
        "N": "Total",
        "I": "3-4"
      }
    ]
  }
]

```



```

    },
    {
      "N": "Sort...",
      "I": "3-5"
    }
  ]
},
{
  "PUL": "Utilities",
  "CHS": [
    {
      "N": "Library",
      "I": "4-1"
    },
    {
      "N": "Data set",
      "I": "4-2"
    },
    {
      "N": "Move/Copy",
      "I": "4-3"
    },
    {
      "N": "Data Set List",
      "I": "4-4"
    },
    {
      "N": "Reset Statistics",
      "I": "4-5"
    },
    {
      "N": "Hardcopy",
      "I": "4-6"
    },
    {
      "N": "Reserved",
      "I": "4-7",
      "D": "TRUE"
    },
    {
      "N": "Outlist",
      "I": "4-8"
    },
    {
      "N": "Commands...",
      "I": "4-9"
    },
    {
      "N": "Reserved",
      "I": "4-10",
      "D": "TRUE"
    },
    {
      "N": "Format",
      "I": "4-11"
    },
    {
      "N": "SuperC",
      "I": "4-12"
    },
    {
      "N": "SuperCE",
      "I": "4-13"
    },
    {
      "N": "Search-For",
      "I": "4-14"
    },
    {
      "N": "Search-ForE",
      "I": "4-15"
    },
    {
      "N": "Table Utility",
      "I": "4-16"
    },
    {
      "N": "Directory List",
      "I": "4-17"
    }
  ]
}
},

```

## JSON data structures sent from ISPF to client (message type 3)

```
{
  "PUL": "Compilers",
  "CHS": [
    {
      "N": "Foreground Compilers",
      "I": "5-1"
    },
    {
      "N": "Background Compilers",
      "I": "5-2"
    },
    {
      "N": "ISPPREP Panel Utility...",
      "I": "5-3"
    },
    {
      "N": "DTL Compiler...",
      "I": "5-4"
    }
  ]
},
{
  "PUL": "Help",
  "CHS": [
    {
      "N": "General",
      "I": "6-1"
    },
    {
      "N": "Description of ISPF supplied line commands",
      "I": "6-2"
    },
    {
      "N": "Description of the block command",
      "I": "6-3"
    },
    {
      "N": "Using the \"\\\" character to represent a quoted Data Set name",
      "I": "6-4"
    },
    {
      "N": "Format of the displayed list",
      "I": "6-5"
    },
    {
      "N": "Available primary commands when the list is displayed",
      "I": "6-6"
    },
    {
      "N": "Appendices",
      "I": "6-7"
    },
    {
      "N": "Index",
      "I": "6-8"
    }
  ]
}
],
"FLD": [
  {
    "T": "O",
    "Y": 3,
    "X": 2,
    "C": "AQUA",
    "I": "LOW",
    "L": 78,
    "N": "ZDLTITL",
    "D": "DSLIS - Data Sets Matching SYS1.DGT*"
  },
  {
    "T": "T",
    "A": "ZDATA",
    "Y": 5,
    "X": 2,
    "C": "BLUE",
    "I": "LOW",
    "L": 36,
    "D": "Command - Enter \"\\\" to select action"
  },
  {
    "T": "T",
    "A": "ZDATA",
    "Y": 5,
    "X": 51,
    "D": "Row 1 of 9"
  }
]
```

```

    "C": "AQUA",
    "I": "LOW",
    "L": 30,
    "D": "      Dsorg  Recfm  Lrecl  Blksz"
  },
  {
    "T": "T",
    "A": "ZDATA  ",
    "Y": 6,
    "X": 2,
    "C": "BLUE",
    "I": "LOW",
    "L": 79,
    "D": "-----"
  },
  {
    "T": "I",
    "A": "ZDATA  ",
    "Y": 7,
    "X": 2,
    "C": "GREEN",
    "I": "LOW",
    "L": 53,
    "N": "ZDATA_8788",
    "D": "          SYS1.DGTCLIB"
  },
  {
    "T": "T",
    "A": "ZDATA  ",
    "Y": 7,
    "X": 56,
    "C": "AQUA",
    "I": "LOW",
    "L": 25,
    "D": "PO      FB          80  27920"
  },
  {
    "T": "I",
    "A": "ZDATA  ",
    "Y": 8,
    "X": 2,
    "C": "GREEN",
    "I": "LOW",
    "L": 53,
    "N": "ZDATA_8786",
    "D": "          SYS1.DGTLLIB"
  },
  {
    "T": "T",
    "A": "ZDATA  ",
    "Y": 8,
    "X": 56,
    "C": "AQUA",
    "I": "LOW",
    "L": 25,
    "D": "PO      U          0  32760"
  },
  {
    "T": "I",
    "A": "ZDATA  ",
    "Y": 9,
    "X": 2,
    "C": "GREEN",
    "I": "LOW",
    "L": 53,
    "N": "ZDATA_8784",
    "D": "          SYS1.DGTMKLB"
  },
  {
    "T": "T",
    "A": "ZDATA  ",
    "Y": 9,
    "X": 56,
    "C": "AQUA",
    "I": "LOW",
    "L": 25,
    "D": "PO      FB          80  27920"
  },
  {
    "T": "I",
    "A": "ZDATA  ",
    "Y": 10,

```

## JSON data structures sent from ISPF to client (message type 3)

```

    "X":2,
    "C":"GREEN",
    "I":"LOW",
    "L":53,
    "N":"ZDATA_8782",
    "D":""          SYS1.DGTMLIB          "
  },
  {
    "T":"T",
    "A":"ZDATA    ",
    "Y":10,
    "X":56,
    "C":"AQUA",
    "I":"LOW",
    "L":25,
    "D":"P0      FB          80  27920"
  },
  {
    "T":"I",
    "A":"ZDATA    ",
    "Y":11,
    "X":2,
    "C":"GREEN",
    "I":"LOW",
    "L":53,
    "N":"ZDATA_8780",
    "D":""          SYS1.DGTPKLB         "
  },
  {
    "T":"T",
    "A":"ZDATA    ",
    "Y":11,
    "X":56,
    "C":"AQUA",
    "I":"LOW",
    "L":25,
    "D":"P0      FB          80  27920"
  },
  {
    "T":"I",
    "A":"ZDATA    ",
    "Y":12,
    "X":2,
    "C":"GREEN",
    "I":"LOW",
    "L":53,
    "N":"ZDATA_8778",
    "D":""          SYS1.DGTPLIB         "
  },
  {
    "T":"T",
    "A":"ZDATA    ",
    "Y":12,
    "X":56,
    "C":"AQUA",
    "I":"LOW",
    "L":25,
    "D":"P0      FB          80  27920"
  },
  {
    "T":"I",
    "A":"ZDATA    ",
    "Y":13,
    "X":2,
    "C":"GREEN",
    "I":"LOW",
    "L":53,
    "N":"ZDATA_8776",
    "D":""          SYS1.DGTSKLB         "
  },
  {
    "T":"T",
    "A":"ZDATA    ",
    "Y":13,
    "X":56,
    "C":"AQUA",
    "I":"LOW",
    "L":25,
    "D":"P0      FB          80  27920"
  },
  {
    "T":"I",

```

