Reference
Note!

Before using this information and the product it supports, be sure to read the general information under "Notices" on page 523.

This edition applies to IBM SMP/E for z/OS, V3R6 (program number 5655-G44) and to all subsequent releases and modifications until otherwise indicated in new editions.

This edition replaces SA22-7772-15.

© Copyright IBM Corporation 1986, 2011.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
Contents

Figures .................................. vii

Tables ................................... ix

About this document .................. xi
Bibliography .............................. xi
The z/OS Basic Skills Information Center .......................... xi

How to send your comments to IBM  xiii
If you have a technical problem ....... xiii

Summary of changes .................... xv
Changes made in SMP/E Version 3 Release 6 ............................ xv
Changes made in SMP/E Version 3 Release 5 ............................ xv
Changes made in SMP/E Version 3 Release 4 ............................ xvi

Chapter 1. Syntax notation and rules ... 1
How to read the syntax diagrams .... 1
Syntax rules for MCS and SMPPARM members. 2
Syntax rules for XML statements .... 3

Chapter 2. SMP/E modification control statements .......................... 5
++APAR MCS .................................. 6
++ASSIGN MCS .................................. 8
Data element MCS ....................... 10
++DELETE MCS ............................... 17
++FEATURE MCS ............................. 21
++FUNCTION MCS ............................ 23
Hierarchical file system element MCS ................................. 26
++HOLD MCS .................................. 37
++IF MCS ...................................... 45
++JAR MCS .................................... 47
++JARUPD MCS .............................. 55
++JCLIN MCS .................................. 60
++MAC MCS .................................... 65
++MACUPD MCS ............................... 72
++MOD MCS .................................... 76
++MOVE MCS .................................... 85
++NULL MCS .................................... 88
++PRODUCT MCS ............................. 89
++PROGRAM MCS .............................. 92
++PTF MCS ..................................... 97
++RELEASE MCS .............................. 100
++RENAME MCS ............................... 105
++SRC MCS ................................. 107
++SRCUPD MCS ............................... 112
++USERMOD MCS ............................. 115
++VER MCS ..................................... 118
++ZAP MCS .................................... 124

Chapter 3. Defining control statements in SMPPARM members ............. 129

Chapter 4. SMP/E data sets and files ........................................ 141
CLIENT ........................................ 141
Distribution library (DLIB) ................. 141
INFILE .......................................... 142
Link library (LKLIB) ......................... 142
ORDERSERVER ................................. 142
OUTFILE ........................................ 143
SERVER .......................................... 143
SMCLNT ......................................... 143
SMCNXL ......................................... 144
SMCPATH ........................................ 144
SMPCSI ......................................... 145
SMPDATA1 ....................................... 146
SMPDATA2 ....................................... 146
SMPEDEBUG .................................... 147
SMPDIR ......................................... 147
SMPDUMMY ...................................... 148
SMPHOLD ......................................... 148
SMPHRPT ......................................... 149
SMPICLN ........................................ 149
SMPJCLN ......................................... 149
SMPJHOME ....................................... 150
SMPLIST ......................................... 150
SMPLOG .......................................... 151
SMPLOGA ......................................... 151
SMPLOGA ......................................... 152
SMPMTS ......................................... 153
SMPMTS ......................................... 154
SMPNTS ......................................... 154
SMPNTS ......................................... 154
SMPOBJ .......................................... 154
SMPPTFIN ....................................... 156
SMPPT ........................................... 157
SMPPTS ......................................... 158
SMPPTS spill data set ....................... 158
SMPPRINTER .................................... 158
SMPRPT ......................................... 159
SMPRPT ......................................... 159
SMPSCDS ........................................ 160
SMPSNAP ......................................... 160
SMPSRVR ........................................ 160
SMPSRS ......................................... 160

© Copyright IBM Corp. 1986, 2011
## Figures

1. Example of using data element MCSs .......................... 12
2. Load module structure for ++ZAP examples .................... 126
3. Sample GIMDDALC member provided in SAMPLIB ............. 132
4. Sample GIMEXITs member provided in SAMPLIB ............. 136
5. Single-CST structure ........................................ 172
6. Multiple-CST structure ......................................... 173
7. Global zone: relationships between entries that control processing ......................................................... 175
8. Target zone and distribution zone: relationships between entries that control processing ......................................................... 177
9. Target Zone: Relationships between Entries that Define Status and Structure ......................................................... 179
10. Distribution zone: relationships between entries that define status and structure ......................................................... 182
11. ASSEM entry: sample LIST output ......................... 184
12. ASSEM entry: sample LIST output when XREF is specified ......................................................... 185
13. ASSEM entry: sample UNLOAD output ................... 186
14. BACKUP entries: sample LIST output .................... 190
15. Data element entry: sample LIST output .................. 193
16. Data element entry: sample LIST output when XREF is specified ......................................................... 194
17. Data element entry: sample UNLOAD output ................ 195
18. DDDEF entry: sample LIST output for a target zone ........ 203
19. DDDEF entry: sample LIST output for a global zone .......... 204
20. DDDEF entry: sample UNLOAD output ................... 205
21. DLIB entry: sample LIST output ............................ 212
22. DLIB entry: sample UNLOAD output ................... 213
23. DLIBZONE entry: sample LIST output ................... 217
24. FEATURE entry: sample LIST output .................... 220
25. FMIDSET entry: sample LIST output .................... 221
26. GLOBALZONE entry: sample LIST output .................. 226
27. Hierarchical file system element entry: sample LIST output ......................................................... 233
28. Hierarchical file system element entry: sample LIST output when XREF is specified ......................................................... 234
29. Hierarchical file system element entry: sample LIST output for SHELLSCR entries ......................................................... 235
30. OS21 element entry: sample UNLOAD output .............. 236
31. HOLDDATA entry: sample LIST output when SYSMOD is not specified ......................................................... 240
32. HOLDDATA listed for SYSMOD entry: sample LIST output when SYSMOD is specified ......................................................... 241
33. JAR entry: sample LIST output ............................ 246
34. JAR entry: sample LIST output when XREF is specified ......................................................... 247
35. Example UNLOAD output for JAR entry .................... 248
36. LMOD entry: sample LIST output .......................... 257
37. LMOD entry: sample LIST output when XREF is specified ......................................................... 259
38. LMOD entry: sample UNLOAD output .......................... 261
39. MAC entry: sample LIST output ............................ 269
40. MAC entry: sample LIST output when XREF is specified ......................................................... 270
41. MAC entry: sample UNLOAD output .................... 271
42. MCS entry: sample LIST output ............................ 274
43. MOD entry: sample LIST output (no cross-zone subentries) ......................................................... 282
44. MOD entry: sample LIST output (cross-zone entries) ......................................................... 283
45. MOD entry: sample LIST output when XREF is specified ......................................................... 284
46. MOD entry: sample UNLOAD output (no cross-zone subentries) ......................................................... 285
47. MOD entry: sample UNLOAD output (cross-zone subentries) ......................................................... 287
48. OPTIONS entry: sample LIST output .......................... 300
49. ORDER entry: sample LIST output .......................... 304
50. PRODUCT entry: sample LIST output .................... 307
51. PROGRAM entry: sample LIST output when XREF is specified ......................................................... 310
52. PROGRAM entry: sample LIST output when XREF is specified ......................................................... 311
53. PROGRAM entry: sample UNLOAD output ............... 312
54. SRC entry: sample LIST output ............................ 316
55. SRC entry: sample LIST output when XREF is specified ......................................................... 317
56. SRC entry: sample UNLOAD output .................... 318
57. SYSMOD entry: sample LIST output for a distribution zone ......................................................... 331
58. SYSMOD entry: sample LIST output for a target zone ......................................................... 332
59. SYSMOD entry: sample LIST output when XREF is specified ......................................................... 333
60. SYSMOD entry: sample UNLOAD output ............... 334
61. SYSMOD entry: sample LIST output for a global zone (Example 1) ......................................................... 342
62. SYSMOD entry: sample LIST output for a global zone (Example 2) ......................................................... 343
63. SYSMOD entry: sample LIST output when HOLDDATA is specified ......................................................... 344
64. TARGETZONE entry: sample LIST output ............... 348
65. UTILITY entry: sample LIST output .......................... 355
66. ZONESET entry: sample LIST output .......................... 358
67. Picture of storage for query output .......................... 390
68. Illustration of VER data structure .......................... 393
69. C syntax of GIMAPI invocation .............................. 396
70. PL/I syntax of GIMAPI invocation ............................ 398
71. Assembler syntax of GIMAPI invocation ....................... 400
72. Example of a shell script .................................. 417
73. Example of alias record type 0 records  ..................... 421
74. Example of alias record type 1 ............................. 423
75. Example of continuation record type 0 ....................... 424
76. Example of element record type 0 ............................ 425
77. Example of element record type 1 ............................ 427
78. Example of header record type 0 ............................. 428
79. Example of library record type 0 ............................ 429

© Copyright IBM Corp. 1986, 2011
<table>
<thead>
<tr>
<th>Example</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example of library record type 1</td>
<td>430</td>
</tr>
<tr>
<td>Example of library record type 2</td>
<td>432</td>
</tr>
<tr>
<td>Example of SYSMOD status records</td>
<td>433</td>
</tr>
<tr>
<td>Example of SMP/E environment record type 0</td>
<td>434</td>
</tr>
<tr>
<td>Trailer record type 0</td>
<td>435</td>
</tr>
<tr>
<td>JCL to call GIMCPTS</td>
<td>447</td>
</tr>
<tr>
<td>Sample GIMCPTS jobstream</td>
<td>449</td>
</tr>
<tr>
<td>Sample GIMDTS jobstream</td>
<td>450</td>
</tr>
<tr>
<td>JCL to call GIMGTPKG</td>
<td>452</td>
</tr>
<tr>
<td>GIMGTPKG example</td>
<td>453</td>
</tr>
<tr>
<td>JCL to call GIMUNZIP</td>
<td>455</td>
</tr>
<tr>
<td>GIMUNZIP example</td>
<td>461</td>
</tr>
<tr>
<td>Sample RECEIVE job for GIMUNZIP</td>
<td>463</td>
</tr>
<tr>
<td>JCL to call GIMXSID</td>
<td>464</td>
</tr>
<tr>
<td>GIMXSID example</td>
<td>465</td>
</tr>
<tr>
<td>JCL to call GIMXTRX</td>
<td>471</td>
</tr>
<tr>
<td>Sample input parameter data set for function LSTTZN</td>
<td>476</td>
</tr>
<tr>
<td>TARGETZN record</td>
<td>476</td>
</tr>
<tr>
<td>Sample input parameter data set for function BMPTZN</td>
<td>477</td>
</tr>
<tr>
<td>PRODLIST record</td>
<td>478</td>
</tr>
<tr>
<td>BITMAP record</td>
<td>478</td>
</tr>
<tr>
<td>Bitstring section of BITMAP record</td>
<td>479</td>
</tr>
<tr>
<td>JCL to call GIMZIP</td>
<td>480</td>
</tr>
<tr>
<td>GIMZIP example</td>
<td>486</td>
</tr>
<tr>
<td>Package attribute file (GIMPAF.XML) example</td>
<td>491</td>
</tr>
<tr>
<td>File attribute file (GIMFAF.XML) example for a sequential data set</td>
<td>500</td>
</tr>
<tr>
<td>File attribute file example for a UNIX file</td>
<td>500</td>
</tr>
<tr>
<td>File attribute file example for a VSAM cluster</td>
<td>501</td>
</tr>
<tr>
<td>JCL to call GIMIAP</td>
<td>508</td>
</tr>
<tr>
<td>Sample DEIINST job for GIMIAP</td>
<td>510</td>
</tr>
<tr>
<td>Sample HFSINST job for invoking GIMIAP</td>
<td>511</td>
</tr>
</tbody>
</table>
### Tables

1. Publications for IBM SMP/E for z/OS, V3R6.............................. xi
2. MCS statements for data elements .................................. 10
3. National language identifiers used for language-unique elements.......................... 13
4. MCS statements for hierarchical file system elements ..................................... 26
5. National language identifiers used for language-unique elements.......................... 27
6. Default values for UTILITY entries ...................................... 350
7. Valid entry values ......................................................... 366
8. Valid subentries for the ASSEM entry .................................. 367
9. Valid subentries for the data element entries .................................. 367
10. Valid subentries for the DDDEF entry .................................... 368
11. Valid subentries for the DLIB entry ........................................ 369
12. Valid subentries for the DLIBZONE entry .................................. 369
13. Valid subentries for the FEATURE entry .................................... 369
14. Valid subentries for the FMIDSET entry ..................................... 370
15. Valid subentries for the GLOBALZONE entry ................................. 370
16. Valid subentries for hierarchical file system element entries ................................ 370
17. Valid subentries for the HOLDDATA entry .................................. 371
18. Valid subentries for the JAR entry ........................................... 372
19. Valid subentries for the LMOD entry .......................................... 372
20. Valid subentries for the MAC entry ........................................... 373
21. Valid subentries for the MOD entry ........................................... 374
22. Valid subentries for the OPTIONS entry ...................................... 375
23. Valid subentries for the ORDER entry ....................................... 376
24. Valid subentries for the PRODUCT entry ...................................... 377
25. Valid subentries for the PROGRAM entry ...................................... 378
26. Valid subentries for the SRC entry ............................................ 378
27. Valid subentries for the SYSMOD entry (GLOBAL zone) ................................ 378
28. Valid subentries for the SYSMOD entry (DLIB and target zones) ......................... 381
29. Valid subentries for the TARGETZONE entry ................................... 384
30. Valid subentries for the UTILITY entry ...................................... 385
31. Valid subentries for the ZONESET entry ..................................... 385
32. Entry and subentry combinations/output .................................... 388
33. QUERY command input parameters ....................................... 391
34. General SMP/E entry list structure ........................................ 391
35. General SMP/E entry structure .............................................. 391
36. General SMP/E subentry structure .......................................... 392
37. VER pseudo-subentry structure (GLOBAL, target, and DLIB zone) ....................... 393
38. Item list structure ......................................................... 393
39. API version ............................................................... 395
40. Alias record type 0 ......................................................... 420
41. Alias record type 1 ......................................................... 422
42. Continuation record type 0 .................................................. 423
43. Element record type 0 ....................................................... 424
44. Element record type 1 ....................................................... 426
45. Header record type 0 ......................................................... 427
46. Library record type 0 ....................................................... 428
47. Library record type 1 ....................................................... 429
48. Library record type 2 ....................................................... 430
49. SYSMOD status record type 0 .............................................. 432
50. SMP/E environment record type 0 ......................................... 433
51. Trailer record type 0 ........................................................ 434
52. GIMMPUXP: exit routine parameter list .................................... 439
53. RECEIVE exit routine: parameter list values .................................. 440
54. RECEIVE exit routine: buffer passed by UXPFRMD .................................. 440
55. Retry exit routine: parameter list values .................................... 442
56. Retry exit routine: parameter list passed by UXPFRMD .................................. 442
57. GIMXSID record format .................................................... 467
58. FEATURE record format .................................................... 467
59. PRODLIST record format .................................................... 469
60. BITPTF record format ...................................................... 470
61. Summary of SMP/E naming conventions .................................... 513
About this document

Use this publication when you need to:
• Allocate SMP/E data sets
• Call SMP/E exit routines or service routines
• Code application programs that use SMP/E application programming interfaces
• Code SMP/E modification control statements
• Code SMPPARM members
• Define which utility programs SMP/E can call
• Transform data elements so that they can be packaged inline

Bibliography

The IBM SMP/E for z/OS, V3R6 publications are available as printable PDF files and BookManager-viewable softcopy on the z/OS Internet Library at http://www.ibm.com/servers/eserver/zseries/zos/bkserv/.

Table 1 lists the IBM SMP/E for z/OS, V3R6 publications and briefly describes each one.

For information about z/OS® publications and more information about the IBM SMP/E for z/OS, V3R6 books, see z/OS Information Roadmap.

Table 1. Publications for IBM SMP/E for z/OS, V3R6

<table>
<thead>
<tr>
<th>Title</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMP/E Messages, Codes, and Diagnosis GA22-7770</td>
<td>Explains SMP/E messages and return codes and the actions to take for each; and how to handle suspected SMP/E problems.</td>
</tr>
<tr>
<td>SMP/E Commands SA22-7771</td>
<td>Explains SMP/E commands and processing in detail.</td>
</tr>
<tr>
<td>SMP/E Reference SA22-7772</td>
<td>Explains SMP/E modification control statements, data sets, exit routines, and programming interfaces in detail and provides additional SMP/E reference material.</td>
</tr>
<tr>
<td>SMP/E User’s Guide SA22-7773</td>
<td>Describes how to use SMP/E to install programs and service.</td>
</tr>
</tbody>
</table>

The z/OS Basic Skills Information Center

The z/OS Basic Skills Information Center is a Web-based information resource intended to help users learn the basic concepts of z/OS, the operating system that runs most of the IBM mainframe computers in use today. The Information Center is designed to introduce a new generation of Information Technology professionals to basic concepts and help them prepare for a career as a z/OS professional, such as a z/OS system programmer.

Specifically, the z/OS Basic Skills Information Center is intended to achieve the following objectives:
• Provide basic education and information about z/OS without charge
• Shorten the time it takes for people to become productive on the mainframe
• Make it easier for new people to learn z/OS.

To access the z/OS Basic Skills Information Center, open your Web browser to the following Web site, which is available to all users (no login required):
http://publib.boulder.ibm.com/infocenter/zos/basics/index.jsp
How to send your comments to IBM

We appreciate your input on this publication. Feel free to comment on the clarity, accuracy, and completeness of the information or give us any other feedback that you might have.

Use one of the following methods to send us your comments:
1. Send an email to mhvrcfs@us.ibm.com
3. Mail the comments to the following address:
   IBM Corporation
   Attention: MHVRCFS Reader Comments
   Department H6MA, Building 707
   2455 South Road
   Poughkeepsie, NY 12601-5400
   U.S.A.
4. Fax the comments to us as follows:
   From the United States and Canada: 1+845+432-9405
   From all other countries: Your international access code +1+845+432-9405

Include the following information:
• Your name and address
• Your email address
• Your telephone or fax number
• The publication title and order number:
  SMP/E V3R6.0 for z/OS V1R13.0 Reference
  SA22-7772-16
• The topic and page number related to your comment
• The text of your comment.

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

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

If you have a technical problem

Do not use the feedback methods listed above. Instead, do one of the following:
• Contact your IBM service representative
• Call IBM technical support
• Visit the IBM support portal at [http://www.ibm.com/systems/z/support/](http://www.ibm.com/systems/z/support/)
Summary of changes

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

Changes made in SMP/E Version 3 Release 6

This document contains information that was previously presented in SMP/E Reference, SA22-7772-15, which supports z/OS Version 1 Release 12.

For SA22-7772-16:

Minor updates were made.

For SA22-7772-15:

New information:

- “Determining the required size of SMPWKDIR” on page 462 was added.

Changed information:

- “+++MOD MCS” on page 76 was updated with information about RMODE(31) and RMOD(31).
- “LMOD entry (distribution and target zone)” on page 249 was updated with information about RMODE(31) for the LKED ATTRIBUTES subentry.
- “MOD entry (distribution and target zone)” on page 275 was updated with information about RMODE(31) for the LKED ATTRIBUTES subentry.
- “UTILITY entry (global zone)” on page 350 was updated.
- Various updates were made to include the use of the HOLDDATA operand.

Changes made in SMP/E Version 3 Release 5

For SA22-7772-14:

Changed information:

- The SYSOUT subentry under the DDDEF entry has been enhanced. For details, see “DDDEF entry (distribution, target, and global zone)” on page 197.
- Information has been added to Chapter 4, “SMP/E data sets and files,” on page 141.
- Information has been added to “GLOBALZONE entry (global zone)” on page 224.

The “Readers’ Comments - We’d Like to Hear from You” section at the back of this publication has been replaced with a new section “How to send your comments to IBM” on page xiii. The hardcopy mail-in form has been replaced with a page that provides information appropriate for submitting readers comments to IBM®.

For SA22-7772-13:

Changed information:
In Chapter 2, “SMP/E modification control statements,” on page 5, the following sections have been updated for corrections:

- “Hierarchical file system element MCS” on page 26
- “++HOLD MCS” on page 37

For SA22-7772-12:

New information:
- These sections were updated to include the new FIXCAT or HOLDFIXCAT subentry:
  - “++HOLD MCS” on page 37 and “++RELEASE MCS” on page 100
  - “HOLDDATA entry (global zone)” on page 239 and Table 17 on page 371
  - “OPTIONS entry (global zone)” on page 292 and Table 22 on page 375
  - Table 27 on page 378 and “SYSMOD entry (global zone)” on page 336
- A description has been added for SMPHRPT; see “SMPHRPT” on page 149
- The parameter COMPAT was added to “EXEC statement” on page 443.

Changed information:
- The UTILITY INPUT subentry of “LMOD entry (distribution and target zone)” on page 249 was updated to state the proper UCLIN syntax when the file is located in the UNIX file system.
- The description for the SOURCEID operand was updated in the following places to describe the long SOURCEID support:
  - “++ASSIGN MCS” on page 8
  - Table 27 on page 378, Table 28 on page 381, and the SOURCEID entry on page 328 and 340
  - Table 61 on page 513
- The SMPOUT DD and the PAGELEN entry (“OPTIONS entry (global zone)” on page 292 and Table 22 on page 375) were updated to include SMPHRPT as one of the applicable data sets.

Changes made in SMP/E Version 3 Release 4

For SA22-7772-11:

New information:
- A new environment variable (PATH) was added to the list of environment variables set by SMP/E if “Designing a shell script for SMP/E processing” on page 413.
- A description of the SMP[J]HOME DD statement was added in “JCL statements used in the DEIINST or HFSINST job” on page 508.

Changed information:
- In Chapter 4, “SMP/E data sets and files,” on page 141 the “Use” of the SMP[J]HOME DD statement was updated.
- In Chapter 2, “SMP/E modification control statements,” on page 5, the “++ZAP MCS” usage note for the NAME statement was updated to add a third way to code the name statement.