```

    "A": "ZDATA  ",
    "Y": 14,
    "X": 2,
    "C": "GREEN",
    "I": "LOW",
    "L": 53,
    "N": "ZDATA_8774",
    "D": "          SYS1.DGTSLIB          "
  },
  {
    "T": "T",
    "A": "ZDATA  ",
    "Y": 14,
    "X": 56,
    "C": "AQUA",
    "I": "LOW",
    "L": 25,
    "D": "PO      FB          80  27920"
  },
  {
    "T": "I",
    "A": "ZDATA  ",
    "Y": 15,
    "X": 2,
    "C": "GREEN",
    "I": "LOW",
    "L": 53,
    "N": "ZDATA_8772",
    "D": "          SYS1.DGTTLIB          "
  },
  {
    "T": "T",
    "A": "ZDATA  ",
    "Y": 15,
    "X": 56,
    "C": "AQUA",
    "I": "LOW",
    "L": 25,
    "D": "PO      FB          80  27920"
  },
  {
    "T": "T",
    "A": "ZDATA  ",
    "Y": 16,
    "X": 2,
    "C": "BLUE",
    "I": "LOW",
    "L": 79,
    "D": "***** End of Data Set list *****"
  },
  {
    "T": "T",
    "Y": 24,
    "X": 2,
    "C": "GREEN",
    "I": "LOW",
    "L": 12,
    "D": "Command ==>"
  },
  {
    "T": "I",
    "Z": "TRUE",
    "Y": 24,
    "X": 15,
    "C": "AQUA",
    "I": "LOW",
    "H": "USCORE",
    "L": 48,
    "N": "ZCMD",
    "D": ""
  },
  {
    "T": "T",
    "Y": 24,
    "X": 64,
    "C": "GREEN",
    "I": "LOW",
    "L": 11,
    "D": "Scroll ==>"
  },
  {
    "T": "I",

```

## JSON data structures sent from ISPF to client (message type 3)

```
        "Y":24,
        "X":76,
        "C":"AQUA",
        "I":"LOW",
        "H":"USCORE",
        "L":4,
        "N":"ZUSC",
        "D":"PAGE"
    },
    "KEY":[{"K":"ENTER",
            "N":"ENTER"},
          {"K":"1",
            "N":"Help"},
          {"K":"2",
            "N":"Split"},
          {"K":"3",
            "N":"Exit"},
          {"K":"5",
            "N":"Rfind"},
          {"K":"7",
            "N":"Up"},
          {"K":"8",
            "N":"Down"},
          {"K":"9",
            "N":"Swap"},
          {"K":"10",
            "N":"Left"},
          {"K":"11",
            "N":"Right"},
          {"K":"12",
            "N":"Cancel"}]
    }
}
```

## ISPF action JSON

The following schema describes the JSON data structure used to indicate an action by ISPF:

```
{
  "ACTION":{
    "description":"The signal of an action by ISPF - EXIT=one or more ISPF logical
screens has been exited",
    "type":"string",
    "enum":["EXIT"],
    "required":true
  },
  "SCR":{
    "description":"For an EXIT action request, a list of identifiers for each ISPF
logical screen that has been exited",
    "type":"array",
    "items":{
      "description":"The identifier for the ISPF logical screen that has been exited",
      "type":"string",
      "maxLength":1
    }
  }
}
```

```
}
}
```

## ISPF action JSON example

Here is an example of the JSON to indicate that the user has exited the third logical screen for their ISPF session:

```
{
  "ACTION": "EXIT",
  "SCR": ["3"]
}
```

## JSON data structures sent from client to TSO (message type 7)

### TSO user response JSON

The following schema describes the JSON data structure for a user response to a TSO message:

```
{
  "TSO RESPONSE": {
    "description": "TSO message response from the client",
    "type": "object",
    "properties": {
      "VER": {
        "description": "TSO response JSON version identifier",
        "type": "string",
        "maxLength": 4,
        "required": true
      },
      "DATA": {
        "description": "response text",
        "type": "string",
        "maxLength": 32767
      }
    }
  }
}
```

### TSO user response JSON example

The following example shows the JSON when the user enters a response required by TSO when the command LISTCAT LVL is issued:

```
{
  "TSO RESPONSE": {
    "VERSION": "0100",
    "DATA": "sys1.parmlib"
  }
}
```

### TSO action request JSON

The following schema describes the JSON data structure for a request for action by TSO:

```
{
  "TSO RESPONSE": {
    "description": "TSO action request from the client",
    "type": "object",
    "properties": {
      "VER": {
        "description": "TSO response JSON version identifier",
        "type": "string",
        "maxLength": 4,
        "required": true
      },
      "ACTION": {

```

## JSON data structures sent from client to ISPF (message type 8)

```
    "description": "action request text - ATTN=attention request",
    "type": "string",
    "enum": ["ATTN"]
  }
}
```

### TSO action request JSON example

The following example shows the JSON when the user issues an attention request:

```
{
  "TSO RESPONSE": {
    "VERSION": "0100",
    "ACTION": "ATTN"
  }
}
```

## JSON data structures sent from client to ISPF (message type 8)

---

### User response JSON

The following schema describes the JSON data structure for a user response to an ISPF panel display:

```
{
  "PANEL": {
    "description": "User response to an ISPF panel display",
    "type": "object",
    "properties": {
      "SCREENID": {
        "description": "ISPF logical screen identifier",
        "type": "string",
        "maxLength": 1,
        "required": true
      },
      "NAME": {
        "description": "Panel name",
        "type": "string",
        "maxLength": 8,
        "required": true
      },
      "RESPONSE": {
        "description": "The form of the response made by the user",
        "type": "object",
        "required": true,
        "properties": {
          "TYPE": {
            "description": "The response type - CHOICE=pull-down choice, CMD=command from the client, KEY=function key, PS=point-and-shoot field",
            "type": "string",
            "enum": ["CHOICE", "CMD", "KEY", "PS"],
            "required": true
          },
          "ID": {
            "description": "The response qualification - pull-down choice identifier, command string, function key identifier, or point-and-shoot field name",
            "type": "string",
            "required": true
          }
        }
      },
      "CURSOR": {
        "description": "The location on the panel where the user has placed the cursor",
        "type": "object",
        "required": true,
        "properties": {
          "ROW": {
            "description": "The number of the panel row the cursor is on",
            "type": "integer",
            "maximum": 204,
            "required": true
          }
        }
      }
    }
  }
}
```



```

    "COLUMN":{
      "description":"The number of the panel column the cursor is on",
      "type":"integer",
      "maximum":160,
      "required":true
    }
  },
  "FIELDS":{
    "description":"The panel fields that have been modified by the user",
    "type":"array",
    "items":{
      "NAME":{
        "description":"The name associated with the field",
        "type":"string",
        "maxLength":14
      },
      "DATA":{
        "description":"The updated data for the field",
        "type":"string",
        "maxLength":32767
      }
    }
  }
}

```

## User response JSON examples

The following example shows the JSON when the user selects the View point-and-shoot field for option 1 on the ISPF primary options menu:

```

{
  "PANEL":{
    "SCREENID":"1",
    "NAME":"ISR@PRIM",
    "RESPONSE":{
      "TYPE":"PS",
      "ID":"OPS-7-5"
    },
    "CURSOR":{
      "ROW":6,
      "COLUMN":5
    }
  }
}

```

The following example shows the JSON when the View entry panel (ISPF option 1) is displayed and the user types 'sys1.maclib(abend)' in the Other Data Set Name field, selects the Browse Mode option, and presses the Enter key:

```

{
  "PANEL":{
    "SCREENID":"1",
    "NAME":"ISRBRO01",
    "RESPONSE":{
      "TYPE":"KEY",
      "ID":"ENTER"
    },
    "CURSOR":{
      "ROW":19,
      "COLUMN":42
    },
    "FIELDS":[{
      "NAME":"ZODSN",
      "DATA":"'sys1.maclib(abend)'"
    },
    {
      "NAME":"VIEWM",
      "DATA":"/"
    }
  ]
}

```

## JSON data structures sent from client to ISPF (message type 8)

The following example shows the JSON when the user defines a new data set using the ISPF Data Set Utility function (ISPF option 3.2):

```
{
  "PANEL": {
    "SCREENID": "1",
    "NAME": "ISRUAASE",
    "RESPONSE": {
      "TYPE": "KEY",
      "ID": "ENTER"
    },
    "CURSOR": {
      "ROW": 21,
      "COLUMN": 28
    },
    "FIELDS": [
      {
        "NAME": "ZALMC",
        "DATA": ""
      },
      {
        "NAME": "ZALSPAC",
        "DATA": "trks"
      },
      {
        "NAME": "ZAL1EX",
        "DATA": "10"
      },
      {
        "NAME": "ZAL2EX",
        "DATA": "2"
      },
      {
        "NAME": "ZALDIR",
        "DATA": "10"
      },
      {
        "NAME": "ZALRF",
        "DATA": "fb"
      },
      {
        "NAME": "ZALLREC",
        "DATA": "80"
      },
      {
        "NAME": "ZALBLK",
        "DATA": "27920"
      },
      {
        "NAME": "ZALDSNT",
        "DATA": "pds"
      }
    ]
  }
}
```

## Client action JSON

The following schema describes the JSON data structure for a request from the client for an action from ISPF:

```
{
  "ACTION": {
    "description": "An action request from the client - ATTN=attention  
interrupt request, EXIT=request to exit one or more ISPF screens,  
FORCETERM=terminate ISPF",
    "type": "string",
    "enum": ["ATTN", "EXIT", "FORCETERM"],
    "required": true
  },
  "SCR": {
    "description": "For an EXIT action request, a list of identifiers for each ISPF  
logical screen to be exited",
    "type": "array",
    "items": {
      "description": "The identifier for the ISPF logical screen to be exited",
      "type": "string",
      "maxLength": 1
    }
  }
}
```

```

    }
  }
}

```

## Client action JSON examples

The following example shows the JSON when the client requests ISPF process an attention request:

```

{
  "ACTION": "ATTN"
}

```

The following example shows the JSON when the client requests the exit of 3 ISPF logical screens with identifier 1, 2, and 3:

```

{
  "ACTION": "EXIT",
  "SCR": [ "1",
           "2",
           "3"
        ]
}

```

## ISPF variables

The ISPF variables shown in Table 11 on page 379 are used to indicate to an application that ISPF is running on behalf of a client and to allow the application to provide information to the client to help process the application's panels.

Table 11. ISPF variables used to indicate ISPF is running on behalf of a client

Variable name	Description
ZGUI	If ISPF is communicating with a client using the JSON API, the variable ZGUI has a value of CLIENT.
ZCLIENT	If ISPF is communicating with a client the JSON API, variable ZCLIENT has a value of JSON.
ZDYNSCR	<p>When about to display a panel with a dynamic area that can be scrolled, the application can set the value of this variable to indicate whether the dynamic area can be scrolled up, down, left, or right on the next display. The variable value must be 4 bytes:</p> <p><b>Byte 1</b> Set to Y when the area can be scrolled up.</p> <p><b>Byte 2</b> Set to Y when the area can be scrolled down.</p> <p><b>Byte 3</b> Set to Y when the area can be scrolled left.</p> <p><b>Byte 4</b> Set to Y when the area can be scrolled right.</p>

Table 11. ISPF variables used to indicate ISPF is running on behalf of a client (continued)	
Variable name	Description
ZTBLSCR	<p>When about to issue a table display and the application is using a variable model line to dynamically build the display area for the table rows, the application can set the value of this variable to indicate whether the table display can be scrolled up, down, left, or right on the next display. The variable value must be 4 bytes:</p> <p><b>Byte 1</b> Set to Y when the table can be scrolled up.</p> <p><b>Byte 2</b> Set to Y when the table can be scrolled down.</p> <p><b>Byte 3</b> Set to Y when the table can be scrolled left.</p> <p><b>Byte 4</b> Set to Y when the table can be scrolled right.</p>

---

## Appendix B. Accessibility

Accessible publications for this product are offered through [IBM Documentation \(www.ibm.com/docs/en/zos\)](http://www.ibm.com/docs/en/zos).

If you experience difficulty with the accessibility of any z/OS information, send a detailed message to the [Contact the z/OS team web page \(www.ibm.com/systems/campaignmail/z/zos/contact\\_z\)](http://www.ibm.com/systems/campaignmail/z/zos/contact_z) or use the following mailing address.

IBM Corporation  
Attention: MHVRCFS Reader Comments  
Department H6MA, Building 707  
2455 South Road  
Poughkeepsie, NY 12601-5400  
United States



## Notices

---

This information was developed for products and services that are offered in the USA or elsewhere.

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 grant you any license to these patents. You can send license inquiries, in writing, to:

*IBM Director of Licensing  
IBM Corporation  
North Castle Drive, MD-NC119  
Armonk, NY 10504-1785  
United States of America*

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

*Intellectual Property Licensing  
Legal and Intellectual Property Law  
IBM Japan Ltd.  
19-21, Nihonbashi-Hakozakicho, Chuo-ku  
Tokyo 103-8510, 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 OF 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.

This information could include missing, incorrect, or broken hyperlinks. Hyperlinks are maintained in only the HTML plug-in output for IBM Documentation. Use of hyperlinks in other output formats of this information is at your own risk.

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

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  
Site Counsel  
2455 South Road*

Poughkeepsie, NY 12601-5400  
USA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

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

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

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.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

#### COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

## Terms and conditions for product documentation

---

Permissions for the use of these publications are granted subject to the following terms and conditions.

### **Applicability**

These terms and conditions are in addition to any terms of use for the IBM website.

### **Personal use**

You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

### **Commercial use**

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or



reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

## Rights

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

## IBM Online Privacy Statement

---

IBM Software products, including software as a service solutions, ("Software Offerings") may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user, or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering's use of cookies is set forth below.

Depending upon the configurations deployed, this Software Offering may use session cookies that collect each user's name, email address, phone number, or other personally identifiable information for purposes of enhanced user usability and single sign-on configuration. These cookies can be disabled, but disabling them will also eliminate the functionality they enable.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM's Privacy Policy at [ibm.com/privacy](http://ibm.com/privacy) and IBM's Online Privacy Statement at [ibm.com/privacy/details](http://ibm.com/privacy/details) in the section entitled "Cookies, Web Beacons and Other Technologies," and the "IBM Software Products and Software-as-a-Service Privacy Statement" at [ibm.com/software/info/product-privacy](http://ibm.com/software/info/product-privacy).

## Policy for unsupported hardware

---

Various z/OS elements, such as DFSMSdfp, JES2, JES3, and MVS, contain code that supports specific hardware servers or devices. In some cases, this device-related element support remains in the product even after the hardware devices pass their announced End of Service date. z/OS may continue to service element code; however, it will not provide service related to unsupported hardware devices. Software problems related to these devices will not be accepted for service, and current service activity will cease if a problem is determined to be associated with out-of-support devices. In such cases, fixes will not be issued.

## Minimum supported hardware

---