For SA22-7772-10:
Changed information:

- In Chapter 4, “SMP/E data sets and files,” on page 141, the data set descriptions for the SMPWRK1, SMPWRK2, SMPWRK3, SMPWRK4, and SMPWRK6 data sets were updated.
- In Chapter 5, “SMP/E data set entries,” on page 171, the ORDERRET option under the ORDER entry has been updated.
- In Chapter 11, “Service routines,” on page 447, syntax notes in the GIMUNZIP and GIMZIP service routines were updated to clarify that a comment may appear between a start-tag and its matching end-tag, but never within a tag; the example of using GIMZIP was updated to include the comment.

Deleted information:

- The following statement in the list of syntax rules in Chapter 1, “Syntax notation and rules,” on page 1 was removed: “Include at least one blank between each operand.” It is recommended but not required to include the blank.

For SA22-7772-09:

Updates were made to support APAR IO03469.

New information:

- Two new data sets, SMPCPATH and SMPJHOME, were added in Chapter 4, “SMP/E data sets and files,” on page 141.
Chapter 1. Syntax notation and rules

This chapter explains the syntax notation and rules for SMP/E modification control statements (MCSs) and OPCODE members used by SMP/E. It describes:

- How to read the notation used to show how control statements should be coded
- The rules to follow when coding control statements

How to read the syntax diagrams

Throughout this publication, the structure defined below is used in describing syntax:

- 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 the statement syntax is continued on the next line.
  The symbol indicates that a statement is continued from the preceding line.
  The symbol indicates the end of a statement.
- Required items appear on the horizontal line (main path).

```
STATEMENT-required_item
```

- Optional items appear below the main path.

```
STATEMENT
    -optional_item
```

- If you can choose from two or more items, they appear in a vertical stack.
  If you must choose one of the items, one item of the stack appears on the main path.

```
STATEMENT
    -required_choice1
    -required_choice2
```

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

```
STATEMENT
    -optional_choice1
    -optional_choice2
```

If one of the optional items is the default, it appears above the main path and the remaining choices will be shown below.

```
STATEMENT
    -default_choice1
    -optional_choice2
    -optional_choice3
```
Syntax notation and rules

- Keywords appear in uppercase (for example, PARM1). They must be spelled exactly as shown.
- Variables appear in lowercase italics (for example, parmx). They represent user-supplied names or values.

```
-STATEMENT-variable-
```

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

```
-STATEMENT-repeatable_item-
```

A repeat arrow above a stack indicates that you can make more than one choice from the stacked items, or repeat a single choice.
- A repeat arrow above a stack of keywords means that you can enter one or more of the keywords. However, each keyword can be entered only once.
- A repeat arrow above a variable means that you can enter one or more values for the variable. However, each value can be entered only once.

- If punctuation marks, parentheses, arithmetic operators, or other such symbols are shown, you must enter them as part of the syntax.
- Sometimes a single substitution represents a set of several parameters. For example, in the following diagram, the callout Parameter Block can be replaced by any of the interpretations of the subdiagram that is labeled Parameter Block:

```
-STATEMENT-CLAUSE1-Parameter Block
```

Syntax rules for MCS and SMPPARM members

Follow these rules when you code SMP/E modification control statements (MCS) and SMPPARM member control statements:
- SMP/E input is case-sensitive. Use uppercase letters to enter all SMP/E keywords. Enter operands in the same case as the intended operand values. Enter the text within a comment in any case you prefer.
- Start each statement on a new logical 80-byte record.

For MCSs, do the following:
- Code the “++” for the MCS in columns 1 and 2.
- Code the MCS name on the same line as the “++”.

For OPCODE member control statements, do the following:
- Code the KEY=xxx operand first.
- Do not continue OPCODE member control statements on a subsequent record.
Syntax notation and rules

Note: Except for these restrictions, the SMP/E MCSs and OPCODE member control statements can begin and end anywhere up to and including column 72.

- You can code optional information in any order, except where noted in the syntax and operand descriptions.
- Separate operands and their values with a blank or comma.

Note: Although the syntax diagrams show only commas when indicating the allowable separator characters for repeating values, one or more blank characters may be used instead to separate repeating values.

- You can continue a statement on more than one line. SMP/E assumes a statement is continued if it did not find a period (.) before column 73.

Notes:
1. OPCODE members are an exception—they cannot span multiple records.
2. If an operand’s value must span multiple lines and that value is delimited by quotation marks, the value should extend up to and including column 72 and restart on column 1 of the next line. Put a quotation mark before the value and another after the value, but do not add extra quotation marks where the value spans lines. Blanks within the quoted value are considered to be part of the value, including any blanks at the beginning of a continuation line.

- Start comments with “"/*"" and end them with “"*/"". The first “"/*"" encountered after the initial “"/*"" will end the comment. A comment can appear anywhere within or after a statement, but should not start before a statement, nor begin in column 1. (When “"/*"" starts in column 1, it indicates the end of an input data set.) A comment after the ending period must start on the same line as the period. You cannot specify any additional operands or comments after that final comment. For example, you can code a comment like this:

```
SET    BDY(MVSTST1) /* Comment after period
    continued on subsequent records is okay. */
```

However, you should not code a comment like this:

```
SET    BDY(MVSTST1) . /* Comment after period okay */
    /* but this comment will give a syntax error */
```

This causes a syntax error at the start of the second comment after the period.

- Comments can be in single-byte characters (such as English alphanumeric characters) or in double-byte characters (such as Kanji).
- End each statement with a period.
- SMP/E completes processing for one statement before it starts processing the next one.
- SMP/E ignores columns 73 through 80. If data, such as a period, is specified beyond column 72, SMP/E ignores it and indicates an error in the statement after the one containing that data.

Syntax rules for XML statements

XML statements may be coded in the CLIENT, SERVER, SYSIN, file attribute, and package attribute files for use with the GIMZIP and GIMUNZIP service routines.

The following syntax rules apply to XML statements:

- SMP/E ignores columns 73 through 80.
Syntax notation and rules

- All tags have a starting and ending delimiter specified as `<keyword>` and `<keyword>`, respectively.
- Any tag that does not contain another tag (that is, nested tags) may have an ending delimiter of either `<keyword>` or just `/`.

![Diagram of start-tag and end-tag with attribute]

- Comments must begin with "<!--" and end with "-->". All data between the "<!--" and the "-->" is ignored. Comments may not be placed inside a tag.
- Any text not contained within comment delimiters is syntax checked.
- Tags are case sensitive; attribute values may be mixed case.
- A tag is not required to start on a new line.
- XML tag names and attribute values may not contain the XML markup characters, '<', '>', and '&'.

```xml
<start-tag>
  attribute="attribute value"
</end-tag>
```

```xml
<start-tag />
  attribute="attribute value"
</>
```
Chapter 2. SMP/E modification control statements

Each SYSMOD processed by SMP/E is composed of two distinct types of data: instructions to SMP/E identifying the elements in the SYSMOD and how to install them, and the actual element replacements or updates. The instructions to SMP/E consist of a series of control statements, called modification control statements (or MCSs). This chapter describes the various MCSs that are processed by SMP/E.

Building SYSMODs (packaging)

Building SYSMODs (“packaging”) includes combining the appropriate MCS statements with software elements to create one or more SYSMODs. Depending on the type of SYSMODs you are building and how you plan to distribute them, packaging can also involve putting the SYSMODs in the proper format on the distribution medium.

Although this book describes the syntax of SMP/E MCS statements, it does not contain all the information you need to use these statements for packaging SYSMODs.

- To package function SYSMODs and the associated service (PTF SYSMODs and APAR SYSMODs), you must use this book along with z/OS Packaging Rules, which contains the rules, restrictions, and recommendations for packaging SYSMODs.

- To package USERMOD SYSMODs, use this book along with SMP/E User’s Guide, which steps you through building a USERMOD and provides USERMOD examples that you might find helpful.

Notes:

1. Each section describing an individual MCS has examples of SYSMODs containing that MCS. In the examples, the MCS being described is underlined. This is done only to make that MCS stand out; it does not imply that any special processing must be done to enter that data.

2. The examples of MCSs do not show the use of all the operands for each MCS. When you want to know how to use a particular operand for a specific MCS, first check the section describing that MCS. If the operand is not shown in an example there, check the index entry for the desired operand to see which other MCSs also contain that operand. Then check the examples under those MCSs. Examples of the use of an operand for one MCS can often illustrate its use for another MCS.
++APAR MCS

++APAR MCS

The ++APAR MCS identifies a service SYSMOD. This type of SYSMOD is a temporary corrective fix to the elements of target system and distribution libraries. All other MCSs for this SYSMOD follow this header MCS. For more information about packaging an APAR fix, see z/OS Packaging Rules.

Syntax

```
++APAR MCS

++APAR(sysmod_id)  DESCRIPTION(description)  FILES(number)  RFDSNPFX(relfile_prefix)  REWORK(level)
```

Operands

**DESCRIPTION**

specifies a descriptive name to be associated with this SYSMOD.

- DESCRIPTION can also be specified DESC.
- The DESCRIPTION value can be in single-byte characters (such as English alphanumeric) or double-byte characters (such as Kanji).
- The DESCRIPTION value can contain up to 64 bytes of data, including blanks. Extra blanks, as well as leading and trailing blanks, are deleted. (For double-byte data, the 64-byte maximum includes all shift-in and shift-out characters, as well as the double-byte characters. Shift-in and shift-out pairs with only blanks between are compressed).
- The DESCRIPTION value can span multiple 80-byte records. Data must continue up to and including column 72 and begin in column 1 of the next line. All data past column 72 is ignored. For single-byte data, the break does not translate to a blank unless a blank is explicitly coded in column 72 of the first line or in column 1 of the second line. For double-byte data, each line must contain both the shift-out and shift-in characters.
- If DESCRIPTION is specified, it must contain at least one non-blank character.
- If parentheses are included in the text, they must be in matched pairs.

**FILES**

specifies the number of relative files belonging to this APAR fix. It can be a decimal number from 1 to 9999. For information about packaging SYSMODs in relative files, see z/OS Packaging Rules.

Notes:

1. Although SMP/E allows you to package APAR fixes in relative files, they are not generally packaged in this format.
2. If a packager uses a high-level qualifier on RELFILE data sets, the RFDSNPFX operand on the header MCS (not the RFPREFIX operand on the RECEIVE command) **must** be used to identify that high-level qualifier.
++APAR MCS

REWORK
specifies the level of this SYSMOD, which was reworked for minor changes. Up to eight numeric characters can be specified.

For SYSMODs supplied by IBM, the REWORK level is yyyyddd, where yyyy is the year the SYSMOD was reworked and ddd is the Julian date.

REWORK allows an updated SYSMOD to be automatically received again, as long as it is more recent than the version that has already been received. This takes the place of rejecting the SYSMOD and receiving it again.

Note: If a SYSMOD appears more than once in the SMPPTFIN data set, the first occurrence may be received. However, none of the subsequent versions of the SYSMOD are received, even if their rework level is higher than the one for the first version of the SYSMOD. (Message GIM40001E is issued for each of the subsequent versions of the SYSMOD.)

RFDSNPFX
identifies to SMP/E the prefix used in the relative file data set names for this SYSMOD. SMP/E uses this prefix when allocating data set names for the SYSMOD’s relative files during RECEIVE processing.

• This operand can be specified only if the FILES operand is also specified.
• The RFDSNPFX value specified on the MCS statement must match the actual prefix used in the data set names for the associated relative files.

For example, if the names of the relative files created for a SYSMOD start with “IBM”, as in IBM.sysmod_id.F1, the header MCS statement for the SYSMOD must specify RFDSNPFX(IBM) so SMP/E knows which prefix to use when allocating the data set names for the SYSMOD’s relative files during RECEIVE processing.

• Following standard data set naming conventions, the prefix can be from 1 to 8 alphanumeric or national ($, #, @) characters or a dash (–).

To enable full Security Server protection for tape data sets and to keep the tape header within the 17-character limit (including periods), you should limit the prefix to 1 to 3 characters. If the name exceeds the 17-character limit, only the rightmost 17 characters are written to the tape header label.

sysmod_id
specifies a unique 7-character system modification identifier for the APAR fix. See “Naming conventions for SYSMODs” on page 517 for more information.

Examples
Here is an example of a SYSMOD containing a ++APAR statement for a temporary fix to module IFBMOD01. As the example shows, this fix is needed to answer APAR AZ12345 on an MVS™ system. The module must be at the service level provided by PTF UZ00004 for function FXY1040.

```plaintext
++APAR(AZ12345) /* APAR type fix */.
++VER(Z038) FMID(FXY1040) /* for MVS product FXY1040 */.
PRE(UZ00004) /* at this service level. */.
++ZAP(IFBMOD01) /* Change to one module */.
DISTLIB(AOSFB) /* in this DLIB. */.
...
... IMASPZAP control statements
...}

Chapter 2. SMP/E modification control statements 7
++ASSIGN MCS

++ASSIGN MCS

The ++ASSIGN MCS assigns a source identifier (source ID) to one or more specified SYSMODs, as long as those SYSMODs are in the SMPPTS data set by the end of RECEIVE processing.

Syntax

```
++ASSIGN MCS

/SM590000/SM590000

++ASSIGN SOURCEID(source_id) TO(sysmod_id)

/SM590000/SM630000
```

Operands

**SOURCEID**

is a 1- to 64-character string identifying the source of the SYSMODs being processed. SMP/E associates the SOURCEID value with the SYSMODs named on the ++ASSIGN MCS. The SOURCEID value can consist of any nonblank character (X'41' through X'FE') except single quotation mark ('), asterisk (*), percent (%), comma (,), left parenthesis (() and right parenthesis ()).

**TO**

specifies the SYSMODs with which the source ID is to be associated.

Usage Notes

- The source ID specified on the ++ASSIGN statement is added to any source ID that was assigned to a specified SYSMOD by the RECEIVE command. It is also added to any source IDs currently associated with a specified SYSMOD that has already been received.
- ++ASSIGN statements are processed only when the SMPPTFIN data set is processed.
  - If the whole SMPPTFIN data set is processed, all ++ASSIGN statements are processed.
  - If only selected SYSMODs are processed, the ++ASSIGN statements for those SYSMODs are processed.
- ++ASSIGN statements are not processed when only the SMPHOLD data set is being processed.
- The SOURCEID and TO values must be validly specified and cannot be blank or null. For more information about source ID naming conventions, see "Naming conventions for source IDs" on page 516.
- The source ID is not assigned to any SYSMODs that are not in the global zone.
- The same SYSMOD cannot appear more than once on a single ++ASSIGN MCS.
- If the same SYSMOD appears on more than one ++ASSIGN MCS, all the source IDs are associated with the SYSMOD.
- The ++ASSIGN MCS is used in the SMPPTFIN data set and can be placed between, before, or after SYSMODs, ++FEATURE MCS, or ++PRODUCT MCS. It must be followed by one of the following: a ++APAR, ++FEATURE, ++FUNCTION, ++PRODUCT, ++PTF, or ++USERMOD MCS; another
++ASSIGN MCS

++ASSIGN MCS; or an end-of-file. If one of these does not follow, SMP/E does not receive the SYSMOD being processed, and it skips the ++ASSIGN MCS.

Examples

Here are some examples of ++ASSIGN statements for SYSMODs from several preventive service levels that have been merged into the same ESO tape. A ++ASSIGN MCS has been placed between the groups of SYSMODs to identify their source:

```plaintext
++ASSIGN SOURCEID(PUT0701) /* service level 0701. */
  TO(UZ12345,UZ12346).
++PTF(UZ12345) /* PTF UZ12345 */.
++VER(Z038) FMID(HXP1100) /* for MVS function HXP1100. */.
++MOD(A) DISTLIB(DN554) /* Update module A. */.
A
++PTF(UZ12346) /* PTF UZ12346 */.
++VER(Z038) FMID(HXP1200) /* for MVS function HXP1200. */.
++MOD(C) DISTLIB(DN554) /* Update module C. */.
C
++ASSIGN SOURCEID(PUT0702) /* service level 0702. */
  TO(UZ12347,UZ12348).
++PTF(UZ12347) /* PTF UZ12347 */.
++VER(Z038) FMID(HXP1100) /* for MVS function HXP1100. */.
++MOD(D) DISTLIB(DN554) /* Update module D. */.
D
++PTF(UZ12348) /* PTF UZ12348 */.
++VER(Z038) FMID(HXP1200) /* for MVS function HXP1200. */.
++MOD(A) DISTLIB(AOS12) /* Update module A. */.
A
++MOD(B) DISTLIB(AOS12) /* Update module B. */.
B
```

Chapter 2. SMP/E modification control statements
Data element MCS

Data element MCSs describe elements that are not macros, modules, or source. Data elements have the following characteristics:

- A data element must be a member of a PDS or PDSE (DSORG=PO) or reside in a sequential data set (DSORG=PS).
- The record format (RECFM) must be F, FA, FM, FS, FB, FBA, FBM, FBS, V, VA, VM, VB, VBA, VBM, VS, or VBS.
- The LRECL for a data element must be from 1 to 32,654.
- The records can be numbered or unnumbered.
- A VSAM data set can be a data element if it is in REPRO format. However, after the data is installed by SMP/E, you must run an AMS REPRO job to create the original form of the VSAM data.

Any type of data element may be installed in any distribution or target library that meets these requirements. For example, CLIST data elements can be installed into variable block target libraries.

When copying a data element during APPLY, ACCEPT, or RESTORE processing, it may be necessary for SMP/E to perform the copy itself, rather than invoking the copy utility. SMP/E performs the copy when:

- The target library or distribution library is a sequential data set.
- The data element must be reformatted to be compatible with the target or distribution library. (For more information about the reformatting of data elements, see the section on reformatting data elements in the APPLY command chapter in SMP/E Commands.)

There are MCSs to replace data elements, just as there are MCSs to replace other types of elements. (There are no MCSs to update data elements.) Table 2 shows the MCSs that can be used for data elements.

Table 2. MCS statements for data elements. If an element is provided in only one language, the x’s can be left off the MCS. If an element is provided in more than one language, replace the x’s with the appropriate value from Table 3 on page 13.

<table>
<thead>
<tr>
<th>MCS</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>++BOOKxxx</td>
<td>Online book member</td>
</tr>
<tr>
<td>++BSINDxxx</td>
<td>Index for an online publications library (bookshelf)</td>
</tr>
<tr>
<td>++CGMxxx</td>
<td>Graphics source for an online book</td>
</tr>
<tr>
<td>++CLIST</td>
<td>CLIST</td>
</tr>
<tr>
<td>++DATA</td>
<td>Data not covered by other types</td>
</tr>
<tr>
<td>++DATA1–++DATA5</td>
<td>IBM generic data types 1–5</td>
</tr>
<tr>
<td></td>
<td>These are for IBM use only, to define elements that are not covered by any existing data types.</td>
</tr>
<tr>
<td>++DATA6xxx</td>
<td>IBM generic data type 6</td>
</tr>
<tr>
<td></td>
<td>This is for IBM use only to define an element not covered by any existing data types.</td>
</tr>
</tbody>
</table>
### Data element MCS

**Table 2. MCS statements for data elements (continued).** If an element is provided in only one language, the x's can be left off the MCS. If an element is provided in more than one language, replace the x's with the appropriate value from Table 3 on page 13.

<table>
<thead>
<tr>
<th>MCS</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>++EXEC</td>
<td>EXEC</td>
</tr>
<tr>
<td>++FONTxxx</td>
<td>Printer Font Object Contents Architecture (FOCA) font</td>
</tr>
<tr>
<td>++GDFxxx</td>
<td>GDF graphics panel</td>
</tr>
<tr>
<td>++HELPxxx</td>
<td>Help information (for example, a member in SYS1.HELP or a dialog help panel)</td>
</tr>
<tr>
<td>++IMGxxx</td>
<td>Graphics image for an online book</td>
</tr>
<tr>
<td>++MSGxxx</td>
<td>Message member (such as for a dialog or for a message data set)</td>
</tr>
<tr>
<td>++PARM</td>
<td>PARMLIB member</td>
</tr>
<tr>
<td>++PNLxxx</td>
<td>Panel for a dialog</td>
</tr>
<tr>
<td>++PROBjxxx</td>
<td>Printer object element</td>
</tr>
<tr>
<td>++PROC</td>
<td>Procedure in PROCLIB</td>
</tr>
<tr>
<td>++PRODXML</td>
<td>Product XML document</td>
</tr>
<tr>
<td>++PRSRCxxx</td>
<td>Printer source element</td>
</tr>
<tr>
<td>++PSEGxxx</td>
<td>Graphics page segment for an online book</td>
</tr>
<tr>
<td>++PUBLBxxx</td>
<td>Online publications library (bookshelf)</td>
</tr>
<tr>
<td>++SAMPxxx</td>
<td>Sample data, program, or JCL in a data set for sample code</td>
</tr>
<tr>
<td>++SKLxxx</td>
<td>File skeleton for a dialog</td>
</tr>
<tr>
<td>++TBLxxx</td>
<td>Table for a dialog</td>
</tr>
<tr>
<td>++TEXTxxx</td>
<td>Text</td>
</tr>
<tr>
<td>++USER1–++USER5</td>
<td>User-defined data types 1–5</td>
</tr>
<tr>
<td></td>
<td>These are for user-defined elements that are not covered by any existing data types.</td>
</tr>
<tr>
<td>++UTINxxx</td>
<td>General utility input</td>
</tr>
<tr>
<td>++UTOUTxxx</td>
<td>General utility output</td>
</tr>
</tbody>
</table>

### Supporting several languages

Some types of elements, such as panels, messages, or text, may have to be translated into several languages. In these cases, the corresponding MCSs contain x's to indicate which language is supported by a given element. Figure 1 on page 12 shows an example where product XX1 must provide both English and French support for a message module, a panel, a panel message, and a sample element.
### Data element MCS

<table>
<thead>
<tr>
<th>+FUNCTION(FXX1101)</th>
<th>+FUNCTION(FXX1102)</th>
</tr>
</thead>
<tbody>
<tr>
<td>+VER(Z038) FMID(EXX1100)</td>
<td>+VER(Z038) FMID(EXX1100)</td>
</tr>
<tr>
<td>++MOD(ZZZMOD0E) message</td>
<td>++MOD(ZZZMOD0F) modules</td>
</tr>
<tr>
<td>DISTLIB(AZZZMOD01)</td>
<td>DISTLIB(AZZZMOD01)</td>
</tr>
<tr>
<td>++PNLENU(ZZZPNL01) panels</td>
<td>++PNLFRA(ZZZPNL01)</td>
</tr>
<tr>
<td>DISTLIB(AZZZPNLE)</td>
<td>DISTLIB(AZZZPNLE)</td>
</tr>
<tr>
<td>SYSLIB(SZZZPNLE)</td>
<td>SYSLIB(SZZZPNLE)</td>
</tr>
<tr>
<td>++MSGENU(ZZZMSG01) dialog</td>
<td>++MSGFRA(ZZZMSG01) messages</td>
</tr>
<tr>
<td>DISTLIB(AZZZMSGE)</td>
<td>DISTLIB(AZZZMSGE)</td>
</tr>
<tr>
<td>SYSLIB(SZZZMSGE)</td>
<td>SYSLIB(SZZZMSGE)</td>
</tr>
<tr>
<td>++SAMPENU(ZZZSMP01) samples</td>
<td>++SAMPFRA(ZZZSMP01)</td>
</tr>
<tr>
<td>DISTLIB(AZZZSAME)</td>
<td>DISTLIB(AZZZSAME)</td>
</tr>
<tr>
<td>SYSLIB(SZZZSAME)</td>
<td>SYSLIB(SZZZSAME)</td>
</tr>
</tbody>
</table>

**Figure 1. Example of using data element MCSs**

**Notes:**

1. The message modules can be in the same distribution library, because the element names are different.

2. For the panels, dialog messages, and samples, there is a different element *type* for each language version of an element. Therefore, the element *name* can remain the same for all the languages in which the element is supported. However, elements with the same name must be installed in different libraries. (SMP/E does not check whether different types of data elements have the same name. Likewise, SMP/E does not prevent elements with the same name from being installed in the same libraries.)

Table 3 on page 13 shows the xxx values that can be used when the MCS indicates the language being supported.
Table 3. National language identifiers used for language-unique elements

<table>
<thead>
<tr>
<th>Value</th>
<th>Language</th>
<th>Value</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARA</td>
<td>Arabic</td>
<td>HEB</td>
<td>Hebrew</td>
</tr>
<tr>
<td>CHS</td>
<td>Simplified Chinese</td>
<td>ISL</td>
<td>Icelandic</td>
</tr>
<tr>
<td>CHT</td>
<td>Traditional Chinese</td>
<td>ITA</td>
<td>Italian (Italy)</td>
</tr>
<tr>
<td>DAN</td>
<td>Danish</td>
<td>ITS</td>
<td>Italian (Switzerland)</td>
</tr>
<tr>
<td>DES</td>
<td>German (Switzerland)</td>
<td>JPN</td>
<td>Japanese</td>
</tr>
<tr>
<td>DEU</td>
<td>German (Germany)</td>
<td>KOR</td>
<td>Korean</td>
</tr>
<tr>
<td>ELL</td>
<td>Greek</td>
<td>NLB</td>
<td>Dutch (Belgium)</td>
</tr>
<tr>
<td>ENG</td>
<td>English (United Kingdom)</td>
<td>NLD</td>
<td>Dutch (Netherlands)</td>
</tr>
<tr>
<td>ENP</td>
<td>Uppercase English</td>
<td>NOR</td>
<td>Norwegian</td>
</tr>
<tr>
<td>ENU</td>
<td>English (United States)</td>
<td>PTB</td>
<td>Portuguese (Brazil)</td>
</tr>
<tr>
<td>ESP</td>
<td>Spanish</td>
<td>PTG</td>
<td>Portuguese (Portugal)</td>
</tr>
<tr>
<td>FIN</td>
<td>Finnish</td>
<td>RMS</td>
<td>Rhaeto-Romanic</td>
</tr>
<tr>
<td>FRA</td>
<td>French (France)</td>
<td>RUS</td>
<td>Russian</td>
</tr>
<tr>
<td>FRB</td>
<td>French (Belgium)</td>
<td>SVE</td>
<td>Swedish</td>
</tr>
<tr>
<td>FRC</td>
<td>French (Canada)</td>
<td>THA</td>
<td>Thai</td>
</tr>
<tr>
<td>FRS</td>
<td>French (Switzerland)</td>
<td>TRK</td>
<td>Turkish</td>
</tr>
</tbody>
</table>

Syntax
The syntax to be used depends on the processing to be done for the element:
- Adding or replacing the element
- Deleting the element

Adding or replacing a data element

```
+element(name)
   | ALIAS(alias)
   | DISTLIB(ddname)
   | RELFILE(number)
   | TXLIB(ddname)
   | FROMDS(dsn)NUMBER(n)
   | VOL(vol)
   | UNIT(unit)
   | RMID(sysmod_id)
   | SYSLIB(ddname)
   | VERSION(sysmod_id)
   | SYSLIB(ddname)
```
Deleting a data element

Data element MCS

```
+++element{name}---DELETE
DISTLIB{ddname}
VERSION{sysmod_id}
```

Operands

**ALIAS**

specifies the alias names for the data element in both the target and distribution libraries.

You can use ALIAS when data elements of the same type must be defined in the same zone and must have the same name for programming access. In this case, you can specify the common name on ALIAS and a unique name as the data element name.

**DELETE**

indicates that the element and all of its alias names are to be removed from the target libraries, the distribution libraries, and the SMP/E data sets.

Notes:

1. DELETE is mutually exclusive with all other operands except DISTLIB and VERSION.
2. If the element statement is in a base function, you may want to use the DELETE operand on the ++VER MCS to delete the previous release, rather than on the element statement to delete a specific element.
3. Specification of the DELETE operand results in all alias names of the data element being deleted along with the data element identified.

**DISTLIB**

specifies the ddname of the distribution library for the data element.

Notes:

1. DISTLIB must be specified if the element has not been previously recorded on the target zone or distribution zone.
2. If a data element entry already exists in the target zone or distribution zone and the value currently in that entry does not match that specified in the DISTLIB operand, the SYSMOD is not applied or accepted.

**element**

specifies the type of element. [Table 2 on page 10](#) shows the MCSs used for the various element types.

**FROMDS**

identifies the partitioned data set that contains this element.

Note: The FROMDS operand and its DSN, NUMBER, VOL, and UNIT suboperands are included in the MCS generated by the BUILDMCS command. IBM does not intend the FROMDS operand to be used in manually coded MCS.

**DSN**

specifies the dsname of the FROMDS data set. The specified data set name
Data element MCS

must conform to standard data set naming conventions and cannot contain parentheses. The maximum length of the entire name is 44 characters (including the periods).

**NUMBER**
specifies a number that SMP/E is to use when assigning a name to the SMPTLIB data set associated with this FROMDS data set. (This is similar to the way the relative file number is used in RELFILE processing.)

**VOL**
specifies, for an uncataloged data set, the volume serial number of the volume containing the FROMDS data set. If specified, this volume identifier must be from 1 to 6 alphanumeric characters.

VOL may be omitted for a cataloged data set.

**UNIT**
specifies, for an uncataloged data set, the UNIT type containing the FROMDS data set. If specified, the UNIT value must be from 1 to 8 characters and must conform to standard UNIT naming conventions.

SMP/E accepts any nonblank characters specified between the open and close parentheses, up to a maximum length of 8.

UNIT may be omitted for a cataloged data set.

**Note:** FROMDS is mutually exclusive with DELETE, RELFILE, and TXLIB.

**name**
specifies the name of the element to be replaced. The name can contain any alphanumeric characters and $, #, @, or hex C0.

**RELFILE**
specifies which relative file associated with the SYSMOD contains this element. This operand is required if you provide the element in RELFILE format, rather than inline or in a TXLIB data set.

**Note:** RELFILE is mutually exclusive with DELETE, FROMDS, and TXLIB.

**RMID**
specifies the last PTF that replaced this data element. This operand can be used only in a service-updated function, and the specified PTF must be integrated into the function.

**Note:** RMID is mutually exclusive with DELETE.

**SYSLIB**
specifies the ddname of the target library for the specified element.

**Note:** SYSLIB is mutually exclusive with DELETE.

**TXLIB**
specifies the ddname of the partitioned data set containing the element. This operand is required if the element is provided in a TXLIB data set rather than inline or in RELFILE format.

**Notes:**
1. SMPTLIB cannot be used as a value on the TXLIB operand.
2. TXLIB is mutually exclusive with DELETE, FROMDS, and RELFILE.

**VERSION**
specifies one or more function SYSMODs that currently contain the element.
Data element MCS

The function containing the data element statement takes over ownership of the element from the specified functions.

When VERSION is specified on an element statement, it overrides any VERSION operand values specified on the ++VER MCS.

Usage Notes

- If the element is packaged inline, it must immediately follow the data element MCS and must not contain any records that start with the characters ++. Neither FROMDS, nor RELFILE, nor TXLIB can be specified on the data element MCS.
- To be packaged inline, a data element must contain fixed-block-80 records. If the original format of the element is not fixed-block-80 records, you can use GIMDTS to transform the element into the required format before packaging it. Later, when SMP/E installs the element, it is changed back to its original format. For more information about using GIMDTS, see "GIMDTS: Data transformation service routine" on page 450.
- If the data element is packaged in a TXLIB data set, the ddname specified on the TXLIB operand is required during APPLY and ACCEPT processing.
- For information about packaging SYSMODs in RELFILE, TXLIB, or inline format, see z/OS Packaging Rules.

Examples

The following examples are provided to help you use the data element MCS:

Example 1: Packaging a CLIST in a function

Here is an example of a SYSMOD containing a ++CLIST statement to install your CLIST and have SMP/E track any changes to it. It can be packaged as a function, as shown below:

```smp/e
++FUNCTION(MYCLST1) /* Function. */.
++VER(Z838) /* For MVS systems. */.
++CLIST(CLIST1) /* Install this CLIST */.
    TXLIB(NEWSMP) /* from this TXLIB */.
    DISTLIB(AMACLICL) /* into this DLIB and */.
    SYSLIB(ISPCCLIB) /* this target library. */.
```

Example 2: Packaging a renamed CLIST

Suppose that, for some reason, you need to rename CLIST1, which was introduced in "Example 1: Packaging a CLIST in a function." The new name is to be CLISTX. You do not need to change anything else about the CLIST. Here is an example of a SYSMOD containing the data element statements needed to package this renamed CLIST:

```smp/e
++USERMOD(USR0001) /* User modification */.
++VER(Z838) FMID(MYCLST1) /* to user application. */.
++CLIST(CLIST1) /* Delete the original */.
    DELETE /* CLIST. */.
++CLIST(CLISTX) /* Add the renamed CLIST */.
    TXLIB(NEWSMP) /* from this TXLIB */.
    DISTLIB(AMACLICL) /* into this DLIB and */.
    SYSLIB(ISPCCLIB) /* this target library. */.
```

You must ensure that the renamed CLIST is in the TXLIB library (NEWSMP) used to provide SMP/E with the element.
During APPLY processing, the ++DELETE MCS deletes a load module and any known alias or symbolic link names from a target library and the associated target zone. It can also delete alias or symbolic link names without deleting the load module itself.

During ACCEPT processing, if inline JCLIN was processed, the distribution zone's entries are updated to reflect the ++DELETE changes. The target libraries and target zone entries are not updated.

**Note:** This change is not reversible. You cannot restore the SYSMOD containing the ++DELETE MCS; therefore, you cannot recreate the deleted load module.

**Syntax**

```plaintext
++DELETE MCS
```

**Operands**

**ALIAS**

specifies that only the indicated alias or symbolic link names are to be deleted, but not the load module.

- An alias name can have up to 1023 characters.

  Although a load module residing in a PDSE can have an alias name greater than 8 characters, the ++DELETE statement cannot be used to delete such an alias value. To delete such an alias value without deleting the load module, you need to resupply JCLIN to define the load module without providing an ALIAS statement for the alias value to be deleted. Make sure to also include a ++MOD statement for a module in the load module to force SMP/E to relink the load module.

- Single apostrophes must be used as delimiters for an alias name in these cases:
  - The alias name contains a character that is not uppercase alphabetic, numeric, national ($, #, or @), slash (/), plus (+), hyphen, period, or ampersand (&).
  - The alias name is continued on another line in the control statement. Apostrophes used as delimiters do not count as part of the 1023-character limit.

- If an apostrophe is part of the alias name and is not a delimiter, it needs to be doubled. These two apostrophes count as 2 characters in the 1023-character limit.
++DELETE MCS

- Characters specified must be in the range X'41' through X'FE'. Unlike the binder, SMP/E does not support shift-in (X'0F') and shift-out (X'0E') characters.
- When processing the ++DELETE statement, SMP/E uses the alias name as is, and does not enforce any rules the binder might be using as a result of the CASE execution parameter.

Notes:
1. Do not specify this operand if you also want to delete the load module.
   ++DELETE without the ALIAS operand automatically deletes any alias or symbolic link names associated with the load module.
2. When ALIAS is specified, SMP/E checks the ALIAS control statements in the LMOD entry to verify that the specified name is actually an alias or symbolic link of the load module. For copied load modules, instead of looking for ALIAS control statements, SMP/E checks the corresponding MOD entry's TALIAS subentries.
3. If a valid ALIAS value is specified, SMP/E deletes the alias from all known target libraries, including any associated side deck library, no matter what was coded for SYSLIB. SMP/E overrides the SYSLIB value with SYSLIB(ALL).
4. When you specify ALIAS to delete an alias for a load module, you must reflect this change using JCLIN. To do this, include a ++JCLIN statement with JCLIN data containing a link-edit step for the load module, with the alias deleted from the list of aliases on the link-edit ALIAS statement. This causes SMP/E to replace the alias list in the LMOD entry.

name specifies the name of the load module to be deleted.

SYSLIB specifies the ddname of the target library where the load module resides.
- If ALL is specified, the load module is deleted from all target libraries defined in the target zone.
- If a single ddname is specified, the load module is deleted only from the indicated target library.
  If the load module or alias is to be deleted from two target libraries, a second ddname can be specified. However, to make sure the module is deleted from both libraries, you should use ALL instead.
- For load modules with a SYSLIB allocation (or load modules having a CALLLIBS subentry), specifying ALL or deleting the load module from all the target libraries in which it resides, deletes the base version of the load module from the SMPLTS library.
- For load modules with a side deck library, specifying ALL or deleting the load module from all the target libraries in which it resides, deletes the associated definition side deck from the side deck library.

Note: If ALIAS is specified, SMP/E deletes the alias from all known target libraries, including any associated side deck library, no matter what was coded for SYSLIB. SMP/E overrides the SYSLIB value with SYSLIB(ALL).

Usage Notes
- SYSLIB must always be specified to identify the affected target libraries.
++DELETE MCS

- **ALIAS** should be specified only if alias names, and not load modules, are to be deleted.
- ++DELETE statements must follow any ++VER and ++IF statements and must precede any ++JCLIN or element MCSs.
- Regardless of the order in which ++MOVE, ++RENAME, and ++DELETE statements are coded in a SYSMOD, they are always processed in this order for APPLY and ACCEPT:
  1. ++MOVE
  2. ++RENAME
  3. ++DELETE

  Afterwards, the ++JCLIN statements are processed, and then the element statements are processed.

  **Note:** You cannot restore the SYSMOD containing the ++DELETE MCS.

**Examples**

The following examples are provided to help you use the ++DELETE MCS:

**Example 1: Deleting a single load module**
Here is an example of a SYSMOD containing a ++DELETE statement that deletes load module LMODA from SYS1.LINKLIB:

```plaintext
++PTF(UR01234) /* Identify the PTF number. */.
++VER(Z038) FMID(HXY1300) /* For MVS function HXY1300. */.
++IF (ESY1300) THEN /* If ESY1300 is installed */
  Req(UR12399) /* UR12399 is required. */.
++HOLD(UR01234) /* Hold UR01234. */
  FMID(HXY1300) /* For MVS function HXY1300. */
  System /* System hold */
  Reason(DELETE) /* because of ++DELETE. */
  Comment(This deletion of LMODA from LINKLIB is irreversible.)
++DELETE (LMODA) /* Delete load module LMODA */
  SYSLIB(LINKLIB) /* from LINKLIB. */
++JCLIN /* JCLIN follows. */.
```

**Example 2: Deleting an alias from a load module**
Assume that IBM has shipped you a PTF that deletes an alias for load module LMODA. Here is an example of a SYSMOD containing a ++DELETE statement that deletes an alias from LMODA:

```plaintext
++PTF(UR01235) /* Identify the PTF number. */.
++VER(Z038) FMID(HXY1300) /* For MVS function HXY1300. */.
++HOLD(UR01235) /* Hold UR01235. */
  FMID(HXY1300) /* For MVS function HXY1300. */
  System /* System hold */
  Reason(DELETE) /* because of ++DELETE. */
  Comment(The deletion of the alias for LMODA is irreversible.)
++DELETE (LMODA) /* Identify LMOD LMODA. */
  SYSLIB(AL1) /* Process all SYSLIBs. */
  Alias(OTHNAME) /* Identify alias. */
++JCLIN /* JCLIN to delete alias. */.
```
Example 3: Deleting an alias from a load module in a UNIX file system
Assume that IBM has shipped you a PTF that deletes an alias for load module BPXLMODB, which resides in a UNIX file system. Here is an example of a SYSMOD containing a ++DELETE statement that deletes an alias from BPXLMODB:

```plaintext
++PTF(UZ00440) /* Identify the PTF number. */
++VER(Z83B) FMID(HOP1101) /* For MVS function HOP1101.*/
++HOLD(UZ00440) /* Hold UZ00440. */
    FMID(HOP1101) /* For MVS function HOP1101.*/
    SYSTEM /* System hold */
    REASON(DELETE) /* because of ++DELETE. */
    COMMENT(THE DELETION OF THE ALIAS FROM BPXLMODB IS IRREVERSIBLE).
++DELETE (BPXLMODB) /* Identify LMOD BPXLMODB. */
    SYSLIB(ALL) /* Process all SYSLIBs. */
    ALIAS('../nicename') /* Identify alias. */
++JCLIN /* JCLIN follows. */
    *
    *
++MOD(BPXMODAA) DISTLIB(A05XX) /* Element MCS statements. */
```
++FEATURE MCS

The ++FEATURE MCS is used to describe a set of function SYSMODs that are collectively referred to as a FEATURE. It introduces descriptive information about a new or replacement set of functions into the global zone. A ++FEATURE MCS may be associated with an orderable software feature.

Syntax

```
++FEATURE MCS

++FEATURE(name) DESCRIPTION(description)

FMID(fmID)

PRODUCT(prodID,vv,rr,mm) REWORK(level)
```

Operands

- **name**: is a 1- to 8-character feature name. It can contain uppercase alphabetic, numeric, or national characters ($, #, @).

- **DESCRIPTION**: describes the feature that represents this collection of function SYSMODs.
  - DESCRIPTION can also be specified as DESC.
  - The DESCRIPTION value can be in single-byte characters (such as English alphanumeric) or double-byte characters (such as Kanji).
  - The DESCRIPTION value can contain up to 64 bytes of data, including blanks. (For double-byte data, the 64-byte maximum includes all shift-in and shift-out characters, as well as the double-byte characters.) Extra blanks as well as leading and trailing blanks are deleted.
  - The DESCRIPTION value can span multiple 80-byte records. Data must continue up to and including column 72 and begin in column 1 of the next line. All data past column 72 is ignored. The break does not translate to a blank unless a blank is explicitly coded in column 72 of the first line or in column 1 of the second line.
  - The DESCRIPTION value cannot be only blanks.
  - If parentheses are included in the text, they must be in matched pairs.

- **FMID**: specifies the list of all function SYSMODs that make up this feature. Each FMID is 7 characters long and must be a valid SYSMOD ID. That is, it must contain uppercase alphabetic, numeric, or national characters ($, @, #). If multiple FMIDs are specified, they must be separated by commas.

- **PRODUCT**: identifies the `prodID` and `vv,rr,mm` of the product with which this feature is associated.

- **REWORK**: is the level of this feature, which was reworked for minor changes. Up to eight numeric characters can be specified.
++FEATURE MCS

For IBM features, the REWORK level is yyyyddd, which is the year followed by the Julian date (for example, 2008110).

REWORK allows an updated feature to be automatically received again, as long as it is more recent than the version that has already been received. This takes the place of rejecting the feature and receiving it again.

**Note:** If a ++FEATURE statement appears more than once in the SMPPTFIN data set, the first occurrence may be received. However, none of the subsequent versions of the ++FEATURE statement are received, even if their rework level is higher than the one for the first version of the ++FEATURE statement.

**Usage Notes**

- The ++FEATURE statements are processed only when the SMPPTFIN data set is processed.
  - If the whole SMPPTFIN data set is processed, all ++FEATURE statements are processed.
  - If only selected SYSMODs are processed, SMP/E processes the ++FEATURE statements that name at least one of the selected SYSMODs in their FMID operand list.

++FEATURE statements are not processed when only the SMPHOLD data set is being processed.

- The name, DESCRIPTION, and PRODUCT values are required and cannot be blank or null.

- The ++FEATURE MCS is used in the SMPPTFIN data set and can be placed between, before, or after SYSMODs, ++FEATURE MCS, or ++PRODUCT MCS. If the function SYSMODs identified in the FMID operand of the ++FEATURE statement have not been previously received, the ++FEATURE statement must follow these function SYSMODs in the SMPPTFIN data set.

The ++FEATURE statement must be followed by one of the following: a ++APAR, ++ASSIGN MCS, ++FUNCTION, ++PRODUCT, ++PTF, ++USERMOD, another ++FEATURE MCS, or an end-of-file. If one of these does not follow, SMP/E skips the ++FEATURE MCS.

**Example**

Here is an example of a ++FEATURE MCS for the OS/390® product.