The minimum supported hardware for z/OS releases identified in z/OS announcements can subsequently change when service for particular servers or devices is withdrawn. Likewise, the levels of other software products supported on a particular release of z/OS are subject to the service support lifecycle of those

products. Therefore, z/OS and its product publications (for example, panels, samples, messages, and product documentation) can include references to hardware and software that is no longer supported.

- For information about software support lifecycle, see: [IBM Lifecycle Support for z/OS \(www.ibm.com/software/support/systemsz/lifecycle\)](http://www.ibm.com/software/support/systemsz/lifecycle)
- For information about currently-supported IBM hardware, contact your IBM representative.

## 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

---

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at [Copyright and Trademark information \(www.ibm.com/legal/copytrade.shtml\)](http://www.ibm.com/legal/copytrade.shtml).

---

# Index

## Special Characters

- & (ampersand)
  - in a variable name [4](#)
  - symbolic variables [7](#)
- % sign
  - beginning a command with [34](#), [203](#)

## A

- abbreviated (generic) search argument [264](#)
- abend [36](#)
- accessibility
  - contact IBM [381](#)
- accessing skeleton files (FTOPEN) [78](#)
- adding a member to a data set or library [135](#)
- adding a row to a table (TBADD) [210](#)
- ADDDPOP parameter [203](#)
- ADDDPOP service
  - description [14](#), [21](#)
  - example [22](#)
  - relation to control service [33](#)
  - return codes [22](#)
- APL2
  - cannot use ISPLINK call [4](#)
  - character-vector [8](#)
  - example [9](#)
  - general call format [8](#)
  - interface with ISPF [8](#)
  - lastrc variable [12](#)
- application commands, definition [245](#)
- ASIS parameter
  - on VGET service [294](#)
- assembler
  - example [10](#)
  - general call format [10](#)
- Assembler language, VL keyword [7](#)
- assistive technologies [381](#)
- attention exits (CLIST) [3](#)
- ATTN statement [3](#)
- audit trail
  - in EDIF recovery mode [54](#), [303](#)
  - in EDIT recovery mode [62](#)
  - in VIEW recovery mode [297](#)
- automatic and non-automatic entry into line mode [34](#), [203](#)
- AUTOSEL (auto-selection)
  - call parameters description [224](#)
  - command procedure description [224](#)
- AUTOSEL control variable, use with TBDISPL [229](#)

## B

- BARRIER keyword [203](#)
- BRIF service, description [23](#)
- Browse Interface service [23](#)
- BROWSE service
  - description [16](#), [28](#)

- BROWSE service (*continued*)
  - recursive use [28](#)

## C

- C
  - example [10](#)
  - general call format [10](#)
- call
  - general format [4](#), [7](#)
  - positional parameters [4](#)
- call interfaces
  - ISPEXEC [3](#), [6](#), [7](#)
  - ISPLINK [5](#)
- call invocation
  - basic interfaces [4](#)
  - general call format
    - APL2 [8](#)
    - COBOL [5](#), [9](#)
    - FORTRAN [7](#)
    - ISPLINK [12](#)
    - Pascal [7](#), [8](#)
    - PL/I [5](#), [9](#)
  - parameters
    - as symbolic variables [7](#)
    - omitting [5](#)
    - positional [5](#), [7](#)
    - types of [5](#)
- CALL ISPEXEC interface [7](#)
- CALL ISPLINK interface [4](#)
- CANCEL mode, effect on error processing [12](#)
- change row in table
  - TBMOD [251](#)
  - TBPUT [254](#)
- character-vector [8](#)
- clear table variables to nulls (TBVCLEAR) [278](#)
- CLIST
  - attention exits [3](#)
  - variables, used in command invocation [3](#)
- close and save table (TBCLOSE) [215](#)
- close table without saving (TBEND) [245](#)
- closing a data set [103](#)
- COBOL
  - call format using ISPEXEC [6](#)
  - example [5](#), [9](#)
  - general call format [4](#)
  - high-order bit generation [5](#)
  - literals
    - in assignment statements [10](#)
    - in call statements, not allowed [10](#)
  - return codes from services [12](#), [13](#)
  - RETURN-CODE built-in variable [12](#)
- coding requests for services
  - keyword parameter [5](#)
  - numeric value parameter [5](#)
  - service name parameter [5](#)
- column of a table, defining [16](#)

- command call
  - general format [3](#)
  - positional parameters [3](#)
- command invocation
  - CLIST variables [3](#)
  - dialog variables as parameters [4](#)
  - general format [3](#)
  - ISPEXEC command [3](#)
  - Option 7.6, Dialog Services [3](#)
  - parameter conventions [3](#)
  - return codes [12](#)
  - variables [3](#)
- command routines and I/O, return codes from [13](#)
- commands
  - reading syntax diagrams [xix](#)
- commands, definition of application [245](#)
- compressing a data set [104](#)
- COND keyword on SETMSG [14](#)
- contact
  - z/OS [381](#)
- CONTROL service
  - ADDPPOP/REMPPOP service relation [33](#)
  - description [18](#), [31](#)
  - example [39](#)
- copying
  - a variable (VCOPY) [281](#)
  - members of a data set [106](#)
  - variables to a shared pool or profile pool (VPUT) [313](#)
- create a new table (TBCREATE) [217](#)
- creating a member list [140](#), [141](#), [158](#)
- CRP, movement of
  - TBBOTTOM [213](#)
  - TBDISPL [223](#)
  - TBSCAN [264](#)
  - TBSKIP [268](#)
  - TBTOP [277](#)
- CSRROW (.CSRROW) control variable [229](#)

## D

- data id
  - definition of [130](#)
  - generating [130](#)
- Data Set Display Service [109](#)
- Data Set Information Dialog Service, DSINFO [51](#)
- Data Set Information Panel, data set list dialog [109](#)
- data structures, JSON [319](#), [321](#), [375](#), [376](#)
- DBCS
  - defining search argument [264](#)
  - sort sequence [271](#)
- DBCS considerations
  - LMGET [127](#)
  - LMPUT [182](#)
- ddname interface [30](#), [66](#), [300](#)
- define function variable (VDEFINE) [283](#)
- delete (set to nulls) table values (TBVCLEAR) [278](#)
- Delete option of LMMDISP [151](#)
- delete row from table (TBDELETE) [220](#)
- delete, a table (TBERASE) [246](#)
- dialog
  - example [237](#)
  - service description [2](#), [21](#)
- dialog function, example [237](#)
- DIRLIST service

- DIRLIST service (*continued*)
  - description [39](#)
- DISPLAY environment
  - EDIT service [62](#)
  - VIEW service [297](#)
- DISPLAY service
  - description [14](#), [45](#)
  - example [49](#)
- display services [14](#)
- DSINFO [51](#)

## E

- EDIF service
  - description of [54](#)
  - recovery mode [54](#), [303](#)
- EDIREC service, description of [60](#)
- Edit interface service [54](#)
- edit macros, ISPF/PDF services in [1](#)
- Edit profile [204](#)
- edit recovery