```
++FEATURE( OS325BAS ) /* Feature definition */
DESCRIPTION( OS/390 Base Feature ) /* Description */
FMID(HBB6605,HMP1B00,JBB66C5,...) /* FMID List */
PRODUCT (5647-A01,2.5.0) /* Owning Product */
```
**++FUNCTION MCS**

The ++FUNCTION MCS identifies a SYSMOD as a base function or dependent function. This type of SYSMOD introduces a new or replacement function into the target system and distribution libraries. All other MCSs follow this header MCS statement. For more information about packaging a function, see [z/OS Packaging Rules](#).

### Syntax

```
++FUNCTION MCS

++FUNCTION(sysmod_id)
  DESCRIPTION(description)
  FESN(fe_service_number)
  FILES(number)
  RFDSNPFX(relfile_prefix)
  REWORK(level)
```

### Operands

**DESCRIPTION**

specifies a descriptive name to be associated with this SYSMOD. [z/OS Packaging Rules](#) manual.

- DESCRIPTION can also be specified DESC.
- The DESCRIPTION value can be in single-byte characters (such as English alphanumeric) or double-byte characters (such as Kanji).
- The DESCRIPTION value can contain up to 64 bytes of data, including blanks. (For double-byte data, the 64-byte maximum includes all shift-in and shift-out characters, as well as the double-byte characters.) Extra blanks, as well as leading and trailing blanks are deleted.
- The DESCRIPTION value can span multiple 80-byte records. Data must continue up to and including column 72 and begin in column 1 of the next line. All data past column 72 is ignored. The break does not translate to a blank unless a blank is explicitly coded in column 72 of the first line or in column 1 of the second line.
- If DESCRIPTION is specified, it must contain at least one non-blank character.
- If parentheses are included in the text, they must be in matched pairs.

**FESN**

is a 7-character field engineering (FE) service number.

**FILES**

specifies the number of relative files belonging to this function. It can be a decimal number from 1 to 9999, and is used only if the function is packaged in relative files, rather than inline or in indirect libraries. For information about packaging SYSMODs in relative files, see [z/OS Packaging Rules](#).
++FUNCTION MCS

Notes:
1. Functions are generally packaged in relative files to improve SMP/E's performance when applying and accepting the SYSMOD.
2. If a packager uses a high-level qualifier on RELFILE data sets, the RFDSNPFX operand on the header MCS (not the RFPREFIX operand on the RECEIVE command) must be used to identify that high-level qualifier.

REWORK
is the level of this SYSMOD, which was reworked for minor changes. Up to eight numeric characters can be specified.
REWORK is generally used only for SYSMODs supplied by IBM that have been reworked for minor changes, such as for a service update or to use a ++MOVE, ++RENAME, or ++DELETE MCS. For these SYSMODs, the REWORK level is yyyyddd, which is the year followed by the Julian date (for example, 20081110).
REWORK allows an updated SYSMOD to be automatically received again, as long as it is more recent than the version that has already been received. This takes the place of rejecting the SYSMOD and receiving it again.

Note: If a SYSMOD appears more than once in the SMPPTFIN data set, the first occurrence may be received. However, none of the subsequent versions of the SYSMOD are received, even if their rework level is higher than the one for the first version of the SYSMOD. (Message GIM40001E is issued for each of the subsequent versions of the SYSMOD.)

RFDSNPFX
identifies to SMP/E the prefix used in the relative file data set names for this SYSMOD. SMP/E uses this prefix when allocating data set names for the SYSMOD's relative files during RECEIVE processing.
• This operand can be specified only if the FILES operand is also specified.
• The RFDSNPFX value specified on the MCS statement must match the actual prefix used in the data set names for the associated relative files.
For example, if the names of the relative files created for a SYSMOD start with "IBM", as in IBM.sysmod_id.F1, the header MCS statement for the SYSMOD must specify RFDSNPFX(IBM) so SMP/E knows which prefix to use when allocating the data set names for the SYSMOD's relative files during RECEIVE processing.
• Following standard data set naming conventions, the prefix can be from 1 to 8 alphanumeric or national ($, #, @) characters or a dash (–).
To enable full Security Server protection for tape data sets and to keep the tape header within the 17-character limit (including periods), you should limit the prefix to 1 to 3 characters. If the name exceeds the 17-character limit, only the rightmost 17 characters are written to the tape header label.

sysmod_id
is a unique 7-character name, or SYSMOD ID, for the function. This ID is also called a function modification identifier (FMID). For more information, see “Naming conventions for SYSMODs” on page 517.

Usage Notes
A function cannot contain statements that update elements (++MACUPD, ++SRCUPD, and ++ZAP).
Examples

Here is an example of a function SYSMOD to be created with a SYSMOD ID of JXY1040 that is dependent on function JXY1000. The elements and JCL input data are members of three unloaded partitioned data sets on a tape created using the relative file method:

```euclid
++FUNCTION(JXY1040) /* New function SYSMOD */
FILES(3) /* in RELFILE format. */
RFDSNPFX(IBM) /* RELFILE prefix for IBM. */
++VER(Z038) FMID(JXY1000) /* For MVS function JXY1000. */
++JCLIN RELFILE(1) /* JCLIN in RELFILE 1. */
++MOD(IFBMOD01) /* This module */
  DISTLIB(AOSFB) /* for this DLIB */
  RELFILE(2) /* is in RELFILE 2. */
++MOD(IFBMOD02) /* This module */
  DISTLIB(AOSFB) /* for this DLIB */
  RELFILE(2) /* is in RELFILE 2. */
++MAC(IFBMAC01) /* This macro */
  DISTLIB(AOSMAC) /* for this DLIB */
  RELFILE(3) /* is in RELFILE 3. */
```
Hierarchical file system element MCS

The hierarchical file system element MCSs describe elements located in a UNIX file system. Hierarchical file system elements can have any of the following characteristics:

- The record format (RECFM) must be F, FA, FM, FB, FBA, FBM, V, VA, VM, VB, VBA, or VBM.
- Elements with variable-length records cannot contain spanned records.
- The maximum LRECL is 32,654.
- The records can be numbered or unnumbered.

There are MCSs to replace hierarchical file system elements, just as there are MCSs to replace other types of elements. (There are no MCSs to update hierarchical file system elements.) Table 4 shows the MCSs that can be used for hierarchical file system elements. Table 5 on page 27 shows the xxx values that can be used when the MCS indicates the language being supported.

**Table 4. MCS statements for hierarchical file system elements.** If an element is provided in only one language, the xxx can be left off the MCS. If an element is provided in more than one language, replace the xxx with the appropriate value from Table 5 on page 27.

<table>
<thead>
<tr>
<th>MCS</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>++AIX1 through ++AIX5</td>
<td>Elements to be used by an AIX® client.</td>
</tr>
<tr>
<td>++CLIENT1 through ++CLIENT5</td>
<td>Elements to be used by any client (intended for clients not described by other elements types).</td>
</tr>
<tr>
<td>++HFSxxx</td>
<td>Generic hierarchical file system element (data not covered by other types)</td>
</tr>
<tr>
<td>++OS21 through ++OS25</td>
<td>Elements to be used by an OS/2 client.</td>
</tr>
<tr>
<td>++SHELLSCR</td>
<td>UNIX shell script elements.</td>
</tr>
<tr>
<td>++UNIX1 through ++UNIX5</td>
<td>Elements to be used by a UNIX client.</td>
</tr>
<tr>
<td>++WIN1 through ++WIN5</td>
<td>Elements to be used by a Windows client.</td>
</tr>
</tbody>
</table>
Hierarchical file system element MCS

Table 5. National language identifiers used for language-unique elements

<table>
<thead>
<tr>
<th>Value</th>
<th>Language</th>
<th>Value</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARA</td>
<td>Arabic</td>
<td>HEB</td>
<td>Hebrew</td>
</tr>
<tr>
<td>CHS</td>
<td>Simplified Chinese</td>
<td>ISL</td>
<td>Icelandic</td>
</tr>
<tr>
<td>CHT</td>
<td>Traditional Chinese</td>
<td>ITA</td>
<td>Italian (Italy)</td>
</tr>
<tr>
<td>DAN</td>
<td>Danish</td>
<td>ITS</td>
<td>Italian (Switzerland)</td>
</tr>
<tr>
<td>DES</td>
<td>German (Switzerland)</td>
<td>JPN</td>
<td>Japanese</td>
</tr>
<tr>
<td>DEU</td>
<td>German (Germany)</td>
<td>KOR</td>
<td>Korean</td>
</tr>
<tr>
<td>ELL</td>
<td>Greek</td>
<td>NLB</td>
<td>Dutch (Belgium)</td>
</tr>
<tr>
<td>ENG</td>
<td>English (United Kingdom)</td>
<td>NLD</td>
<td>Dutch (Netherlands)</td>
</tr>
<tr>
<td>ENP</td>
<td>Uppercase English</td>
<td>NOR</td>
<td>Norwegian</td>
</tr>
<tr>
<td>ENU</td>
<td>English (United States)</td>
<td>PTB</td>
<td>Portuguese (Brazil)</td>
</tr>
<tr>
<td>ESP</td>
<td>Spanish</td>
<td>PTG</td>
<td>Portuguese (Portugal)</td>
</tr>
<tr>
<td>FIN</td>
<td>Finnish</td>
<td>RMS</td>
<td>Rhaeto-Romanic</td>
</tr>
<tr>
<td>FRA</td>
<td>French (France)</td>
<td>RUS</td>
<td>Russian</td>
</tr>
<tr>
<td>FRB</td>
<td>French (Belgium)</td>
<td>SVE</td>
<td>Swedish</td>
</tr>
<tr>
<td>FRC</td>
<td>French (Canada)</td>
<td>THA</td>
<td>Thai</td>
</tr>
<tr>
<td>FRS</td>
<td>French (Switzerland)</td>
<td>TRK</td>
<td>Turkish</td>
</tr>
</tbody>
</table>

Syntax
The syntax to be used depends on the processing to be performed for the element:
- Adding or replacing the element
- Deleting the element
Hierarchical file system element MCS

Adding or replacing a hierarchical file system element

Hierarchical file system element MCS

- Element
- NAME
  - BINARY
  - TEXT
- DISTLIB(ddname)
- LINK(linkname)
- PARM(character_string)
- RELFILE(number)
- TXLIB(ddname)
- FROMOS(DSN(dsn)NUMBER(n))
- VOL(vol)
- UNIT(unit)
- SHSCRIPT(scriptname)
  - PRE
  - POST
- SYMLINK(symlinkname)
- SYMPATH(sympathname)
- SYSLIB(ddname)
- VERSION(sysmod_id)

Deleting a hierarchical file system element

Hierarchical file system element MCS

- Element
  - NAME
- DELETE
- DISTLIB(ddname)
- VERSION(sysmod_id)

Operands

**BINARY**

indicates that the hierarchical file system copy utility should install the element into a UNIX file system in binary mode. This means that the element is installed in its entirety as a data stream, with no breaks for logical records.

**Notes:**

1. BINARY is mutually exclusive with TEXT.
2. When BINARY is specified on the element MCS, SMP/E sets the BINARY mode indicator in the hierarchical file system element entry. When TEXT is specified on the element MCS, SMP/E sets the TEXT mode indicator in the hierarchical file system element entry.
Hierarchical file system element MCS

If neither BINARY nor TEXT is specified on the element MCS, SMP/E uses the mode indicator in the hierarchical file system element entry to tell the HFS copy utility how to install the element.

If neither BINARY nor TEXT is specified on the element MCS and there is no mode indicator in the hierarchical file system element entry, the HFS copy utility determines how to install the element.

3. SMP/E recommends the appropriate value BINARY or TEXT be specified to ensure that the HFS copy utility uses the correct mode. If no value is specified, the HFS copy utility chooses either binary or text mode based on the RECFM of the element to be copied, and it might choose incorrectly.

DELETE

specifies that the hierarchical file system element and all of its link names and symbolic link names are to be removed from the “target library” (UNIX file system) and the distribution library.

Notes:
1. DELETE is mutually exclusive with all other operands except DISTLIB and VERSION.
2. If the element statement is in a base function, you may want to use the DELETE operand on the ++VER MCS to delete the previous release, rather than on the element statement to delete a specific element.
3. Specification of the DELETE operand results in all link names and symbolic link names of the element being deleted along with the element identified.

DISTLIB

specifies the ddname of the distribution library for the specified hierarchical file system element. During ACCEPT processing, SMP/E installs the hierarchical file system element into the distribution library as a member. (The distribution library must be a PDS or PDSE; it cannot be part of a UNIX file system.)

Notes:
1. DISTLIB must be specified when the hierarchical file system element is first installed.
2. If an element entry already exists in the target zone or distribution zone and the value currently in that entry does not match that specified in the DISTLIB operand, the SYSMOD is not applied or accepted.

element

specifies the type of element. Table 4 on page 26 shows the MCSs used for the various hierarchical file system element types.

FROMDS

identifies the partitioned data set that contains this element.

Note: The FROMDS operand and its DSN, NUMBER, VOL, and UNIT suboperands are included in the MCS generated by the BUILDMCS command. IBM does not intend the FROMDS operand to be used in manually coded MCS.

DSN

specifies the dsname of the FROMDS data set. The specified data set name must conform to standard data set naming conventions and cannot contain parentheses. The maximum length of the entire name is 44 characters (including the periods).

NUMBER

specifies a number that SMP/E is to use when assigning a
name to the SMPTLIB data set associated with this FROMDS data set. (This is similar to the way the relative file number is used in RELFILE processing.)

**VOL**

specifies, for an uncataloged data set, the volume serial number of the volume containing the FROMDS data set. If specified, this volume identifier must be from 1 to 6 alphanumeric characters.

VOL may be omitted for a cataloged data set.

**UNIT**

specifies, for an uncataloged data set, the UNIT type containing the FROMDS data set. If specified, the UNIT value must be from 1 to 8 characters and must conform to standard UNIT naming conventions. SMP/E accepts any nonblank characters specified between the open and close parentheses, up to a maximum length of 8.

UNIT may be omitted for a cataloged data set.

**Note:** FROMDS is mutually exclusive with DELETE, RELFILE, and TXLIB.

**LINK**

specifies the alternative names by which this hierarchical file system element can be known in a UNIX file system. The full name is produced by concatenating the specified linkname with the UNIX file system directory identified by the SYSLIB subentry. Each linkname is passed to the HFS copy utility as an execution parameter.

**Notes:**

1. The linkname can be from 1 to 1023 characters.
2. A linkname can be enclosed in single apostrophes ('). A linkname **must** be enclosed in single apostrophes if any of the following is true:
   - The linkname contains lowercase alphabetic characters.
   - The linkname contains a character that is not uppercase alphabetic, numeric, national ($, #, @), slash (/), plus (+), hyphen, period, or ampersand (&).
   - The linkname spans more than one line in the control statement.

   The single apostrophes used to enclose a linkname (the delimiters) do not count as part of the 1023-character limit.
3. Any apostrophes specified as part of a linkname (not the delimiters) must be doubled.

   Double apostrophes count as two characters in the 1023-character limit.
4. The linkname can include characters X'40' through X'FE'.
5. LINK values are saved and passed to the HFS copy utility as follows:
   - If LINK is specified on the element MCS, any values previously saved in the element entry are overlaid.
   - If LINK is not specified on the element MCS and saved values exist in the hierarchical file system element entry, the saved values are passed to the HFS copy utility as execution parameters.

   If LINK is not specified on the element MCS and there are no saved values in the hierarchical file system element entry, no linknames are passed to the HFS copy utility.
Hierarchical file system element MCS

**name**
specifies the name of the hierarchical file system element member. The name can contain any uppercase alphabetic, numeric, or national ($, #, @) character and can be 1 to 8 characters long.

**PARM**
specifies a character string that is to be passed to the hierarchical file system copy utility as an execution-time parameter. (The values that can be specified on the PARM operand, such as PATHMODE, are those accepted by the BPXCOPY utility. See [z/OS UNIX System Services Command Reference](z/OS UNIX System Services Command Reference) for a description of BPXCOPY and the values it accepts.) The maximum length of this character string is 300 bytes of nonblank data. If any blanks are specified in the PARM value, they are deleted by SMP/E during processing and do not count toward the 300-byte maximum.

**Notes:**
1. PARM is an optional operand.
2. The character string can be entered free-form, without regard to blanks (which are compressed out of the string), and can span multiple 80-byte records.
3. If parentheses are specified in the PARM value, there must always be a pair (left and right); otherwise, the results are unpredictable.
4. PARM values are saved and passed to the HFS copy utility as follows:
   • If PARM is specified on the element MCS, any values previously saved in the hierarchical file system element entry are overlaid.
   • If PARM is not specified on the element MCS and saved values exist in the hierarchical file system element entry, the saved values are passed to the HFS copy utility as execution parameters.
   If PARM is not specified on the element MCS and there are no saved values in the hierarchical file system element entry, no parameters are saved from the element MCS or passed from the hierarchical file system element entry to the HFS copy utility.
5. If the UTILITY entry for the HFS copy utility specifies a PARM value, those parameters are passed to the utility in addition to any parameters saved in the hierarchical file system element entry.

**RELFIELD**
identifies which relative file associated with the SYSMOD contains this element. This operand is required if you provide the element in RELFILE format, rather than inline or in a TXLIB data set.

**Notes:**
1. The RELFILE value must be a decimal number from 1 to 9999.
2. RELFILE is mutually exclusive with FROMDS and TXLIB.

**RMID**
specifies the last SYSMOD that replaced this element. This operand can be used only in a service-updated function, and the specified PTF must be integrated into the function.

**SHSCRIPT**
specifies a UNIX shell script, `scriptname`, to be invoked when the element is installed in (or deleted from) a directory of a UNIX file system. `scriptname` can contain any uppercase alphabetic, numeric, or national ($, #, @) character and can be 1 to 8 characters long.
Hierarchical file system element MCS

A shell script is commonly used to complete the installation of an element. For example, if the hierarchical file system element is a TAR or PAX file, you can provide a shell script that performs the necessary steps to restore the file.

`scriptname` must be the first value to follow the SHSCRIPT operand.

You define the shell script to SMP/E through a `++SHELLSCR` statement, which can be within the same SYSMOD as the hierarchical file system element, or within a SYSMOD that was processed previously. When the element is itself a SHELLSCR type, the SHSCRIPT operand must match the name of the element.

You cannot define more than one shell script for an element.

You can follow `scriptname` with either of two optional values, PRE and POST, to specify the point in SMP/E processing when the shell script is to be invoked. The following examples show how you can use the PRE and POST values:

- To run the shell script before the element is copied to a UNIX file system directory, specify:
  `SHSCRIPT(scriptname,PRE)`

- To run the shell script after the element is copied to a UNIX file system directory, specify:
  `SHSCRIPT(scriptname,POST)`

  or specify no value after `scriptname` to use POST by default.

- To run the shell script both before and after the element is copied to a UNIX file system directory, specify:
  `SHSCRIPT(scriptname,PRE,POST)`

  If you do not specify a PRE or POST value for the shell script, SMP/E invokes the shell script after the element is installed in the directory.

When the element is a shell script, PRE is not valid. Also, you cannot specify the SHSCRIPT operand with the DELETE operand.

SYMLINK specifies a list of one or more symbolic links, which are file names that can be used as alternate names for referring to this element in a UNIX file system. Each linkname listed here is associated with a pathname listed in the SYMPATH operand. For more information about how the linknames and pathnames are associated, see the description of the SYMPATH operand and “Example 3: Packaging a SYSMOD with a symbolic link” on page 36.

The SYMLINK value specified should be a relative path value (that is, it does not start with a slash ["/"]). When the symbolic link is created, it is created relative to the pathname of the element's SYSLIB ddname.

SYMLINK must be specified if the SYMPATH operand is specified, otherwise it must be omitted.

A symbolic linkname can be from one to 1023 characters. Any characters in the range X'40' through X'FE' may be specified.

The value may be enclosed in single apostrophes. It **must** be enclosed in single apostrophes if:

- it is continued to the next line in the MCS, or

- it contains a character that is not uppercase alphabetic, numeric, national ($, #, @), slash (/), plus (+), hyphen, period, or ampersand (&).

If an apostrophe is a part of the symbolic linkname and is not a delimiter, then it must be doubled. These two apostrophes count as two characters against the...
1023 character limit for a symbolic linkname. The single apostrophes used to
enclose a symbolic linkname do not count against the 1023 character limit.

**SYMPATH**
specifies a list of one or more pathnames that are associated with symbolic
links identified by the SYMLINK operand. The first pathname in the
SYMPATH operand is associated with the first symbolic link in the SYMLINK
operand, the second pathname with the second symbolic link, and so on. If
there are more symbolic links listed than there are pathnames, then the last
listed pathname is used for the remaining symbolic links. If more pathnames
are specified than symbolic linknames, then the excess pathnames (at the end
of the list) are ignored.

The SYMPATH value specified should be a relative path value. When the
symbolic link is accessed, the system assumes the destination of that link (the
SYMPATH value) is relative to that symbolic link (the SYMLINK value). For
more information about how the pathnames and linknames are associated, see
"Example 3: Packaging a SYSMOD with a symbolic link" on page 36.

**SYMPATH** must be specified if the **SYMLINK** operand is specified, otherwise
it must be omitted.

A symbolic pathname can be one to 1023 characters. Any characters in the
range X'40' through X'FE' may be specified.

The value may be enclosed in single apostrophes. It must be enclosed in single
apostrophes, if:
- it is continued to the next line in the MCS, or
- it contains a character that is not uppercase alphabetic, numeric, national ($,
  #, @), slash (/), plus (+), hyphen, period, or ampersand (&).

If an apostrophe is a part of the symbolic pathname and is not a delimiter,
then it must be doubled. These two apostrophes count as two characters
against the 1023 character limit for a symbolic pathname. The single
apostrophes used to enclose a symbolic linkname do not count against the 1023
character limit.

**SYSLIB**
specifies the ddname of the “target library” within the UNIX file system for
the element.

During APPLY processing, the HFS copy utility installs the hierarchical file
system element into a UNIX file system. During RESTORE processing, the HFS
copy utility copies the hierarchical file system element from the distribution
library member into a UNIX file system.

**Note:** **SYSLIB** must be specified when the hierarchical file system element is
first installed.

**TEXT**
indicates that the hierarchical file system copy utility should install the element
into a UNIX file system in text mode. This means that the element is installed
with breakpoints for logical records.

**Notes:**
1. TEXT is mutually exclusive with BINARY.
2. When TEXT is specified on the element MCS, SMP/E sets the TEXT mode
   indicator in the hierarchical file system element entry. When BINARY is
   specified on the element MCS, SMP/E sets the BINARY mode indicator in
   the hierarchical file system element entry.
Hierarchical file system element MCS

If neither BINARY nor TEXT is specified on the element MCS, SMP/E uses the mode indicator in the hierarchical file system element entry to tell the HFS copy utility how to install the element.

If neither BINARY nor TEXT is specified on the element MCS and there is no mode indicator in the hierarchical file system element entry, the HFS copy utility determines how to install the element.

3. SMP/E recommends the appropriate value BINARY or TEXT be specified to ensure that the HFS copy utility uses the correct mode. If no value is specified, the HFS copy utility chooses either binary or text mode based on the RECFM of the element to be copied, and it might choose incorrectly.

TXLIB

is the ddname of the partitioned data set containing the hierarchical file system element. This operand is required if the hierarchical file system element is provided in a data set that the users have access to, rather than inline or in RELFILE format.

Notes:
1. SMPTLIB cannot be used as a value on the TXLIB operand.
2. TXLIB is mutually exclusive with FROMDS and RELFILE.

VERSION

specifies one or more function SYSMODs that currently contain the element. The function containing the element MCS takes ownership of the element from the specified functions.

When VERSION is specified on an element statement, it overrides any VERSION operand values that might be specified on the ++VER MCS.

Usage Notes

• If the hierarchical file system element is packaged inline, it must immediately follow the hierarchical file system element MCS and must not contain any records starting with ++. Neither FROMDS, nor RELFILE, nor TXLIB can be specified on the hierarchical file system element MCS.

• To be packaged inline, a hierarchical file system element must contain fixed-block–80 records. If the original format of the element is not fixed-block–80 records, you can use GIMDTS to transform the element into the required format before packaging it. Later, when SMP/E installs the element, it is changed back to its original format. For more information about using GIMDTS, see "GIMDTS: Data transformation service routine" on page 450.

• If the hierarchical file system element is packaged in a TXLIB data set, the ddname specified in the TXLIB operand is required during APPLY and ACCEPT processing.

• For information about elements packaged in RELFILE format, see "z/OS Packaging Rules.

• A ++HFS MCS can be used to supply a pre-built program object to be placed into a UNIX file system. A user can do an OGET for an existing program object in the file system to cause the program object to be placed into an MVS data set as fixed length records. This data can then be packaged as a ++HFS element with the BINARY operand. The HFS copy utility can then copy the element into the file system as a binary entity, which can then be executed.

Examples

The following examples are provided to help you use the hierarchical file system element MCS:
Hierarchical file system element MCS

- "Example 1: Packaging a hierarchical file system element in a function"
- "Example 2: Packaging a renamed hierarchical file system Element"
- "Example 3: Packaging a SYSMOD with a symbolic link" on page 36

Example 1: Packaging a hierarchical file system element in a function

Here is an example of a SYSMOD containing a hierarchical file system element statement to install your element and have SMP/E track any changes to it. The element:

- Uses the Japanese language
- Is to be installed in BINARY mode
- Can also be known by its linkname “USERHFS”
- Is to be processed by shell script “UNTAR” after the element is copied to a UNIX file system directory.

The element can be packaged in a function, as shown below:

```
++FUNCTION(MYHFSEL) /* Function. */.
++VER(Z038) /* For MVS SRELs. */.
++HFSJPN(HFSELEM1) /* Install this element */
   TXLIB(NEWHFS) /* from this TXLIB */
   DISTLIB(ABPXLIB) /* into this DLIB and */
   SYSLIB(BPXLIB1) /* this target library. */
   BINARY /* Use BINARY mode. */
   LINK('../USERHFS') /* Define linkname. */
   SHSCRIPT(UNTAR,POST) /* Untar the file with the UNTAR exec. */.
```

To install the function, you need to specify DDDEF entries or DD statements for the TXLIB, DISTLIB, and SYSLIB data sets. Remember that the SYSLIB data set is actually a pathname in a UNIX file system. For an example of defining a pathname in a DDDEF entry, see "Example 10: Defining pathnames in a UNIX file system" on page 209.

Example 2: Packaging a renamed hierarchical file system Element

Suppose that, for some reason, you need to rename HFSELEM1, which was introduced in "Example 1: Packaging a hierarchical file system element in a function." The new name is to be HFSELEMX. You do not need to change anything else about the hierarchical file system element. Here is an example of a SYSMOD containing the hierarchical file system element statements needed to package this renamed hierarchical file system element:

```
++USERMOD(USR0001) /* User modification */.
++VER(Z038) FMID(MYHFSEL) /* to user application. */.
++HFSJPN(HFSELEM1) /* Delete the original */
   DELETE /* HFS element. */
++HFSJPN(HFSELEMX) /* Add the renamed element */
   TXLIB(NEWHFS) /* from this TXLIB */
   DISTLIB(ABPXLIB) /* into this DLIB and */
   SYSLIB(BPXLIB1) /* this target library. */
   TEXT /* Use TEXT mode. */
   LINK('../USERHFS') /* Define linkname. */
```

To install the USERMOD, you need to specify DDDEF entries or DD statements for the TXLIB, DISTLIB, and SYSLIB data sets. Remember that the SYSLIB data set is actually a pathname in a UNIX file system. For an example of defining a pathname in a DDDEF entry, see "Example 10: Defining pathnames in a UNIX file system" on page 209.
Hierarchical file system element MCS

You must also ensure that the renamed hierarchical file system element is in the TXLIB library used to provide SMP/E with the element.

**Example 3: Packaging a SYSMOD with a symbolic link**

Here is an example of a SYSMOD containing a hierarchical file system element statement to install your element with a symbolic link. The element:
- Is to be installed in TEXT mode
- Can be known by its linkname “/usr/lpp/gskssl/include/gskssl.h”
- Can also be known by its symbolic link “/usr/include/gskssl.h”.

The element can be packaged in a function, as shown below:

```plaintext
++FUNCTION(SAMPLE) /* Function. */.
++VER(Z838) /* For MVS SRELS. */.
++HFS(GSKAH010) /* Install this element */.
RELFILE(2) /* from a RELFILE */.
DISTLIB(AGSKHFS) /* into this DLIB and */.
SYSLIB(SGSKHFS) /* this target library. */.
LINK('..include/gskssl.h') /* Define linkname. */.
SYMLINK('..include/gskssl.h') /* Define symbolic link */.
SYMPATH('..lpp/gskssl/include/gskssl.h') /* Define path name. */.
PARM(PATHMODE(0,6,4,4)) /* HFS copy utility parameters */.
TEXT /* Use TEXT mode. */.
```

To install the function, you need to specify DDDEF entries or DD statements for the DISTLIB and SYSLIB data sets. Remember that the SYSLIB data set is actually a pathname in a UNIX file system. In this example, the DDDEF for SYSLIB must be defined as follows:

```plaintext
UCLIN.
ADD DDDEF(SGSKHFS) PATH('/service/usr/lpp/gskssl/IBM').
ENDUCLIN.
```

When SMP/E installs file GSKAH010 into the directory specified on the given DDDEF entry, the LINK and SYMLINK values will be resolved relative to the DDDEF directory. The file will then be known by the following absolute path names in a UNIX file system:

<table>
<thead>
<tr>
<th>Source</th>
<th>Resolved absolute pathname</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMP/E element name</td>
<td>/service/usr/lpp/gskssl/IBM/GSKAH010</td>
</tr>
<tr>
<td>LINK name</td>
<td>/service/usr/lpp/gskssl/include/gskssl.h</td>
</tr>
<tr>
<td>SYMLINK name</td>
<td>/service/usr/include/gskssl.h</td>
</tr>
</tbody>
</table>

The contents of a symbolic link (SYMLINK) is its associated SYMPATH. In this case, the contents of the symbolic link is “/service/usr/lpp/gskssl/include/gskssl.h”. When a symbolic link is resolved, its content is resolved relative to the symbolic link itself. That is, the SYMPATH value is relative to the SYMLINK value. Therefore, in this case, the SYMPATH value resolves to the LINK name, “/service/usr/lpp/gskssl/include/gskssl.h”.


++HOLD MCS

The ++HOLD MCS identifies a SYSMOD to be placed into exception SYSMOD status (signifying that special SMP/E processing is required before it can be applied or accepted). ++HOLD statements can occur within a SYSMOD (internal HOLDDATA), or they can be read directly from the SMPHOLD file during RECEIVE processing (external HOLDDATA). For additional information about processing the ++HOLD statements, see "SMP/E Commands".

Syntax

++HOLD MCS

++HOLD(sysmod_id) FMID(fmrid) REASON(reason_id)

ERROR

FIXCAT CATEGORY(category)

SYSTEM

USER

CLASS(class)

DATE(yyddd)

COMMENT(text)

COMMENT( Enhanced HOLDDATA text )

SYSTEM reason IDs used by IBM:

- ACTION
- AO
- DB2BIND
- DDFDEF
- DELETE
- DEP
- DOC
- DOWNLD
- EC
- ENH
- EXIT
- EXRF
- FULLGEN
- IODGEN
- IPL
- MSGSKEL
- MULTSYS
- RESTART
++HOLD MCS

Enhanced HOLDDATA text used by IBM:

| SMRTDATA(CHGDT(yymmdd)) | FIX(sysmod_id) | SYMP(symptoms) |

Operands

CATEGORY

a 1- to 64-character string that specifies a Fix Category. There can be one or more Fix Category values. A Fix Category value associates the reason ID APAR to a particular category of fixes. A Fix Category might be used to identify a group of APAR fixes required to support a particular hardware device, or to provide a particular software capability, similar to how a Preventive Service Planning Bucket (PSP-Bucket) identifies a group of APARs. Examples of Fix Categories used by IBM are

IBM.Device.2094.z/OS

and

IBM.HealthChecker.

The Fix Category values are used during RECEIVE command processing to create source IDs for the SYSMODs that will resolve the specified reason ID APAR. During APPLY and ACCEPT command processing the Fix Category values are used to determine if the HOLDDATA is applicable to the current command by comparing the values in the HOLDDATA to those in the active Fix Category interest list. The active Fix Category interest list is specified on the APPLY and ACCEPT command, or within the active OPTIONS entry.

A category value might contain any nonblank character in the range X’41’ - X’FE’ except single quotation mark (‘), asterisk (*), percent (%), comma (,), left parenthesis (()), and right parenthesis ()).

CLASS

a 1- to 7-character string indicating an alternative reason to release an exception SYSMOD for processing. A class name is specified along with a reason ID to identify a condition when the reason ID need not be resolved. The same class name can be specified on any number of +++HOLD statements in any number of SYSMODs.

These are the specific values currently used by IBM:

<table>
<thead>
<tr>
<th>Class</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERREL</td>
<td>The SYSMOD is held for an error reason ID but should be installed anyway. IBM has determined that the problem the SYSMOD resolves is significantly more critical than the error reflected by the holding APAR.</td>
</tr>
<tr>
<td>HIPER</td>
<td>The SYSMOD is held with a hold class of HIPER (High Impact)</td>
</tr>
<tr>
<td>PE</td>
<td>The SYSMOD is held with a hold class of “PTF in Error”.</td>
</tr>
<tr>
<td>UCLREL</td>
<td>UCLIN needed for the SYSMOD has been handled by IBM and no longer requires your attention.</td>
</tr>
<tr>
<td>YR2000</td>
<td>Identifies PTFs that provide Year 2000 function, or fix a Year 2000-related problem.</td>
</tr>
</tbody>
</table>
++HOLD MCS

For additional information, see "Naming conventions for HOLD reason IDs and HOLD classes" on page 514.

COMMENT(text)
free-form text to be used to describe the problem identified by the REASON operand. The comments supplied are associated only with the reason ID supplied. The comments can be in single-byte characters (such as English alphanumeric characters) or double-byte characters (such as Kanji).

COMMENT(Enhanced HOLDDATA)
contains IBM–supplied Enhanced HOLDDATA, as follows:

SMRTDATA
indicates that the COMMENT operand contains Enhanced HOLDDATA.

CHGDT specifies the date that the ++HOLD statement was last updated with Enhanced HOLDDATA. The date is specified as yyymmdd, where yy is the last two digits of the year, mm is the month and dd is the date of the month.

FIX identifies the SYSMOD that resolves the held SYSMOD.

SYMP contains a description of the hold. The description might include symbols representing symptoms of the problem, a description of the problem in text form, or a combination of both. The symbols that represent symptoms identify how the problem affects the held SYSMOD.

Here is a list of symbols that might appear and their meanings. For a complete list of the symbols, see Enhanced HOLDDATA for z/OS and OS/390 (http://service.software.ibm.com/holdata/390holddata.html).

DAL Data loss
DST2007 Daylight Saving Time support
EURO99 Euro currency symbol support
FUL Major function loss
IPL System outage (Requires IPL)
PRF Performance
PRV Pervasive
SYSPLEX SYSPLEX data sharing
XSYS Cross-system migration, compatibility, or toleration
YR2000 Year 2000

The SYMP field can be no more than 30 characters in length. Parentheses can be used within the field, but they must be matched pairs.

DATE specifies the date that the ++HOLD statement was generated. The date is specified as yyddd, where yy is the last two digits of the year and ddd is the Julian date.

ERROR, FIXCAT, SYSTEM, or USER specifies the hold category into which the SYSMOD is to be put. At least one of the values must be specified.

ERROR An APAR reported an error in the SYSMOD. The SYSMOD
should not be applied or accepted until the APAR is resolved. A PTF held for this reason is also called a program error PTF, or PE-PTF. SMP/E automatically resolves the APAR and allows the SYSMOD to be applied or accepted when a SYSMOD that either matches or supersedes the APAR is also applied or accepted. Error holds can be read only from the SMPHOLD data set.

**Note:** ERROR can also be specified as ERR.

**FIXCAT**

An APAR provides a fix for the held SYSMOD and the fix is associated with one or more Fix Categories. It is optional whether the APAR will affect processing for the held SYSMOD, based on the APAR's Fix Categories and the Fix Categories of interest specified by the user. If any one or more Fix Categories for the APAR match any of those of interest to the user, then the held SYSMOD will not be applied or accepted until the APAR is resolved. The APAR is resolved when a SYSMOD that matches the APAR name, or a SYSMOD that supersedes the APAR, is applied or accepted. FIXCAT holds can be read only from the SMPHOLD data set.

See the FIXCAT operand for APPLY and ACCEPT command processing for details of specifying the Fix Categories of interest.

**SYSTEM**

Special action outside normal SMP/E processing is required for the SYSMOD. Examples are SYSMODs requiring a SYSGEN after they are installed, or SYSMODs requiring the installation of an associated engineering change (EC) level. System holds can appear in the SYSMOD itself or in the SMPHOLD data set.

**Note:** SYSTEM can also be specified as SYS.

**USER**

The SYSMOD requires special processing because of a decision you have made. User holds can be read only from the SMPHOLD data set.

**FMID**

specifies the FMID to which the held SYSMOD is applicable. For external HOLDDATA (a ++HOLD statement not within a SYSMOD), this information allows SMP/E to receive only those statements associated with FMIDs defined in the user's global zone. This operand is required.

**REASON**

a 1- to 7-character string used to help users identify the reason why the SYSMOD is being put into exception SYSMOD status. The reason IDs that can be specified depend on the type of hold.

**Note:** A SYSMOD can contain only one ++HOLD MCS for each required reason ID.

- An error reason ID is the APAR number that caused the SYSMOD to be placed in exception status.
- A system reason ID identifies some special processing the SYSMOD requires. Although SMP/E accepts any 1- to 7-character alphanumeric string as a system reason ID, these are the values currently used by IBM on ++HOLD statements:

<table>
<thead>
<tr>
<th>ID</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
++HOLD MCS

**ACTION**  The SYSMOD needs special handling before or during APPLY processing, ACCEPT processing, or both.

**AO**  The SYSMOD may require action to change automated operations procedures and associated data sets and user exits in products or in customer applications. The PTF cover letter describes any changes (such as to operator message text, operator command syntax, or expected actions for operator messages and commands) that can affect automation routines.

**DB2BIND**  A DB2® application REBIND is required for the SYSMOD to become effective.

**DDDEF**  Data set changes or additions as required.

**DELETE**  The SYSMOD contains a ++DELETE MCS, which deletes a load module from the system.

**DEP**  The SYSMOD has a software dependency.

**DOC**  The SYSMOD has a documentation change that should be read before the SYSMOD is installed.

**DOWNLD**  Code that is shipped with maintenance that needs to be downloaded.

**DYNACT**  The changes supplied by the SYSMOD may be activated dynamically without requiring an IPL. The HOLD statement describes the instructions required for dynamic activation. If those instructions are not followed, then an IPL is required for the SYSMOD to take effect.

**EC**  The SYSMOD needs a related engineering change.

**ENH**  The SYSMOD contains an enhancement, new option or function. The HOLD statement provides information to the user regarding the implementation and use of the enhancement.

**EXIT**  The SYSMOD contains changes that may affect a user exit. For example, the interface for an exit may be changed, an exit may need to be reassembled, or a sample exit may be changed.

**EXRF**  The SYSMOD must be installed in both the active and the alternative Extended Recovery Facility (XRF) systems at the same time to maintain system compatibility. (If you are not running XRF, you should bypass this reason ID.)

**FULLGEN**  The SYSMOD needs a complete system or subsystem generation to take effect.

**IOGEN**  The SYSMOD needs a system or subsystem I/O generation to take effect.

**IPL**  The SYSMOD requires an IPL to become effective. For example, the SYSMOD may contain changes to LPA or NUCLEUS, the changes may require a CLPA, or a failure to perform an IPL might lead to catastrophic results, such as could be caused by activation of a partial fix.
++HOLD MCS

**Note:** If you plan to perform an IPL with CLPA after the SYSMOD has been applied, then no further investigation of the HOLD is required; simply bypass the IPL reason ID. However, if you are not planning to perform an IPL with CLPA, then the details of the HOLD statement must be investigated to determine what kind of actions are required to activate the SYSMOD.

**MSGSKEL**  This SYSMOD contains message changes that must be compiled for translated versions of the message changes to become operational on extended TSO consoles.

If you want to use translated versions of the messages, you must run the message compiler once for the library containing the English message outlines, and once for each additional language you want to be available on your system. For details, see [z/OS MVS Planning: Operations](reference).

If you want to use only the English version of the messages, you do not need to run the message compiler. You should bypass this reason ID.

**MULTSYS**  Identifies fixes that need to be applied to multiple systems, in one of three cases: preconditioning, coexistence, or exploitation.

**RESTART**  To become effective, the SYSMOD requires a special subsystem restart operation. The HOLD statement contains information regarding the required restart actions.

- A user reason ID is defined by the user.
- A Fix Category reason ID is the SYSMOD ID for the APAR that caused the SYSMOD to be placed into exception status. The APAR reason ID is associated to one or more categories of fixes.

For additional information, see "Naming conventions for HOLD reason IDs and HOLD classes" on page 514.

**RESOLVER**  Identifies the SYSMOD that resolves the held SYSMOD. More specifically, the resolving SYSMOD supersedes the reason ID APAR that caused the SYSMOD to be held.

**sysmod_id**  If the ++HOLD is being processed from the SMPHOLD data set, `sysmod_id` specifies the name of the SYSMOD to be placed into exception SYSMOD status.

If the ++HOLD is internal to a SYSMOD, then `sysmod_id` specifies either the SYSMOD that contains the ++HOLD or the originating SYSMOD that contained the ++HOLD and is now superseded by the current SYSMOD.

This operand is required.

**Usage Notes**

- If the text within the COMMENT operand contains parentheses, they must be in matched pairs. For example:
  ```
  COMMENT ( ADD 'PARM(COND(5))' ON THE EXEC )
  ```
  This works, but the following statements result in an error:
  ```
  COMMENT ( ADD OPENING PARENTHESES '(' )
  ```
The following restrictions apply only to ++HOLD MCSs packaged within a SYSMOD:

- The only HOLD type allowed is SYSTEM.
- ++HOLD statements must be placed after all ++VER and ++IF statements and before the first ++JCLIN or element statement.
- The SYSMOD ID on the ++HOLD MCS must match either the SYSMOD containing the ++HOLD MCS, or the originating SYSMOD that contained the ++HOLD MCS and is now superseded by the current SYSMOD.

Examples

The following examples are provided to help you use the ++HOLD MCS:

Example 1: Noting a special documentation change

Here is an example of a SYSMOD containing a ++HOLD statement. PTF UZ12345, which is applicable to function FXY1040, introduces a documentation change that the system operator should know about.

```
++PTF(UZ12345) /* PTF modification. */.
++VER(Z038) FMID(FXY1040) /* For MVS product FXY1040. */.
++HOLD (UZ12345) /* Hold this PTF */.
FMID(FXY1040) /* for this function */.
SYSTEM /* for system processing and*/.
REASON(DOC) /* for doc change. */.
COMMENT(this PTF changes operator console message xxx1233. A reply of U is required to this message ).
++MOD(IFBMOD01) /* Change this module */.
DISTLIB(AOS12) /* in this DLIB. */.
```

Example 2: Marking a PTF that is in error

When IBM discovers errors in PTFs it has already shipped, it provides information about these program error PTFs (PE-PTFs) in RETAIN®. To make sure these PTFs are held, you can use the RETAIN information to build ++HOLD statements for the PTFs.

Note: In this example, because only ++HOLD statements are being shown, none of them are underlined.

```
++HOLD (UZ12345) /* Hold this PTF */.
FMID(FXY1040) /* for this function */.
ERROR /* for APAR fix. */.
REASON(AZ00001) /* APAR is AZ00001. */.
COMMENT(this APAR causes loop) /* */.
++HOLD (UZ12345) /* Hold this PTF */.
FMID(FXY1040) /* for this function */.
ERROR /* for APAR fix. */.
REASON(AZ00002) /* APAR is AZ00002. */.
COMMENT(this APAR causes OC4) /* */.
++HOLD (UZ12346) /* Hold this PTF */.
FMID(FXY1040) /* for this function */.
ERROR /* for APAR fix. */.
REASON(AZ00003) /* APAR is AZ00003. */.
COMMENT(incorrect output on console) /* */.
++HOLD (UZ12347) /* Hold this PTF */.
FMID(FXY1040) /* for this function */.
ERROR /* for APAR fix. */.
REASON(AZ00004) /* APAR is AZ00004. */.
/* Integrity - no comment. */.
```
Example 3: Specifying a hold class

Here is an example of a SYSMOD containing a ++HOLD statement for a PTF requiring a change that applies only if a Year 2000-related fix is desired.