  - VIEW service [297](#)
- edit recovery table
  - initialization of [60](#), [70](#)
  - scanning for pending recovery [60](#), [71](#)
- EDIT service
  - description [16](#), [62](#)
  - recovery mode [62](#)
  - recursive use [62](#)
  - running in a pop-up window [62](#)
- EDREC service
  - CANCEL option [72](#)
  - DEFER option [72](#)
  - description [16](#), [69](#)
  - INIT option [70](#)
  - PROCESS option [71](#)
  - QUERY option [71](#)
- ending, file tailoring (FTCLOSE) [73](#)
- ENQ issued by TBOPEN [252](#)
- erase (set to nulls) table variables (TBVCLEAR) [278](#)
- erase a table (TBERASE) [246](#)
- erasing
  - member of file tailoring output library (FTERASE) [75](#)
  - variables from shared or profile pool (VDELETE) [292](#)
- erasing a data set [122](#)
- error modes (return code of 12 or higher)
  - CANCEL [12](#)
  - RETURN [12](#)
- exit routine, VDEFINE service [288](#)
- exits, CLIST attention [3](#)

## F

- feedback [xxv](#)
- file tailoring services [14](#)
- find table variable TBSARG [258](#)
- finding a library member [154](#)
- FORTTRAN
  - example [7](#)
  - general call format [4](#), [7](#)
  - high-order bit generation [5](#)
  - ISPEX alternate name [4](#)
  - ISPLNK alternate name [4](#)

- FORTTRAN (*continued*)
  - lastrc variable [7](#)
  - passing arguments [7](#)
  - return code variable [12](#)
  - return codes from services [12](#), [13](#)
  - variable names [4](#)
- fragments, syntax diagrams [xix](#)
- freeing a data set from association with a data ID [125](#)
- FTCLOSE service
  - description [73](#)
  - example [75](#)
- FTERASE service
  - description [75](#)
  - example [76](#)
- FTINCL service
  - description [76](#)
  - example [77](#)
- FTOPEN service
  - description [78](#)
  - example [79](#)
- function commands, definition [245](#)
- function variable pool, LMMDISP, variable saved [141](#)
- function variables, define in function pool (VDEFINE) [283](#)

## G

- generic search argument, specification of
  - TBSARG [258](#)
  - TBSCAN [264](#)
- get a copy of variable (VCOPY) [281](#)
- get row from table (TBGET) [249](#)
- get variable from shared pool or profile pool(VGET) [294](#)
- GETMSG service
  - description [18](#), [79](#)
  - example [81](#)
- graphics interface mode, for 3290 terminal [83](#)
- GRERROR service [81](#)
- GRINIT service
  - description [82](#)
  - example [83](#)
- GRTERM service [83](#)

## I

- I/O and command routines, return codes from [13](#)
- including file tailoring skeleton (FTINCL) [76](#)
- initializing edit recovery [60](#)
- invoking
  - dialog management services [2](#)
  - services [21](#)
- invoking a dialog (SELECT) [201](#)
- ISPEX, alternate call interface name for FORTRAN and Pascal [4](#)
- ISPEX, call interface [4](#), [7](#)
- ISPEXEC
  - call interface [3](#), [4](#), [7](#)
  - command invocation [3](#), [7](#)
  - using DM services [2](#)
- ISPF library, defined [16](#)
- ISPF variables, running on behalf of client [379](#)
- ISPF, ISPQRY, testing if active [1](#)
- ISPF/PDF services
  - BROWSE [16](#)

- ISPF/PDF services (*continued*)
  - command invocation [3](#)
  - description of [16](#)
  - EDIF service [16](#)
  - EDIT [16](#)
  - EDREC [16](#)
  - introduction to [1](#)
  - invoking [2](#)
  - notation conventions [2](#)
  - prerequisites [1](#)
  - with dialog management service [2](#)
- ISPFILU ddname [85](#)
- ISPLINK
  - call interface [4](#), [5](#), [7](#)
  - parameters [5](#), [12](#)
- ISPLINK routine, invoking DM services [2](#)
- ISPLNK
  - alternate call interface name for FORTRAN [4](#)
  - alternate call interface name for Pascal [4](#)
  - call interface [4](#)
  - parameters [7](#)
- ISPLUSR ddname [85](#)
- ISPMUSR ddname [85](#)
- ISPPUSR ddname [85](#)
- ISPQRY, testing if active [1](#)
- ISPSUSR ddname [85](#)
- ISPTABU ddname [85](#)
- ISPTUSR ddname [85](#)

## J

- JSON
  - data structures
    - between ISPF and client [319](#)
    - sent from client to ISPF [376](#)
    - sent from client to TSO [375](#)
    - sent from ISPF to client [321](#)
    - sent from TSO to client [319](#)
  - ISPF variables [379](#)

## K

- keyboard
  - navigation [381](#)
  - PF keys [381](#)
  - shortcut keys [381](#)
- keyword parameter [3](#)
- keyword parameter, coding requests for services [5](#)
- keywords, syntax diagrams [xix](#)

## L

- LANG keyword
  - CREX parameter [203](#)
- lastrc variable
  - APL2 [8](#)
  - FORTTRAN [7](#)
  - Pascal [7](#)
- LIBDEF null statement [85](#), [94](#)
- LIBDEF service [18](#), [84](#)
- library
  - opening [13](#)
  - renaming [187](#)

- library access services [14](#)
- line length on LIST service [100](#)
- line mode
  - automatic entry [34](#), [203](#)
  - non-automatic entry [34](#), [203](#)
- list data set, writing to [18](#), [98](#)
- LIST service description [18](#), [98](#)
- LMCLOSE [14](#)
- LMCLOSE service, description [103](#)
- LMCOMP [15](#)
- LMCOPY [15](#)
- LMCOPY service, description [106](#)
- LMDDISP [15](#)
- LMDDISP service, description [109](#)
- LMDFREE [15](#)
- LMDFREE service, description [112](#), [125](#)
- LMDINIT [15](#)
- LMDINIT service, description [114](#)
- LMDLIST [15](#)
- LMDLIST service, description [117](#)
- LMERASE [15](#)
- LMERASE service, description [122](#)
- LMFREE [15](#)
- LMFREE service, description [125](#)
- LMGET service
  - DBCS considerations [127](#)
  - description [15](#), [126](#)
- LMINIT
  - ddname
    - to BROWSE [30](#)
    - to EDIT [66](#)
    - to VIEW [300](#)
  - description [15](#), [130](#)
- LMMADD [15](#)
- LMMADD service
  - description [135](#), [140](#)
  - statistical variables [136](#), [141](#)
  - ZLMSEC [137](#)
- LMMDEL service [139](#)
- LMMDISP [15](#)
- LMMFIND [15](#)
- LMMFIND service
  - description [154](#)
  - statistical variables [155](#)
- LMMLIST [15](#)
- LMMLIST service
  - description [158](#)
  - FREE option [160](#)
  - LIST option [160](#)
  - statistical variables [159](#)
- LMMOVE [15](#), [163](#)
- LMMOVE service, description [163](#)
- LMMREN [15](#), [166](#)
- LMMREN service, description [166](#)
- LMMREP [15](#), [168](#)
- LMMREP service
  - ZLMSEC [170](#)
- LMMSTATS [15](#)
- LMMSTATS service, description [171](#)
- LMOPEN [15](#)
- LMOPEN service
  - description [177](#)
  - INPUT/OUTPUT options [176](#)

- LMPRINT [16](#)
- LMPRINT service, description [178](#)
- LMPUT [16](#), [104](#)
- LMPUT service
  - DBCS considerations [180](#)
  - description [182](#)
- LMQUERY [16](#)
- LMQUERY service, description [183](#)
- LMRENAME [16](#)
- LMRENAME service, description [187](#)
- LMREP service
  - description [168](#)
  - statistical variables [169](#)
- LMxxxxxx - library access services [14](#)
- LNCT Search-For process statement [15](#)
- load module search order [2](#)
- LOG service
  - description [18](#), [189](#)
  - example [190](#)
- logging a message (LOG service) [18](#)

## M

- mask association with dialog variables (VMASK) [309](#)
- member
  - copying [106](#)
  - deleting [139](#)
  - erasing [122](#)
  - finding [154](#)
  - renaming [166](#)
  - replacing [168](#)
- member list
  - adding a member [149](#)
  - creating [158](#)
  - dialog variables saved [141](#)
  - displaying [142](#)
  - freeing storage space associated with [153](#), [158](#)
  - getting the next member [146](#), [161](#)
  - putting information in the line command field and the user data field [147](#)
- Member List Dialog Service, MEMLIST [190](#)
- MEMLIST [190](#)
- message library, LIBDEF definition [93](#)
- message logging (LOG service) [18](#)
- messages, setting (SETMSG) [208](#)
- model sets, example [221](#), [237](#)
- modify a table row
  - TBMOD [251](#)
  - TBPUT [254](#)
- move current row pointer (CRP)
  - TBBOTTOM [268](#)
  - TBSCAN [268](#)
  - TBSKIP [268](#)
  - TBTOP [268](#), [270](#)
- moving data set members [163](#)
- multicultural support, for numeric representation [271](#)
- MULTX mode
  - on LMGET service [126](#)
  - on LMPUT service [181](#)

## N

- name-list, VSYM service [316](#)

naming restrictions for dialog functions [204](#)

navigation

    keyboard [381](#)

NEST keyword [203](#)

NEWAPPL

    data element search order [86](#)

    description of command procedures [204](#)

numeric value parameter, coding requests for services [5](#)

## O

open a table (TBOPEN) [252](#)

open and create a table (TBCREATE) [217](#)

opening a data set [176](#)

opening skeleton files (FTOPEN) [78](#)

## P

page eject on list data set [100](#)

panel definition, used by TBDISPL [14](#)

parameters

    coding rules for service requests [3, 5](#)

    specified as variables [9](#)

    used as symbolic variables [7](#)

partition mode for 3290 terminal [83](#)

Pascal

    general call format [7](#)

    ISPEX alternate name [4](#)

    ISPLINK alternate name [4](#)

    lastrc variable [7](#)

    passing arguments as variables or literals [8](#)

    return code registers [12](#)

    return code variable [12](#)

    variable names [4](#)

PASSLIB

    data element search order [86](#)

    description of command procedures [205](#)

PDF services, with edit macros [1](#)

percent (%) sign, beginning a command with [34, 203](#)

PL/I

    call format using ISPEXEC [6](#)

    call format using ISPLINK [5](#)

    example of statements you should use [9](#)

    high-order bit generation [5, 7](#)

    PLIRETV build-in function should use [12](#)

    return codes [12](#)

    return codes from services [12, 13](#)

    using literals in assignment statements [9](#)

PLIRETV build-in function [12, 13](#)

pop-up window, and EDIT service [62](#)

POSITION, TBDISPL parameter [229](#)

positional parameters, command invocation [3](#)

PQUERY service, description [18, 193](#)

printing data sets [178](#)

PROFILE parameter

    on VGET service [294](#)

put variables in shared pool or profile pool (VPUT) [313](#)

## Q

QBASELIB [195](#)

QBASELIB service [19](#)

QLIBDEF service [19](#)

QLIBDEF service, description [196](#)

QTABOPEN service [19, 198](#)

Query Base Library Information, QBASELIB [195](#)

QUERYENQ service [19, 198](#)

## R

RACF (Resource Access Control Facility) [29](#)

read a table into virtual storage (TBOPEN) [252](#)

reading a data set record [126](#)

reading row from table

    TBBOTTOM [213](#)

    TBGET [249](#)

    TBSCAN [264](#)

reinitialization section of panel definition, panel processing considerations [47](#)

remove definition of variables from function pool

    VDELETE [291](#)

    VRESET [315](#)

REMPPOP service

    description [18, 201](#)

    relation to control service [33](#)

    return codes [198, 201](#)

renaming a member [166](#)

renaming an ISPF library [187](#)

repeatable items, syntax diagrams [xix](#)

replace a data set member [168](#)

replace variable in function pool (VREPLACE) [314](#)

reset table variables to nulls (TBVCLEAR) [278](#)

reset variables [315](#)

restrictions on member expansion and member part lists

    I/O and command routines (return codes) [11](#)

    service (return codes) [13](#)

retrieve variables from shared pool or profile (VGET) [294](#)

retrieving a row from table

    TBBOTTOM [213](#)

    TBDISPL [223](#)

    TBGET [249](#)

    TBSCAN [264](#)

    TBSKIP [268](#)

return codes

    from services [11](#)

    I/O and command routines

[13](#)

RETURN mode, effect on error processing [12](#)

RETURN-CODE

    COBOL built-in variable [12](#)

    system variables to format error messages [13](#)

row deletion (TBDELETE) [220](#)

row table services [17](#)

row, determine existence (TBEXIST) [248](#)

ROWID, TBDISPL parameter [229](#)

rows of a table, content [16](#)

## S

save and close table (TBCLOSE) [215](#)

save table (TBSAVE) [262](#)

search argument, specification of TBSARG [258](#)

search, a table (TBSCAN) [264](#)

SELECT command

    NEWAPPL [201](#)

    PASSLIB parameter [205](#)



- SELECT service
  - ADDPPOP parameter [14, 203](#)
  - BARRIER keyword [203](#)
  - description [201](#)
  - example [207](#)
  - LANG keyword [203](#)
  - NEST keyword [203](#)
- sending to IBM
  - reader comments [xxv](#)
- service call, general call format [4](#)
- service interface routines [2, 4](#)
- service name parameter, coding on service requests [5](#)
- services
  - command procedure format [1](#)
  - description [1, 264](#)
  - QBASELIB [19](#)
  - QLIBDEF [19](#)
  - QTABOPEN [19](#)
  - QUERYENQ [19](#)
  - TRANS [19](#)
- services description
  - CONTROL [18](#)
  - display [14](#)
  - file tailoring [14](#)
  - LOG [18](#)
  - PQUERY [18](#)
  - table [16](#)
- SETMSG service
  - description [14, 208](#)
  - example [210](#)
- setting row pointer
  - TBBOTTOM [213](#)
  - TBDISPL [213, 223](#)
  - TBSCAN [213, 264](#)
  - TBSKIP [213](#)
  - TBTOP [277](#)
- setting, processing modes (CONTROL) [31](#)
- SETTINGS option, affect on LIST service [100](#)
- SHARED parameter
  - on VGET service [294](#)
- shortcut keys [381](#)
- single name parameter, coding on request for services [5](#)
- SISPSASC [2](#)
- sort information record [270](#)
- spacing on list data set [100](#)
- SSI, returning value of [142, 156](#)
- statistical information
  - setting and storing statistics [171](#)
  - variables
    - LMMADD [136](#)
    - LMMFIND [155](#)
    - LMMLIST [160](#)
    - LMMREP [169](#)
- storing statistics [171](#)
- summary of changes [xxvii, xxviii](#)
- SYMDEF parameter
  - on VGET service [294](#)
- SYMNAMES parameter
  - on VGET service [295](#)
- syntax diagrams, how to read [xix](#)
- syntax rules, services requests (parameters) [5](#)

## T

- table
  - adding or updating information [210](#)
  - columns [16](#)
  - definition [16](#)
  - rows description [16](#)
- table display (TBDISPL) [221](#)
- table services
  - description [16](#)
  - general services [16](#)
  - row services [17](#)
- TBADD service
  - description [210](#)
  - example [212](#)
- TBBOTTOM service
  - description [213](#)
  - example [214](#)
- TBCLOSE service
  - description [215](#)
  - example [217, 238](#)
- TBCREATE service
  - description [217](#)
  - example [219](#)
- TBDELETE service
  - description [220](#)
  - example [221](#)
- TBDISPL service
  - control variables related to [229](#)
  - description [14, 221](#)
  - example [227, 237](#)
  - hints, tips, and techniques [230](#)
  - notes about [242](#)
  - system variables related to [227](#)
  - use with other services [229](#)
  - using [221](#)
- TBEND service
  - description [245](#)
  - example [246](#)
- TBERASE service
  - description [246, 247](#)
- TBEXIST service
  - description [248](#)
  - example [248](#)
- TBGET service
  - description [249](#)
  - example [250](#)
- TBMOD service
  - description [251](#)
  - example [252](#)
- TBOPEN service
  - description [252](#)
  - example [238, 254](#)
- TBPUT service
  - description [254](#)
  - example
    - command procedure function [238](#)
    - using function variable pool values [256](#)
    - using with TBDISPL service [237](#)
- TBQUERY service
  - description [256](#)
  - example [258](#)
- TBSARG service
  - description [258](#)



TBSARG service (*continued*)

example [261](#)

TBSAVE service

description [262](#)

example [264](#)

TBSCAN service

description [264](#)

example [267](#)

TBSKIP service

description [268](#)

example [270](#)

TBSORT service

description [270](#)

example [272](#)

TBSTATS service

description [273](#)

example [277](#)

TBTOP service

description [277](#)

example [278](#)

TBVCLEAR service

description [278](#)

example [279](#)

trademarks [386](#)

TRANS service [19](#)

TRANS service, description [279](#)

translate CCSID data (TRANS) [279](#)

## U

update row in table

TBMOD [251](#)

TBPUT [254](#)

update variables in shared pool or profile pool (VPUT) [313](#)

use count

TBCLOSE (close and save a table) [215](#)

TBCREATE (create a new table) [219](#)

TBEND (close a table without saving) [245](#)

user interface

ISPF [381](#)

TSO/E [381](#)

## V

variable model lines, use [232](#)

variable services summary [17](#)

variables

associate edit mask with (VMASK) [309](#)

clearing to nulls (TBVCLEAR) table [278](#)

copy (VCOPY) [281](#)

define in function pool (VDEFINE) [283](#)

erase from shared profile pool (VERASE) [292](#)

passed as parameter to services [3](#)

remove definition from function pool (VRESET) [315](#)

remove definition of from function pool (VDELETE) [291](#)

replace in function pool (VREPLACE) [314](#)

reset [315](#)

retrieve from shared pool or profile pool (VGET) [294](#)

TBDISPL parameters [229](#)

update in shared pool or profile pool (VPUT) [313](#)

variables, syntax diagrams [xix](#)

VCOPY service

description [281](#)

VCOPY service (*continued*)

example [282](#)

used to access system variables [281](#)

VDEFINE service

description [283](#)

examples [287](#), [290](#)

exit routine [288](#)

VDELETE service

description [291](#)

example [292](#)

VERASE service

description [17](#), [292](#)

example [293](#)

using [292](#)

VGET service

accessing [294](#)

accessing application profile pool [296](#)

View Interface service [303](#)

VIEW service

description [296](#)

recovery mode [297](#)

recursive use [296](#)

VIIF service [303](#)

VL keyword assembler language [5](#), [7](#)

VMASK service

description [309](#)

example [312](#)

VPUT service

accessing application profile pool [313](#)

accessing read-only extension [314](#)

VREPLACE service

description [314](#)

example [315](#)

VRESET service

description [315](#)

example [316](#)

VSYM service

name-list [316](#)

## W

Write data set list dialog [109](#)

writing a message to log file (LOG) [189](#)

writing a record to a data set [180](#)

## Z

ZDSxxxx dialog variables [51](#)

ZEDBDSN [71](#)

ZEDROW [71](#)

ZEDTDSN [71](#)

ZEDTMEM [71](#)

ZEDTRD [71](#)

ZEDUSER [62](#), [297](#)

ZEDUSER extension variable [71](#)

ZEIUSER extension variable [61](#)

ZERRALRM [13](#)

ZERRALRM system variable [13](#)

ZERRHM [13](#)

ZERRHM system variable [13](#)

ZERRLM [13](#)

ZERRLM system variable [13](#)

ZERRMSG [13](#)

ZERRMSG system variable [13](#)  
ZERRSM [13](#)  
ZERRSM system variable [13](#)  
ZLC4DATE [137](#), [141](#), [156](#)  
ZLCDATE [136](#), [141](#)  
ZLCNORC [136](#), [141](#)  
ZLCNORCE [141](#)  
ZLEXT [141](#)  
ZLINORC [137](#), [141](#)  
ZLINORCE [142](#)  
ZLLCMD [142](#)  
ZLLIB [142](#), [155](#)  
ZLM4DATE [137](#), [142](#)  
ZLMDATE [142](#)  
ZLMEMBER [142](#)  
ZLMNORC [137](#), [142](#)  
ZLMNORCE [142](#)  
ZLMOD  
    LMMADD, add a member to a data set [136](#)  
    LMMDISP, member list service [142](#)  
    LMMFIND, find a library member [157](#)  
    LMMREP, replace a member of a data set [169](#)  
ZLMSEC [137](#), [142](#), [157](#), [170](#)  
ZLMTIME [142](#)  
ZLMTOP [142](#)  
ZLPDSUDA [142](#)  
ZLSSI [142](#), [156](#)  
ZLSTLPP system variable [101](#)  
ZLSTNUML [101](#)  
ZLSTTRUN system variable [100](#)  
ZLUDDATA [142](#)  
ZLUSER [137](#), [142](#)  
ZLUSER8 [142](#)  
ZLVERS [136](#), [142](#)  
ZTDMARK system variable [228](#)  
ZTDMMSG system variable [228](#)  
ZTDROWS system variable [228](#)  
ZTDSELS system variable [228](#)  
ZTDTOP system variable [223](#), [228](#)  
ZTDVROWS system variable [228](#)  
ZTEMPF system variable [78](#)





Product Number: 5650-ZOS

SC19-3626-50