```plaintext
++PTF(UZ12345) /* PTF modification. */
++VER(Z038) FMID(FXY1040) /* For MVS product FXY1040. */
   /* No prerequisites. */
++HOLD (UZ12345) /* Hold this PTF */
   FMID(FXY1040) /* for this function */
   ERROR /* for APAR fix. */
   REASON(AQ19558) /* APAR is AQ19558. */
   COMMENT(SMRTDATA(FIX(UQ21725)
   SYMP(YR2000) CHGDT(981019))
   CLASS(YR2000) /* This change is required */
   /* only for Yr 2000 support. */
++MOD(IFBMOD01) /* Change this module */
   DISTLIB(AOS12) /* in this DLIB. */
```

Example 4: Identifying APARs and their associated fix categories

Here are some examples of using the ++HOLD FIXCAT MCS to identify APARs and their associated fix categories.

```plaintext
++HOLD(HBB7730) /* Held FMID */
   FMID(HBB7730)
   FIXCAT /* Associates an APAR to a fix category */
   REASON(AA15968) /* The APAR */
   CATEGORY(IBM.Device.2094) /* A fix category */
   RESOLVER(UA27113) /* The fixing PTF */
```
++IF MCS

The ++IF MCS specifies requisites that must be installed with a SYSMOD if a certain function is installed. If the specified function is installed in the zone that SMP/E is trying to install this SYSMOD into, the requisite must also be installed in that zone; otherwise SMP/E does not install the SYSMOD containing the ++IF MCS. If the specified function is not yet installed in the zone where SMP/E is trying to install this SYSMOD, SMP/E saves the information from the ++IF MCS in case the specified function is installed later.

++IF statements are interpreted, reformatted, and placed in the target zone data set during APPLY processing and in the distribution zone data set during ACCEPT processing.

++IF statements can describe dependencies between two products, even if those products have different SRELs and are defined in different zones. The information from these statements is used by the REPORT command to identify cross-zone requisites—that is, conditional requisites needed in one zone because of ++IF statements in a SYSMOD installed in another zone.

A ++IF statement is associated with the ++VER MCS preceding it in the SYSMOD. Multiple ++IF statements can be specified following each ++VER MCS.

Syntax

```
++IF MCS

++IF FMID(sysmod_id) THEN REQ(sysmod_id) .
```

Operands

**FMID**

specifies the function that is to be checked to determine whether it is installed in one of the following:
- The target libraries (for APPLY processing)
- The distribution libraries (for ACCEPT processing)

This operand is required. It is not satisfied by superseded FMIDs.

**REQ**

specifies the SYSMODs that are needed if the function SYSMOD specified on the FMID operand of the ++IF MCS is installed. This operand is required.

**THEN**

specifies that the requisite for the ++IF MCS follows.

Usage Notes

- The operands must be specified in the order shown in the syntax.
- The function specified for FMID must be different from the SYSMOD ID on the header statement.
Examples

PTF UZ00004 contains service for elements belonging to function FXY1040. If function FXY1050 is installed in the same zone, PTF UZ00005 must also be installed in that zone in order for PTF UZ00004 to be installed successfully. If function FXY1050 is not presently installed, PTF UZ00005 is not required; however, SMP/E saves the ++IF MCS in case function FXY1050 is installed in the future. Then, when FXY1050 is installed, PTF UZ00005 is considered an unsatisfied conditional requisite that must also be installed.

Here is an example of a SYSMOD containing a ++IF statement to define UZ00005 as a conditional requisite:

```
++PTF(UZ00004) /* PTF SYSMOD. */.
++VER(Z038) FMID(FXY1040) /* For MVS product FXY1040. */.
++IF FMID(FXY1050) /* If FXY1050 is on, or */
    THEN /* when it's put on later, */
    REQ(UZ00005) /* UZ00005 is required. */.
++MOD(IFBMOD01) /* Replace this module */.
    DISTLIB(AOSFB) /* in this DLIB. */.
++MACUPD(IFBMAC01) /* Update this macro */.
    DISTLIB(IFBMACS) /* in this DLIB. */.
```
++JAR MCS

The ++JAR MCS describes a Java ARchive (JAR) file. The Java jar command is used to construct and update such files in a UNIX file system. JAR elements can have any of the following characteristics:

- The record format (RECFM) must be F, FA, FM, FB, FBA, FBM, V, VA, VM, VB, VBA, or VBM.
- Elements with variable-length records cannot contain spanned records.
- The maximum LRECL is 32,654.
- The records can be numbered or unnumbered.

There are MCS statements to replace JAR elements, just as there are MCS statements to replace other types of elements. The ++JARUPD MCS can be used to update JAR elements.

Syntax

The syntax to be used depends on the processing to be performed for the element:

- Adding or replacing the element
- Deleting the element
Adding or replacing a JAR element

```
++JAR MCS

++JAR MCS
  +JAR(name)
    +DISTLIB(ddname)
    +JARPARM(character_string)

  +PARM(character_string)
    +RELFILE(number)
    +TXLIB(ddname)

  +LINK(linkname)
    +PARM(character_string)

  +FROMDS
```

Deleting a JAR element

```
++JAR MCS
  +JAR(name)
    +DELETE
    +DISTLIB(ddname)
      +VERSION(sys_id)
```

Operands

**DELETE**

specifies that the JAR element and all of its link names and symbolic link names are to be removed from the “target library” (UNIX file system) and the distribution library.
+++JAR MCS

Notes:
1. DELETE is mutually exclusive with all other operands except DISTLIB and VERSION.
2. If the element statement is in a base function, you may want to use the DELETE operand on the ++VER MCS to delete the previous release, rather than on the element statement to delete a specific element.
3. Specification of the DELETE operand results in all link names and symbolic link names of the element being deleted along with the element identified.

DISTLIB
specifies the ddname of the distribution library for the specified JAR element. During ACCEPT processing, SMP/E installs the JAR element into the distribution library as a member. (The distribution library must be a PDS or PDSE; it cannot be part of a UNIX file system.)

Notes:
1. DISTLIB must be specified when the JAR element is first installed.
2. If an element entry already exists in the target zone or distribution zone and the value currently in that entry does not match that specified in the DISTLIB operand, the SYSMOD is not applied or accepted.

FROMDS
identifies the partitioned data set that contains this element.

Note: The FROMDS operand and its DSN, NUMBER, VOL, and UNIT suboperands are included in the MCS generated by the BUILDMCS command. IBM does not intend the FROMDS operand to be used in manually coded MCS.

DSN specifies the dsname of the FROMDS data set. The specified data set name must conform to standard data set naming conventions and cannot contain parentheses. The maximum length of the entire name is 44 characters (including the periods).

NUMBER specifies a number that SMP/E is to use when assigning a name to the SMPTLIB data set associated with this FROMDS data set. (This is similar to the way the relative file number is used in RELFILE processing.)

VOL specifies, for an uncataloged data set, the volume serial number of the volume containing the FROMDS data set. If specified, this volume identifier must be from 1 to 6 alphanumeric characters. VOL may be omitted for a cataloged data set.

UNIT specifies, for an uncataloged data set, the UNIT type containing the FROMDS data set. If specified, the UNIT value must be from 1 to 8 characters and must conform to standard UNIT naming conventions. SMP/E accepts any nonblank characters specified between the open and close parentheses, up to a maximum length of 8. UNIT may be omitted for a cataloged data set.

Note: FROMDS is mutually exclusive with DELETE, RELFILE, and TXLIB.
**JAR MCS**

**JARPARM**
specifies a character string that is to be passed to the jar command as a command option when updating the JAR file. The maximum length of this character string is 300 bytes of data.

Only those jar command options that require only the option indicator to be specified are supported, such as 0 and M. Options requiring additional input over and above the option indicator are not supported. Examples of unsupported options are the m and -C options.

**Notes:**
1. The character string can include any non-blank characters (characters x'41' through x'FE').
2. The JARPARM value is saved and passed to the jar command as follows:
   - If JARPARM is specified on the element MCS, any value previously saved in the JAR element entry is overlayed and the new value is passed to the jar command.
   - If JARPARM is not specified on the element MCS and a saved value exists in the JAR element entry, then the saved value is passed to the jar command.
   - If JARPARM is not specified on the element MCS and there is no saved value in the JAR element entry, then no value is saved in the element entry and no value is passed to the jar command.

**LINK**
specifies the alternative names by which this JAR element can be known in a UNIX file system. The full name is produced by concatenating the specified linkname with a UNIX file system directory identified by the SYSLIB subentry. Each linkname is passed to the HFS copy utility as an execution parameter.

**Notes:**
1. The linkname can be from 1 to 1023 characters.
2. A linkname can be enclosed in single apostrophes ('). A linkname **must** be enclosed in single apostrophes if any of the following is true:
   - The linkname contains lowercase alphabetic characters.
   - The linkname contains a character that is not uppercase alphabetic, numeric, national ($, #, @), slash (/), plus (+), hyphen, period, or ampersand (&).
   - The linkname spans more than one line in the control statement.
   The single apostrophes used to enclose a linkname (the delimiters) do not count as part of the 1023-character limit.
3. Any apostrophes specified as part of a linkname (not the delimiters) must be doubled.
   Double apostrophes count as two characters in the 1023-character limit.
4. The linkname can include characters X'40' through X'FE'.
5. **LINK** values are saved and passed to the HFS copy utility as follows:
   - If LINK is specified on the element MCS, any values previously saved in the element entry are overlaid.
   - If LINK is not specified on the element MCS and saved values exist in the JAR element entry, the saved values are passed to the HFS copy utility as execution parameters.
   If LINK is not specified on the element MCS and there are no saved values in the JAR element entry, no linknames are passed to the HFS copy utility.
name

specifies the name of the JAR element member. The name can contain any
uppercase alphabetic, numeric, or national ($, #, @) character and can be 1 to 8
characters long.

PARM

specifies a character string that is to be passed to the hierarchical file system
copy utility as an execution-time parameter. (The values that can be specified
on the PARM operand, such as PATHMODE, are those accepted by the
BPXCOPY utility. See z/OS UNIX System Services Command Reference for a
description of BPXCOPY and the values it accepts.) The maximum length of
this character string is 300 bytes of nonblank data. If any blanks are specified
in the PARM value, they are deleted by SMP/E during processing and do not
count toward the 300-byte maximum.

Notes:
1. PARM is an optional operand.
2. The character string can be entered free-form, without regard to blanks
(which are compressed out of the string), and can span multiple 80-byte
records.
3. If parentheses are specified in the PARM value, there must always be a pair
(left and right); otherwise, the results are unpredictable.
4. PARM values are saved and passed to the HFS copy utility as follows:
   • If PARM is specified on the element MCS, any values previously saved
     in the JAR element entry are overlaid.
   • If PARM is not specified on the element MCS and saved values exist in
     the JAR element entry, the saved values are passed to the HFS copy
     utility as execution parameters.
     If PARM is not specified on the element MCS and there are no saved
     values in the JAR element entry, no parameters are saved from the
     element MCS or passed from the JAR element entry to the HFS copy
     utility.
5. If the UTILITY entry for the HFS copy utility specifies a PARM value, those
parameters are passed to the utility in addition to any parameters saved in
the JAR element entry.

RELFILE

identifies which relative file associated with the SYSMOD contains this
element. This operand is required if you provide the element in RELFILE
format, rather than inline or in a TXLIB data set.

Notes:
1. The RELFILE value must be a decimal number from 1 to 9999.
2. RELFILE is mutually exclusive with FROMDS and TXLIB.

RMID

specifies the last SYSMOD that replaced this element. This operand can be
used only in a service-updated function, and the specified PTF must be
integrated into the function.

SHSCRIPT

specifies a UNIX shell script, scriptname, to be invoked when the element is
installed in (or deleted from) a directory of a UNIX file system. scriptname can
contain any uppercase alphabetic, numeric, or national ($, #, @) character and
can be 1 to 8 characters long.
A shell script is commonly used to complete the installation of an element. For example, if the JAR element requires an external link, you can provide a shell script that performs the necessary steps to create the external link for the JAR element.

`scriptname` must be the first value to follow the SHSCRIPT operand.

You define the shell script to SMP/E through a `++SHELLSCR` statement, which can be within the same SYSMOD as the JAR element, or within a SYSMOD that was processed previously.

You cannot define more than one shell script for an element.

You can follow `scriptname` with either of two optional values, PRE and POST, to specify the point in SMP/E processing when the shell script is to be invoked. The following examples show how you can use the PRE and POST values:

- To run the shell script before the element is copied to a UNIX file system directory, specify:
  ```
  SHSCRIPT(scriptname, PRE)
  ```

- To run the shell script after the element is copied to a UNIX file system directory, specify:
  ```
  SHSCRIPT(scriptname, POST)
  ```

  or specify no value after `scriptname` to use POST by default.

- To run the shell script both before and after the element is copied to a UNIX file system directory, specify:
  ```
  SHSCRIPT(scriptname, PRE, POST)
  ```

If you do not specify a PRE or POST value for the shell script, SMP/E invokes the shell script after the element is installed in the directory.

You cannot specify the SHSCRIPT operand with the DELETE operand.

**SYMLINK**

specifies a list of one or more symbolic links, which are file names that can be used as alternate names for referring to this element in a UNIX file system. Each linkname listed here is associated with a pathname listed in the SYMPATH operand. For more information about how the linknames and pathnames are associated, see the description of the SYMPATH operand and "Example 3: Packaging a SYSMOD with a symbolic link" on page 36.

The SYMLINK value specified should be a relative path value (that is, it does not start with a slash ["/"]). When the symbolic link is created, it is created relative to the pathname of the element's SYSLIB ddname.

**SYMLINK** must be specified if the SYMPATH operand is specified, otherwise it must be omitted.

A symbolic linkname can be from one to 1023 characters. Any characters in the range X’40’ through X’FE’ may be specified.

The value may be enclosed in single apostrophes. It **must** be enclosed in single apostrophes if:

- it is continued to the next line in the MCS, or
- it contains a character that is not uppercase alphabetic, numeric, national ($, #, @), slash (/), plus (+), hyphen, period, or ampersand (&).

If an apostrophe is a part of the symbolic linkname and is not a delimiter, then it must be doubled. These two apostrophes count as two characters against
1023 character limit for a symbolic linkname. The single apostrophes used to enclose a symbolic linkname do not count against the 1023 character limit.

SYMPATH
specifies a list of one or more pathnames that are associated with symbolic links identified by the SYMLINK operand. The first pathname in the SYMPATH operand is associated with the first symbolic link in the SYMLINK operand, the second pathname with the second symbolic link, and so on. If there are more symbolic links listed than there are pathnames, then the last listed pathname is used for the remaining symbolic links. If more pathnames are specified than symbolic linknames, then the excess pathnames (at the end of the list) are ignored.

The SYMPATH value specified should be a relative path value. When the symbolic link is accessed, the system assumes the destination of that link (the SYMPATH value) is relative to that symbolic link (the SYMLINK value). For more information about how the pathnames and linknames are associated, see “Example 3: Packaging a SYSMOD with a symbolic link” on page 36.

SYMPATH must be specified if the SYMLINK operand is specified, otherwise it must be omitted.

A symbolic pathname can be one to 1023 characters. Any characters in the range X'40' through X'FE' may be specified.

The value may be enclosed in single apostrophes. It must be enclosed in single apostrophes, if:
• it is continued to the next line in the MCS, or
• it contains a character that is not uppercase alphabetic, numeric, national ($, #, @), slash (/), plus (+), hyphen, period, or ampersand (&).

If an apostrophe is a part of the symbolic pathname and is not a delimiter, then it must be doubled. These two apostrophes count as two characters against the 1023 character limit for a symbolic pathname. The single apostrophes used to enclose a symbolic linkname do not count against the 1023 character limit.

SYSLIB
specifies the ddname of the “target library” within the UNIX file system for the element.

During APPLY processing, the HFS copy utility installs the JAR element into a UNIX file system. During RESTORE processing, the HFS copy utility copies the JAR element from the distribution library member into a UNIX file system.

Note: SYSLIB must be specified when the JAR element is first installed.

TXLIB
is the ddname of the partitioned data set containing the JAR element. This operand is required if the JAR element is provided in a data set that the users have access to, rather than inline or in RELFILE format.

Notes:
1. SMPTLIB cannot be used as a value on the TXLIB operand.
2. TXLIB is mutually exclusive with FROMDS and RELFILE.

UMID
specifies the SYSMODs that have updated this JAR file since it was last replaced. This operand can be used only in a service-updated function, and the specified SYSMODs must be integrated into the function.
++JAR MCS

**VERSION**

specifies one or more function SYSMODs that currently contain the element. The function containing the element MCS takes over ownership of the element from the specified functions.

When **VERSION** is specified on an element statement, it overrides any VERSION operand values that might be specified on the ++VER MCS.

**Usage Notes**

- If the JAR element is packaged inline, it must immediately follow the ++JAR MCS and must not contain any records starting with ++. Neither FROMDS, nor RELFILE, nor TXLIB can be specified on the ++JAR MCS when the element is packaged inline.
- To be packaged inline, a JAR file element must contain fixed-block-80 records. To achieve this, the original element can be extracted from a UNIX file system into a data set (using the TSO command OGET for example), and then GIMDTS can be used to transform the element into fixed-block-80 records.

**Examples**

Here is an example of the ++JAR MCS.

++JAR(ABCTTT) DISTLIB(AABCBIN) SYSLIB(SABCBIN) RELFILE(2)
PARM(PATHMODE(0,6,4,4))
LINK('..//TicTacToe.jar')
SYMLINK('.../usr/lib/TicTacToe.jar')
SYMPATH('.../usr/lpp/abc/bin/TicTacToe.jar').
++JARUPD MCS

The ++JARUPD MCS describes an update to a Java ARchive (JAR) file in a UNIX file system. The Java jar command is used to construct and update such files in a UNIX file system. A JARUPD element is itself a Java Archive file, but it contains only the component files to be added to or replaced in an existing JAR file, as opposed to a complete replacement for an existing JAR file.

Notes:

1. To update JAR files, SMP/E must use the update (u) option of the jar command, which is provided in version 1.2 of the Java Development Kit (JDK). Therefore, in order to perform APPLY or ACCEPT command operations for a SYSMOD that contains JARUPD elements, SMP/E requires the IBM Developer Kit for OS/390, Java 2 Technology Edition, or its successor. Java 2 Technology Edition is a no-charge product available for users of OS/390 Version 2 Release 8 and above, or z/OS Version 1 Release 1 and above.

Java must be available in the execution environment for SMP/E. The SMPJHOME DD statement or DDDEF entry should be used to specify the directory where Java resides. For example, if Java 1.4, which has been installed in the /usr/lpp/java/J1.4 directory, should be used, then the following DD statement should be specified:

//SMPJHOME DD PATH='/usr/lpp/java/J1.4/',PATHDISP=KEEP

For additional information about how to set the SMPJHOME value, see Chapter 4, “SMP/E data sets and files,” on page 141.

2. There is no way to delete component files from a JAR file. ++JARUPD can be used only to add or replace component files in a JAR file. If component files must be deleted, then the entire JAR file must be replaced with another JAR file from which those component files have been removed.

Syntax

++JARUPD MCS

++JARUPD(name) JARPARM(character_string) LINK(linkname)

PARM(character_string) RELFILE(number) SYMLINK

SYMLINK SYMLINK.symlinkname SYMPATH(sympathname)
Operands

JARPARM
specifies a character string that is to be passed to the jar command as a command option when updating the JAR file. The maximum length of this character string is 300 bytes of data.

Only those jar command options that require only the option indicator to be specified are supported, such as 0 and M. Options requiring additional input over and above the option indicator are not supported. Examples of unsupported options are the m and -C options.

Notes:
1. The character string can include any non-blank characters (characters x'41' through x'FE').
2. The JARPARM value is saved and passed to the jar command as follows:
   • If JARPARM is specified on the ++JARUPD, any value previously saved in the JAR element entry is overlaid and the new value is passed to the jar command.
   • If JARPARM is not specified on the ++JARUPD and a saved value exists in the JAR element entry, then the saved value is passed to the jar command.
   • If JARPARM is not specified on the ++JARUPD and there is no saved value in the JAR element entry, then no value is saved in the element entry and no value is passed to the jar command.

LINK
specifies the alternative names by which this JAR element can be known in a UNIX file system. The full name is produced by concatenating the specified linkname with a UNIX file system directory identified by the SYSLIB subentry. Each linkname is passed to the HFS copy utility as an execution parameter.

Notes:
1. The linkname can be from 1 to 1023 characters.
2. A linkname can be enclosed in single apostrophes ('). A linkname must be enclosed in single apostrophes if any of the following is true:
   • The linkname contains lowercase alphabetic characters.
   • The linkname contains a character that is not uppercase alphabetic, numeric, national ($, #, @), slash (/), plus (+), hyphen, period, or ampersand (&).
   • The linkname spans more than one line in the control statement.
   The single apostrophes used to enclose a linkname (the delimiters) do not count as part of the 1023-character limit.
3. Any apostrophes specified as part of a linkname (not the delimiters) must be doubled.
   Double apostrophes count as two characters in the 1023-character limit.
4. The linkname can include characters X'40' through X'FE'.
5. LINK values are saved and passed to the HFS copy utility as follows:
   • If LINK is specified on the ++JARUPD, any values previously saved in the element entry are overlaid.
   • If LINK is not specified on the ++JARUPD and saved values exist in the JAR element entry, the saved values are passed to the HFS copy utility as execution parameters.
++JARUPD MCS

If LINK is not specified on the ++JARUPD and there are no saved values in the JAR element entry, no linknames are passed to the HFS copy utility.

**name**

specifies the name of the JAR element member. The name can contain any uppercase alphabetic, numeric, or national ($, #, @) character and can be 1 to 8 characters long.

**PARM**

specifies a character string that is to be passed to the hierarchical file system copy utility as an execution-time parameter. (The values that can be specified on the PARM operand, such as PATHMODE, are those accepted by the BPXCOPY utility. See [z/OS UNIX System Services Command Reference](https://www.ibm.com/support/knowledgecenter/SSEPGG_2.2.0/com.ibm.zos.v2r11.jdk.javadoc.custom.doc/topics/c00006189.html) for a description of BPXCOPY and the values it accepts.) The maximum length of this character string is 300 bytes of nonblank data. If any blanks are specified in the PARM value, they are deleted by SMP/E during processing and do not count toward the 300-byte maximum.

**Notes:**

1. PARM is an optional operand.
2. The character string can be entered free-form, without regard to blanks (which are compressed out of the string), and can span multiple 80-byte records.
3. If parentheses are specified in the PARM value, there must always be a pair (left and right); otherwise, the results are unpredictable.
4. PARM values are saved and passed to the HFS copy utility as follows:
   - If PARM is specified on the ++JARUPD, any values previously saved in the JAR element entry are overlaid.
   - If PARM is not specified on the ++JARUPD and saved values exist in the JAR element entry, the saved values are passed to the HFS copy utility as execution parameters.
     - If PARM is not specified on the ++JARUPD and there are no saved values in the JAR element entry, no parameters are saved from the ++JARUPD or passed from the JAR element entry to the HFS copy utility.
   - If the UTILITY entry for the HFS copy utility specifies a PARM value, those parameters are passed to the utility in addition to any parameters saved in the JAR element entry.

**RELFILE**

identifies which relative file associated with the SYSMOD contains this element. This operand is required if you provide the element in RELFILE format, rather than inline or in a TXLIB data set.

**Notes:**

1. The RELFILE value must be a decimal number from 1 to 9999.
2. RELFILE is mutually exclusive with TXLIB.

**SYMLINK**

specifies a list of one or more symbolic links, which are file names that can be used as alternate names for referring to this element in a UNIX file system. Each linkname listed here is associated with a pathname listed in the SYMPATH operand. For more information about how the linknames and pathnames are associated, see the description of the SYMPATH operand and "Example 3: Packaging a SYSMOD with a symbolic link" on page 36.
++JARUPD MCS

The SYMLINK value specified should be a relative path value (that is, it does not start with a slash ["/"]. When the symbolic link is created, it is created relative to the pathname of the element's SYSLIB ddname.

**SYMLINK** must be specified if the **SYMPATH** operand is specified, otherwise it must be omitted.

A symbolic linkname can be from one to 1023 characters. Any characters in the range X'40' through X'FE' may be specified.

The value may be enclosed in single apostrophes. It **must** be enclosed in single apostrophes if:
- it is continued to the next line in the MCS, or
- it contains a character that is not uppercase alphabetic, numeric, national ($, #, @), slash (/), plus (+), hyphen, period, or ampersand (&).

If an apostrophe is a part of the symbolic linkname and is not a delimiter, then it must be doubled. These two apostrophes count as two characters against the 1023 character limit for a symbolic linkname. The single apostrophes used to enclose a symbolic linkname do not count against the 1023 character limit.

**SYMPATH** specifies a list of one or more pathnames that are associated with symbolic links identified by the **SYMLINK** operand. The first pathname in the **SYMPATH** operand is associated with the first symbolic link in the **SYMLINK** operand, the second pathname with the second symbolic link, and so on. If there are more symbolic links listed than there are pathnames, then the last listed pathname is used for the remaining symbolic links. If more pathnames are specified than symbolic linknames, then the excess pathnames (at the end of the list) are ignored.

The **SYMPATH** value specified should be a relative path value. When the symbolic link is accessed, the system assumes the destination of that link (the **SYMPATH** value) is relative to that symbolic link (the **SYMLINK** value). For more information about how the pathnames and linknames are associated, see "Example 3: Packaging a SYSMOD with a symbolic link" on page 36.

**SYMPATH** must be specified if the **SYMLINK** operand is specified, otherwise it must be omitted.

A symbolic pathname can be one to 1023 characters. Any characters in the range X'40' through X'FE' may be specified.

The value may be enclosed in single apostrophes. It **must** be enclosed in single apostrophes, if:
- it is continued to the next line in the MCS, or
- it contains a character that is not uppercase alphabetic, numeric, national ($, #, @), slash (/), plus (+), hyphen, period, or ampersand (&).

If an apostrophe is a part of the symbolic pathname and is not a delimiter, then it must be doubled. These two apostrophes count as two characters against the 1023 character limit for a symbolic pathname. The single apostrophes used to enclose a symbolic linkname do not count against the 1023 character limit.

**TXLIB**

is the ddname of the partitioned data set containing the JAR element. This operand is required if the JAR element is provided in a data set that the users have access to, rather than inline or in RELFILE format.
+++JARUPD MCS

Notes:
1. SMPTLIB cannot be used as a value on the TXLIB operand.
2. TXLIB is mutually exclusive with RELFILE.

Usage Notes

- If the JAR update file is packaged inline, it must immediately follow the 
  +++JARUPD MCS and must not contain any records starting with ++. Neither 
  RELFILE nor TXLIB can be specified on the MCS when the element is packaged 
  inline.
- To be packaged inline, a JAR file element must contain fixed-block-80 records. To 
  do this, the original element can be extracted from a UNIX file system into a 
  data set (using, for example, the TSO command OGET), and then GIMDTS can 
  be used to transform the element into fixed-block-80 records.

Examples

Here is an example of the +++JARUPD MCS.

+++JARUPD(ABCTTT) JARPARM(OM)
  PARM(PATHMODE(0,6,4,4))
  LINK('../TicTacToe.jar')
  SYMLINK('../../../../../usr/lib/TicTacToe.jar')
  SYMPATH('../../usr/lpp/abc/bin/TicTacToe.jar').
The ++JCLIN MCS identifies the start of the job control language (JCL) input that must be processed as part of installing the SYSMOD. The JCLIN input is used to identify the structure of the target system libraries to SMP/E so that the modules, macros, and source code being replaced can be installed correctly. Processing of ++JCLIN input during APPLY or ACCEPT is similar to the processing done for the JCLIN command. For an additional description of JCLIN processing, see SMP/E Commands.

Note: The ++JCLIN MCS does not cause SMP/E to update the target or distribution libraries; only the entries in the target and distribution zones are updated. These libraries are updated when SMP/E processes the elements in the SYSMOD. The element statements in the SYSMOD determine which elements should be installed and cause SMP/E to invoke the utilities that install the elements.

Syntax

++JCLIN MCS

Operands

ASM

specifies the name of the assembler program or procedure that is used in the JCLIN data. This operand must be specified if the name is different from those recognized by SMP/E, which are the program names ASMBLR, ASMA90, IEUASM, IEV90, and IFOX00, and procedure name ASMS.

CALLLIBS

specifies that SMP/E is to process SYSLIB DD statements in JCLIN link-edit steps. SYSLIB DD statements are processed only if the CALLLIBS operand is specified on the JCLIN command or ++JCLIN MCS, or if //CALLLIBS=YES is encountered after a job card preceding a link-edit step. If the CALLLIBS operand or the CALLLIBS comment is not specified, SMP/E ignores any SYSLIB DD statements it encounters.

COPY

specifies the name of the copy program or procedure that is used in the JCLIN
data. This operand must be specified if the name is different from that recognized by SMP/E, which is the program name IEBCOPY.

**FROMDS**
identifies the partitioned data set that contains this element.

**Note:** The FROMDS operand and its DSN, NUMBER, VOL, and UNIT suboperands are included in the MCS generated by the BUILDMCS command. IBM does not intend the FROMDS operand to be used in manually coded MCS.

**DSN**
specifies the dsname of the FROMDS data set. The specified data set name must conform to standard data set naming conventions and cannot contain parentheses. The maximum length of the entire name is 44 characters (including the periods).

**NUMBER**
specifies a number that SMP/E is to use when assigning a name to the SMPTLIB data set associated with this FROMDS data set. (This is similar to the way the relative file number is used in RELFILE processing.)

**VOL**
specifies, for an uncataloged data set, the volume serial number of the volume containing the FROMDS data set. If specified, this volume identifier must be from 1 to 6 alphanumeric characters.

VOL may be omitted for a cataloged data set.

**UNIT**
specifies, for an uncataloged data set, the UNIT type containing the FROMDS data set. If specified, the UNIT value must be from 1 to 8 characters and must conform to standard UNIT naming conventions. SMP/E accepts any nonblank characters specified between the open and close parentheses, up to a maximum length of 8.

UNIT may be omitted for a cataloged data set.

**Note:** FROMDS is mutually exclusive with DELETE, RELFILE, and TXLIB.

**LKED**
specifies the name of the link-edit program or procedure that is used in the JCLIN data. This operand must be specified if the name is different from those recognized by SMP/E, which are the program names HEWL, HEWLH096, HEWLKED, IEWBLINK, IEWL, and LINKEDIT, and procedure name LINKS.

**OPCODE**
identifies an OPCODE member containing control statements identifying character strings to be treated as OPCODEs or macros.

**PGM**
specifies a program name (instead of a procedure name) for a utility.

**RELFILE**
specifies the relative position, within the files associated with this SYSMOD, of the file containing the JCLIN data as one of its members.

The member in the RELFILE data set containing the JCLIN input data must match the SYSMOD’s SYSMOD ID; for example, the JCLIN for ++PTF(UZ11111) is supplied in member UZ11111 of the unloaded PDS identified by the RELFILE operand.
++JCLIN MCS

Note: RELFILE is mutually exclusive with FROMDS and TXLIB.

TXLIB

specifies the ddname of a partitioned data set containing the JCLIN data as one of its members.

The member of the TXLIB data set containing the JCLIN input data must match the SYSMOD's SYSMOD ID; for example, the JCLIN for ++PTF(UZ11111) is supplied in member UZ11111 of the data set identified by the TXLIB operand.

Notes:
1. SMPTLIB cannot be used as a value on the TXLIB operand.
2. TXLIB is mutually exclusive with FROMDS and RELFILE.

UPDATE

specifies the name of the update program or procedure that is used in the JCLIN data. This operand must be specified if the name is different from that recognized by SMP/E, which is the program name IEBUPDTE.

Usage Notes

- If the JCLIN data is packaged inline, it must immediately follow the ++JCLIN MCS and must not contain any records that start with ++. Neither FROMDS, nor RELFILE, nor TXLIB can be specified on the ++JCLIN MCS.
- If the JCLIN data is packaged in a TXLIB data set, the ddname specified on the TXLIB operand is required during APPLY and ACCEPT processing.
- For information about packaging SYSMODs in RELFILE format, see "Z/OS Packaging Rules."

Examples

For these examples, assume function JXY1040 contains some inline JCLIN, one module (IFBMOD01), and one macro (IFBMAC01). The same format was used to package the JCLIN and the elements.

Note: Even though the examples are for a function, they are equally valid for PTFs, APARs, and USERMODs.

The following examples are provided to help you use the ++JCLIN MCS:

Example 1: ++JCLIN data packaged inline

Here is an example of a SYSMOD containing a ++JCLIN statement for packaging JCLIN inline:

```plaintext
++FUNCTION(JXY1040) /* Function SYSMOD. */.
++VER(Z038) /* For MVS, no requisites. */.
++JCLIN /* JCLIN is required. */.
//JOB JOB 'accounting info',MSGLEVEL=(1,1)
//STEP1 EXEC PGM=IEWL
//SYSLMOD DD DSN=SYS1.LINKLIB,DISP=SHR
//AOS12 DD DSN=SYS1.AOS12,DISP=SHR
//SYSLIN DD *
INCLUDE AOS12(IFBMOD01)
ENTRY IFBMOD01
ALIAS IFBMOD01A
NAME IFBMOD01(R)
/*
++MOD(IFBMOD01) /* This module */.
DISTLIB(AOS12) /* for this DLIB. */.
...
... object deck for IFBMOD01
```
No special DD statements, other than those for the target and DLIBs, are required to process this SYSMOD.

During APPLY, SMP/E processes the ++JCLIN data, determines that module IFBMOD01 gets linked into load module IFBMOD01 (which has an alias), stores information in the target zone, and then invokes the proper utilities to install the module and macro.

**Example 2: ++JCLIN data packaged in a RELFILE**
Here is an example of a SYSMOD containing a ++JCLIN statement for packaging JCLIN in relative files. No special ++JCLIN operands are required.

```plaintext
++FUNCTION(JXY1040) /* Function SYSMOD */
  FILES(3) /* in RELFILE format. */.
++VER(Z038) /* For MVS, no requisites. */.
++JCLIN /* JCLIN is required. */
  RELFILE(1) /* Data is in first RELFILE. */.
++MOD(IFBMOD01) /* This module */
  DISTLIB(AOS12) /* for this DLIB */
  RELFILE(2) /* is in second RELFILE. */.
++MAC(IFBMAC01) /* This macro */
  DISTLIB(AMACLIB) /* for this DLIB */
  RELFILE(3) /* is in third RELFILE. */.
```

The SMPTLIB DD statement is required for the RECEIVE command to load the RELFILE data sets from tape onto DASD, and at APPLY and ACCEPT, in order to access the data in the unloaded partitioned data sets on DASD.

**Note:** Different RELFILE data sets are required for the module and macro, because they are in different formats (modules must be in load module format, RECFM=U, while macros are RECFM=FB). Although a separate RELFILE data set was used to contain the ++JCLIN data, that data could have been put into the data set with the macros, because both have the same attributes (LRECL=80, RECFM=FB).

**Example 3: ++JCLIN data packaged in a TXLIB with a user utility program name**
Here is an example of a SYSMOD containing a ++JCLIN statement for packaging JCLIN in text libraries. The JCLIN contains assembly steps using a special link-edit utility, ALTIIEWL.

```plaintext
++FUNCTION(JXY1040) /* Function SYSMOD. */.
++VER(Z038) /* For MVS, no requisites. */.
++JCLIN /* JCLIN is required. */
  TXLIB(LIB1040) /* Data is in text library. */
  LKED(PGM=ALTIIEWL) /* Use other link-edit. */.
++MOD(IFBMOD01) /* This module */
  DISTLIB(AOS12) /* for this DLIB */
  TXLIB(LIB1040) /* is in text library. */.
++MAC(IFBMAC01) /* This macro */
  DISTLIB(AMACLIB) /* for this DLIB */
  TXLIB(LIB1040) /* is in text library. */.
```
A DD statement for LIB0104 is required during APPLY and ACCEPT in order for SMP/E to call the utilities to get the replacements installed into the operating system libraries. An example of the DD statement is:

//LIB1040 DD DSN=...

Note: In this case, the replacements for all three elements were put in one data set. That data set must contain three members: member JXY1040 must be the JCLIN data, member IFBMOD01 must contain the object deck for module IFBMOD01, and member IFBMAC01 must contain the replacement for macro IFBMAC01. Because all have the same formats (that is, RECFM=FB LRECL=80) they can be packaged in one data set.

Member JXY1040 looks as follows:

//JOB JOB 'accounting info',MSGLEVEL=(1,1)
//STEP1 EXEC PGM=ALTIEWL
//SYSLMOD DD DSN=SYS1.LINKLIB,DISP=SHR
//AOS12 DD DSN=SYS1.AOS12,DISP=SHR
//SYSLIN DD *

INCLUDE AOS12(IFBMOD01)
ENTRY IFBMOD01
ALIAS IFBMOD01A
NAME IFBMOD01(R)

/*

Note the use of ALTIEWL on the EXEC statement.
++MAC MCS

The ++MAC MCS describes a single macro replacement. It must immediately precede the macro definition statements when they are within the SYSMOD.

Syntax

The syntax to be used depends on the processing to be done for the element:

- Adding or replacing the element
- Deleting the element

Adding or replacing a macro

```
++MAC MCS
++MAC(name)
```

Deleting a macro

```
++MAC MCS
++MAC(name) DELETE
```

Operands

**ASSEM** specifies the names of modules to be assembled in addition to those modules named as GENASM subentries in the MAC entry. The source for the assemblies is the ASSEM entry, SRC entry, or DISTSRC member whose name matches the specified ASSEM value. SMP/E looks for a match—first with an ASSEM entry, then with a SRC entry, and finally with an entry in the DISTSRC data set—and uses the first match it finds. The source must either be known to
SMP/E at the time the ASSEM operand is encountered on the ++MAC statement, or be included in the same SYSMOD.

Notes:
1. APPLY and ACCEPT processing place the specified names into the SYSMOD entry created on the target zone and distribution zone.
2. If the object deck for the element specified on the ASSEM operand is also provided by the SYSMOD, the assembly may not occur. (See SMP/E Commands for more information.)

DELETE
specifies that the macro and alias names are to be removed from the target libraries, distribution libraries, and SMP/E data sets.

Notes:
1. DELETE is mutually exclusive with all other operands except DISTLIB and VERSION.
2. If the element statement is in a base function, you may want to use the DELETE operand on the ++VER MCS to delete the previous release, rather than on the element statement to delete a specific element.

DISTLIB
specifies the ddname of the distribution library for the specified macro.

Notes:
1. DISTLIB must be specified if the macro has not been previously recorded on the target zone or distribution zone. If a MAC entry already exists in the target zone or distribution zone and the value currently in that entry does not match that specified in the DISTLIB operand, the SYSMOD is not applied or accepted, unless that SYSMOD also used the ++MOVE MCS to change the DISTLIB to that new value.
2. You cannot use SYSPUNCH as the DISTLIB. It is used by SMP/E and other products to process assembled modules.

DISTMOD
specifies the ddname of the link-edit distribution library for those modules specified in the ASSEM operand. During ACCEPT processing, the object code from the assembler is link-edited to the library specified.

DISTSRC
specifies the ddname of the library containing the additional assembly or source to be assembled. The additional assembly or source must be specified in the ASSEM operand.

FROMDS
identifies the partitioned data set that contains this element.

Note: The FROMDS operand and its DSN, NUMBER, VOL, and UNIT suboperands are included in the MCS generated by the BUILDMCS command. IBM does not intend the FROMDS operand to be used in manually coded MCS.

DSN
specifies the dsname of the FROMDS data set. The specified data set name must conform to standard data set naming conventions and cannot contain parentheses. The maximum length of the entire name is 44 characters (including the periods).

NUMBER
specifies a number that SMP/E is to use when assigning a name to the
++MAC MCS

SMPTLIB data set associated with this FROMDS data set. (This is similar to the way the relative file number is used in RELFILE processing.)

VOL specifies, for an uncataloged data set, the volume serial number of the volume containing the FROMDS data set. If specified, this volume identifier must be from 1 to 6 alphanumeric characters.

VOL may be omitted for a cataloged data set.

UNIT specifies, for an uncataloged data set, the UNIT type containing the FROMDS data set. If specified, the UNIT value must be from 1 to 8 characters and must conform to standard UNIT naming conventions. SMP/E accepts any nonblank characters specified between the open and close parentheses, up to a maximum length of 8.

UNIT may be omitted for a cataloged data set.

Note: FROMDS is mutually exclusive with DELETE, RELFILE, SSI, and TXLIB.

MALIAS specifies the alias names for the macro in both the target system and the distribution libraries.

Note: MALIAS must be specified on the ++MAC MCS even if ALIAS was specified on the COPY SELECT statement. MALIAS is required for the RECEIVE command to properly handle aliases in RELFILEs. During RECEIVE processing, SMP/E copies RELFILEs to SMPTLIB data sets. All element members and their associated aliases are copied. Because JCLIN is not processed during RECEIVE, the aliases must be identified on the MCS to get RECEIVE to copy the aliases.

You can use MALIAS when two or more macros that must be defined in the same zone must have the same name for programming access. For example, you can use MALIAS if several products have a help macro whose name must match the name of the command processing module it describes. You can specify HELP on MALIAS and a unique element name as the macro name.

name specifies the name of the macro member in the distribution library and, optionally, in the target system library. The name can contain any alphanumeric characters and $, #, @, or hex C0.

PREFIX specifies the first characters (prefix) of the names of modules to be assembled in addition to those modules named as GENASM subentries in the target zone MAC entry. The prefix values must contain no more than 7 characters.

The full module names are determined by comparing the prefix with the target zone or distribution zone MOD entry names.

The source for the assembly is the ASSEM entry, SRC entry, or DISTSRC member whose name matches a MOD entry name beginning with one of the specified prefixes. SMP/E looks for a match—first with an ASSEM entry, then with a SRC entry, finally with an entry in the DISTSRC data set—and uses the first match it finds. The source must either be known to SMP/E at the time the PREFIX operand is encountered on the ++MAC statement, or be included in the same SYSMOD.
++MAC MCS

**Note:** If the object deck for an element selected by the `PREFIX` operand is also provided by the SYSMOD, the assembly may not occur. (See [SMP/E Commands](#) for more information.)

**RELFILE**
identifies which relative file associated with the SYSMOD contains this element. This operand is required if you provide the element in RELFILE format, rather than inline or in a TXLIB data set.

**Note:** RELFILE is mutually exclusive with FROMDS and TXLIB.

**RMID**
specifies the last SYSMOD that replaced this macro. This operand may be used only in a service-updated function, and the specified PTF must be integrated into the function.

**SSI**
specifies eight hexadecimal digits of system status information. This information is placed in the directory of the target system library or SMPMTS or SMPSTS during APPLY processing, and in the distribution library during ACCEPT processing, as four packed hexadecimal bytes of user data. See the IEBUPDTE program description in [z/OS DFSMShsm Utilities, SC26-7414](#).

**Note:** This operand is ignored if text is located in a library, as is the case when either the FROMDS, RELFILE, or TXLIB operand is specified.

**SYSLIB**
specifies the ddname of the target library, if the macro exists in one. APPLY and RESTORE processing update this library.

**Note:** If a MAC entry already exists in the target zone or distribution zone and the value currently in that entry does not match that specified in the SYSLIB operand, SMP/E ignores the SYSLIB value in the SYSMOD being installed, unless that SYSMOD also used the ++MOVE MCS to change the SYSLIB to that new value.

**TXLIB**
is the ddname of the partitioned data set containing the macro. This operand is required if the macro is provided in a data set that the users have access to, rather than inline or in RELFILE format.

**Notes:**
1. SMPTLIB cannot be used as a value on the TXLIB operand.
2. TXLIB is mutually exclusive with FROMDS and RELFILE.

**UMID**
specifies the SYSMODs that have updated this macro since it was last replaced. This operand can be used only in a service-updated function, and the specified PTFs must be integrated into the function.

**VERSION**
specifies one or more function SYSMODs that currently contain the element. The function containing the ++MAC MCS will take over ownership of the element from the specified functions.

When **VERSION** is specified on an element statement, it overrides any VERSION operand values that might be specified on the ++VER MCS.
++MAC MCS

Usage Notes

- If the macro is packaged inline, it must immediately follow the ++MAC MCS and must not contain any records that start with ++. Neither FROMDS, nor RELFILE, nor TXLIB can be specified on the ++MAC MCS.
- If the macro is packaged in a TXLIB data set, the ddname specified in the TXLIB operand is required during APPLY and ACCEPT processing.
- For information about elements packaged in RELFILE format, see z/OS Packaging Rules.
- If the ++MAC MCS is for an inner macro (a macro referred to by another macro instruction that resides in the macro library), the modules that require reassembly must be specified in the ASSEM operand list.
- If the macro resides in a target library (rather than the SMPMTS), the target library should be included in the SYSLIB DD concatenation for assemblies during APPLY processing. For additional information about SYSLIB requirements, see SMP/E User’s Guide and SMP/E Commands.

Examples

The following examples are provided to help you use the ++MAC MCS.

Example 1: Replacing a macro through a USERMOD

Assume you want to replace a macro, named YOURMAC, that is part of a product you own, named YOURPROD. You have updated a copy of the macro in a text library, TSO.NEWSMP.MACLIB. Here is an example of a SYSMOD containing a ++MAC statement to build a USERMOD that installs the changes:

```++++USERMOD(USR0001) /* User modification */.
++++VER(Z038) FMID(HUSR003) /* for your product */.
++++MAC(YOURMAC) /* to replace this macro. */.
TXLIB(NEWSMP) /* Macro is in this TXLIB. */.
```

In this example, you have just applied another piece of service to the YOURPROD macro, but YOURPROD is still the owner of the macro. If you attempt to install some vendor-supplied service (that is, a PTF) to that macro, SMP/E issues an error message indicating that your user modification will be regressed, and will not install that service until the BYPASS(ID) operand is used.

Another method of installing the new macro is for you to assume ownership for the macro by using the VERSION operand. Assume you already have a user function, JXY1040, installed, and you want to transfer ownership of the SMP/E macro to your function. The following SYSMOD contains a ++MAC statement to do that:

```++++USERMOD(USR0001) /* User modification */.
++++VER(Z038) FMID(JXY1040) /* for user application */.
++++MAC(GIMOPCDE) /* to replace this macro. */.
VERSION(HMP1E00) /* Version SMP/E. */.
TXLIB(NEWSMP) /* Macro is in this TXLIB. */.
```

If after the installation of this SYSMOD any subsequent IBM service modifies this macro, the replacement or update from the IBM service is not selected. It is your responsibility to provide continued modifications for the macro. Thus, this method of updating an element should be used carefully.

In both examples, because the new macro exists in a TXLIB, the following DD statement is required during APPLY and ACCEPT:

```
//NEWSMP DD DSN=TSO.NEWSMP.MACLIB,DISP=SHR
```
Example 2: Deleting a macro
Assume you have installed one of your application programs as function HUSR001. The function contains macro USRMAC01, which is no longer required. Here is an example of a SYSMOD containing a ++MAC statement to delete the macro from the target and distribution libraries:

```mlist
++USERMOD(USR0001) /* User modification */.
++VER(Z038) FMID(HUSR001) /* for user application. */.
++MAC(USRMAC01) /* Delete this macro. */.
DELETE /* */.
```

Example 3: Adding a new macro
Assume you have installed one of your application programs as function HUSR001. A new macro USRMAC02 is required, and it has these requirements:

- The macro is to be put into SYS1.USRMACS in the target zone.
- The macro is to be put into SYS1.AUSRMACS in the distribution zone.
- The macro is to be installed with an alias of TERMINAL.
- After being installed, modules USRASM01, USRASM02, USRSRC01, and USRSRC02 are to be assembled. USRASM01 and USRASM02 already exist as assembler entries in the CSI, and USRSRC01 and USRSRC02 exist as source entries in the CSI.

Here is an example of a SYSMOD containing a ++MAC statement defining all this information:

```mlist
++USERMOD(USR0002) /* User modification */.
++VER(Z038) FMID(HUSR001) /* for user application. */.
++MAC(USRMAC02) /* Add this macro */.
  DISTLIB(AUSRMACS) /* to this DLIB */.
  SYSLIB(USRMACS) /* and to this tgt lib */.
  MALIAS(TERMINAL)/* with this alias. */.
  ASSEM(USRASM01, /* Assemble these modules. */
  USRASM02, /* */
  USRSRC01, /* */
  USRSRC02) /* */.
/* Inline text follows. */.
...
... macro USRMAC02 goes here
...
Example 4: Packaging a renamed macro

Suppose that, for some reason, you need to rename macro USRMAC02, which was introduced in “Example 3: Adding a new macro” on page 70. The new name is to be USRMACXX. You do not need to change anything else about the macro. Here is an example of a SYSMOD containing the ++MAC statements needed to package this renamed macro:

```
++USERMOD(USR0003) /* User modification */.
++VER(Z038) FMID(HUSR001) /* to user application. */
  PRE(USR0002) /* Base on previous USERMOD. */.
++MAC(USRMAC02) /* Delete the original */
DELETE /* macro. */.
++MAC(USRMACXX) /* Add the renamed macro */
  DISTLIB(AUSRMACS) /* to this DLIB */
  SYSLIB(USRMACS) /* and to this tgt lib */
  MALIAS(TERMINAL) /* with this alias. */
  ASSEM(USRASM01, /* Assemble these modules. */
    USRASM02, /* */
    USRSRC01, /* */
    USRSRC02 /* */
  )/* Inline text follows. */.
...
... macro USRMACXX goes here
...
The ++MACUPD MCS describes a single macro update within a PTF, an APAR fix, or a USERMOD. It must immediately precede the macro update statements within the SYSMOD.

**Syntax**

```
++MACUPD MCS
```

- **++MACUPD(name)**
- **ASSEM(module)**
- **DISTMOD(ddname)**
- **DISTSRC(ddname)**
- **SYSLIB(ddname)**
- **MALIAS(alias)**
- **PREFIX(prefix)**
- **DISTLIB(ddname)**

**Operands**

**ASSEM**

specifies the names of modules to be assembled in addition to those named as GENASM subentries in the MAC entry. The source for the assemblies is the ASSEM entry, SRC entry, or DISTSRC member whose name matches the specified ASSEM value. SMP/E looks for a match—first with an ASSEM entry, then with a SRC entry, and finally with an entry in the DISTSRC data set—and uses the first match it finds. The source must either be known to SMP/E at the time the ASSEM operand is encountered on the ++MACUPD statement, or be included in the same SYSMOD.

**Notes:**

1. APPLY and ACCEPT processing place the specified names into the SYSMOD entry created on the target zone and distribution zone.
2. If the object deck for the element specified on the ASSEM operand is also provided by the SYSMOD, the assembly may not occur. For more information, see the section on assemblies in the APPLY command chapter in *SMP/E Commands*.

**DISTLIB**

specifies the ddname of the distribution library for the specified macro.

**Notes:**

1. DISTLIB must be specified if the macro has not been previously recorded on the target zone or distribution zone. If a MAC entry already exists in the target zone or distribution zone, and the value currently in that entry does not match that specified in the DISTLIB operand, the SYSMOD is not applied or accepted.
**++MACUPD MCS**

2. You cannot use SYSPUNCH as the DISTLIB. It is used by SMP/E and other products to process assembled modules.

**DISTMOD**

specifies the ddname of the link-edit distribution library for the modules specified in the ASSEM operand. During ACCEPT processing, the object code from the assembler is link-edited to the library specified.

**DISTSRC**

specifies the ddname of the library containing the additional assembly or source to be assembled. The additional assembly or source must be specified in the ASSEM operand.

**MALIAS**

specifies the alias names for the macro in both the target system and distribution libraries.

You can use MALIAS when two or more macros that must be defined in the same zone must have the same name for programming access. For example, you can use MALIAS if several products have a help macro whose name must match the name of the command processing module it describes. You can specify HELP on MALIAS and a unique element name as the macro name.

**name**

specifies the name of the macro member in the distribution library and, optionally, in the target system library. The name can contain any alphanumeric characters and $, #, @, or hex C0.

**PREFIX**

specifies the first characters (prefix) of the names of modules to be assembled in addition to those modules named as GENASM subentries in the target zone MAC entry. The prefix values must be 7 characters or less.

The full module names are determined by comparing the prefix with the target zone or distribution zone MOD entry names.

The source for the assembly is the ASSEM entry, SRC entry, or DISTSRC member whose name matches a MOD entry name beginning with one of the specified prefixes. SMP/E looks for a match—first with an ASSEM entry, then with a SRC entry, and finally with an entry in the DISTSRC data set—and uses the first match it finds. The source must be either known to SMP/E at the time the PREFIX operand is encountered on the ++MACUPD statement, or be included in the same SYSMOD.

**Note:** If the object deck for the element specified on the PREFIX operand is also provided by the SYSMOD, the assembly may not occur. For more information, see the section on assemblies in the APPLY command chapter in [SMP/E Commands](#).

**SYSLIB**

specifies the ddname of the target library, if the macro exists in one. APPLY and RESTORE processing update this library.

**Usage Notes**

- If a SYSMOD containing a ++MACUPD statement attempts to change the ownership (FMID) of the element (with the VERSION operand), the SYSMOD cannot be installed.
- The changes for the macro must immediately follow the ++MACUPD MCS and must not contain any records that start with ++.
## ++MACUPD MCS

- If the macro resides in a target library (rather than the SMPMTS), that target library should be included in the SYSLIB DD concatenation for assemblies during APPLY processing. For additional information about SYSLIB requirements, see [SMP/E User’s Guide](#) and [SMP/E Commands](#).
- The only IEBUPDTE control statements allowed in a SYSMOD are ./CHANGE and ./ENDUP.
- The only IEBUPDTE CHANGE operand SMP/E checks is NAME, which must specify the same element as the ++MACUPD MCS. Other CHANGE operands may produce undesired results and are used at your own risk. For example, if you code UPDATE=INPLACE, SMP/E may update the distribution library. Once the distribution libraries are changed, there is no way to remove the updates.
- SMP/E does not support a continuation of the ./CHANGE statement.
- SMP/E generates any ./ALIAS statements needed and places them in the IEBUPDTE input data following the last text statement. The ./ALIAS control statements are generated only for macro updates.
- When processing multiple updates to the same lines in a given macro, SMP/E uses the ./CHANGE statement from the last update to the lines.
- If an APAR fix or USERMOD updates a macro that causes an assembly, SMP/E sets the ASSEMBLY indicator in the MOD entry for the assembled module. This can cause a problem when additional service that does not know about the macro change is installed at the same time as the APAR or USERMOD—for example, if you are installing your own USERMOD and IBM-supplied PTFs with the same APPLY command. In such cases, because the ASSEMBLY indicator is set, the module is reassembled, but does not contain the macro changes from the APAR or USERMOD. To prevent these assemblies, you can reset the ASSEMBLY indicator using UCLIN after installing the APAR or USERMOD.

### Examples

Assume you want to update macro IFBMAC02, which resides in distribution library IFBMACS. Because of this change, module IFBSRC01 must be reassembled. Module IFBSRC01 exists as a source in distribution library SYS1.IFBSRC, and as an object module in distribution library SYS1.AOS23. The macro and the modules are part of JXY1040, a user-written function. Here is an example of a SYSMOD containing a ++MACUPD statement to make the necessary changes:

```plaintext
++PTF(USR0001) /* Preventive service */.
++VER(Z038) FMID(JXY1040) /* for user product. */.
++MACUPD(IFBMAC02) /* Update this macro */.
   DISTLIB(IFBMACS) /* in this DLIB. */.
   ASSEM(IFBSRC01) /* Assemble this source. */.
   DISTSRC(IFBSRC) /* Source is here. */.
   DISTMOD(AOS23) /* Assembled SRC goes here. */.
./CHANGE name=IFBMAC02
... IEBUPDTE control cards and data
...
```

In this example, DD statements are required when the SYSMOD is applied to define the target libraries for the macro and the load module to be updated as a result of the assembly. For example, if the modules in SYS1.AOS23 (the assembled module's distribution library) were copied to SYS1.LINKLIB and the source in SYS1.IFBSRC (the source element's distribution library) were copied to SYS1.CHGLIB, the following DD statements are needed:

```plaintext
//LINKLIB DD DSN=SYS1.LINKLIB...
//CHGLIB DD DSN=SYS1.CHGLIB...
```
++MACUPD MCS

In this example, the following DD statements are needed when the SYSMOD is being accepted to define the distribution libraries:

//IFBMACS DD DSN=SYS1.IFBMACS... (macro DLIB)
//IFBSRC DD DSN=SYS1.IFBSRC... (source DLIB for assembly)
//AOS23  DD DSN=SYS1.AOS23... (DLIB for module assembled)
The +MOD MCS describes a single module replacement. It must immediately precede the module definition statements when they are within the SYSMOD. You should use the +MOD MCS when you want to provide the object form of a module. If you want to provide the source form and have it assembled when the SYSMOD is installed, use the +SRC MCS instead.

Syntax
The syntax to be used depends on the processing to be done for the element:
- Adding or replacing the element
- Deleting the element

Adding or replacing a module

Deleting a module
++MOD MCS

Operands

**CSECT**
lists all the CSECTs contained in the module. This operand is required if the module contains more than one CSECT or if the CSECT name is different from the module name on the ++MOD MCS.

- If no CSECT name is specified, SMP/E assumes that the module contains only one CSECT, whose name matches the module name on the ++MOD MCS.
- If **CSECT** is specified, it must include all the CSECTs contained in the module, even if one of them has the same name as the module.

**Notes:**
1. A CSECT name can contain from 1 to 8 characters. The name can contain any characters except the following:
   - Comma ,
   - Left parenthesis (
   - Right parenthesis )
   - Blank
2. Comments are not allowed within a CSECT name. For example, the following is not allowed:
   
   ```
   CSECT ( /* this is a csect name */ CSECT01)
   ```
   The comment is interpreted as part of the CSECT name, instead of a comment.
3. Even if **CSECT** is not specified on the ++MOD MCS used to create a MOD entry, CSECT information is saved if **CSECT** is specified on subsequent ++MOD statements that update the MOD entry.

**DALIAS**
is the alias name of a module that has an alias in the distribution library, but not in the target library. This might be used if the module is included under its alias name during system generation.

**Note:** DALIAS is mutually exclusive with TALIAS.

**DELETE**
indicates that the module and alias names are to be removed from the target libraries, distribution libraries, and SMP/E data sets.

In order to DELETE a module, the FMID on the SYSMOD must match the FMID of the module. If the module has no FMID associated with it, then the SYSMOD will delete the module. If the FMID of the module does not match the FMID on the SYSMOD, then the SYSMOD will not delete the module, because the specified FMID does not own the module.

**Note:** DELETE is mutually exclusive with all other operands except CSECT, DISTLIB, and VERSION.

**DISTLIB**
specifies the ddname of the distribution library for the specified module.

**Notes:**
1. This operand must be specified if the module has not been previously recorded on the target zone or distribution zone. If a MOD entry already exists in the target zone or distribution zone and the value currently in that entry does not match that specified in the DISTLIB operand, the SYSMOD
is not applied or accepted, unless that SYSMOD also used the ++MOVE
MCS to change the DISTLIB to that new value.

2. You cannot use SYSPUNCH as the DISTLIB. It is used by SMP/E and other
products to process assembled modules.

FROMDS
identifies the partitioned data set that contains this element.

**Note:** The FROMDS operand and its DSN, NUMBER, VOL, and UNIT
suboperands are included in the MCS generated by the BUILDMCS
command. IBM does not intend the FROMDS operand to be used in
manually coded MCS.

DSN  specifies the dsname of the FROMDS data set. The specified data set
name must conform to standard data set naming conventions and
cannot contain parentheses. The maximum length of the entire name is
44 characters (including the periods).

NUMBER
specifies a number that SMP/E is to use when assigning a name to the
SMPTLIB data set associated with this FROMDS data set. (This is
similar to the way the relative file number is used in RELFILE
processing.)

VOL  specifies, for an uncataloged data set, the volume serial number of the
volume containing the FROMDS data set. If specified, this volume
identifier must be from 1 to 6 alphanumeric characters.

UNIT specifies, for an uncataloged data set, the UNIT type containing the
FROMDS data set. If specified, the UNIT value must be from 1 to 8
characters and must conform to standard UNIT naming conventions.
IBM SMP/E for z/OS, V3R6 accepts any nonblank characters specified
between the open and close parentheses, up to a maximum length of 8.

UNIT may be omitted for a cataloged data set.

**Note:** FROMDS is mutually exclusive with DELETE, LKLIB, RELFILE,
and TXLIB.

LEPARM
specifies link-edit utility attributes for the module. Any of the following values
can be specified:
Notes:

1. The LEPARM values from the ++MOD MCS are associated with a load module entry only if the module was copied, not link-edited, into the target libraries. (The COPY indicator is set in the load module entry.) If the load module was link-edited, JCLIN must be used to change its link-edit utility attributes.

2. During APPLY processing of a ++MOD MCS with LEPARMS, the LEPARM options are saved not in the MOD entry created, but in the LMOD entry. During ACCEPT processing, the MOD entry is created with the LEPARMS present. The target zone MOD entry can contain the LEPARM options through either UCLIN or the copying of the distribution zone to a target zone.

3. All LEPARM attributes may also be specified in the format ‘attribute=value’. For example, FILL(nn) may also be specified as FILL=nn.

4. The previously listed link-edit attributes are the only attributes that can be specified on the LEPARM operand. If any other attributes are specified, a syntax error will result during RECEIVE processing.

5. The LEPARM values of DCBS, LET, LIST, XCAL, and XREF are recognized by SMP/E, but are not saved. Specifying them on the ++MOD MCS does not cause them to be passed to the link-edit utility.

6. RMODE(31) is a synonym for RMODE(ANY).

For more information about how the LEPARM operand is processed, see the chapters on ACCEPT and APPLY processing in SMP/E Commands. These attributes are described in full in “MOD entry (distribution and target zone)” on page 275.
**++MOD MCS**

**LKLlib**

is the ddname of the partitioned data set containing the link-edited format of the object module. This operand is required if the module is provided in a data set, rather than inline or on a tape.

*Note:* LKLlib is mutually exclusive with FROMDS, RELFILE, and TXLIB.

**LMOD**

lists existing load modules that are to contain the module. If any of the names specified are not already LMOD subentries in the target zone MOD entry, they are added during APPLY processing.

*Notes:*
1. LMOD can be used only to add a module to an existing load module.
2. LMOD cannot be used to create a new load module. Nor can it be used if any link-edit control statements must be added or changed to add the module to an existing load module. However, you can use JCLIN data to create a new load module and to add or change link-edit control statements.
3. If an LMOD entry does not exist for one of the load modules specified, sufficient information is not available to create one. Thus, when the MOD is to be link-edited during APPLY processing, an error message is issued, and no link-edit is performed for that load module.

**name**

specifies the name of the module in the distribution library and, optionally, in the target library. The name can contain any alphanumeric characters and $, #, @, or hex C0.

**RELFILE**

identifies which relative file associated with the SYSMOD contains this module. This operand is required if you provide the element in RELFILE format, rather than inline or in a LKLlib or TXLIB data set.

*Notes:*
1. RELFILE is mutually exclusive with FROMDS, LKLlib, and TXLIB.
2. If an object module is provided in RELFILE format, it must be in link-edited format.

**RMID**

specifies the last SYSMOD that replaced this module. This operand can be used only in a service-updated function, and the specified PTF must be integrated into the function.

**TALIAS**

specifies one or more alias names for the module. The aliases exist in the distribution library and the target library.

You can use TALIAS for a module that was copied from a distribution library into a target library (defined by JCLIN data as a copied module), but not for one that is link-edited (defined by JCLIN data as a link-edited module).

*Note:* TALIAS must be specified on the ++MOD MCS even if ALIAS was specified on the COPY SELECT statement. TALIAS is required for the RECEIVE command to properly handle aliases in RELFILEs. During RECEIVE processing, SMP/E copies RELFILEs to SMPTLIB data sets. All element members and their associated aliases are copied. Because JCLIN is not processed during RECEIVE, the aliases must be identified on the MCS to get RECEIVE to copy the aliases.
Likewise, to specify an alias for a copied load module, you must use the TALIAS operand on the ++MOD statement for that load module. (To specify an alias for a link-edited load module, do not use TALIAS. You must identify that alias using an ALIAS link-edit control statement in the JCLIN that defined the load module. For more information, see [SMP/E Commands](#).)

**Note:** TALIAS is mutually exclusive with DALIAS.

**TXLIB**

is the ddname of the partitioned data set containing an object module that has not been link-edited. This operand is required if the module is provided in a TXLIB data set rather than inline, in a FROMDS data set, in a LKLIB data set, or in RELFILE format.

**Notes:**
1. SMPTLIB cannot be used as a value on the TXLIB operand.
2. TXLIB is mutually exclusive with FROMDS, LKLIB, and RELFILE.

**UMID**

specifies the SYSMODs that have updated this module since it was last replaced. This operand can be used only in a service-updated function, and the specified PTFs must be integrated into the function.

**VERSION**

specifies one or more function SYSMODs that currently contain the element. The function containing the ++MOD MCS takes over ownership of the element from the specified functions.

When VERSION is specified on an element statement, it overrides any VERSION operand values specified on the ++VER MCS.

**Usage Notes**

- If the module is packaged inline, it must immediately follow the ++MOD MCS and must not contain any records that start with ++. Neither FROMDS, LKLIB, RELFILE, nor TXLIB may be specified.
- If the module is packaged in a TXLIB data set, the ddname specified on the TXLIB operand is required during APPLY and ACCEPT processing.
- If the module is in an LKLIB data set, the ddname specified in the LKLIB operand is required during APPLY and ACCEPT processing. Module replacements in an LKLIB data set must be in load module format (that is, processed by the link-edit utility).
- For information about packaging SYSMODs in RELFILE, TXLIB, or inline format, see [z/OS Packaging Rules](#).
- There are several ways to associate a module with a load module:
  - The DISTLIB library can be totally copied into the target library.
  - JCLIN can identify the module as part of one or more load modules.
  - The LMOD operand on the ++MOD MCS can indicate the associated load module.

If SMP/E cannot identify the load module associated with a given module, it does not update the target libraries during APPLY processing. Instead, it issues warning message GIM43401W.

**Examples**

The following examples are provided to help you use the ++MOD MCS:
Example 1: Adding a new module to an existing load module

Module IFBMOD01 is a new module that is to be placed in the distribution library SYSLIB.AOSFB and is to be link-edited with the existing load module IEEFRQ in the target system library SYSLIB.LINKLIB. Module IFBMOD01 contains two CSECTs, IFBCST01 and IFBCST02. Here is an example of a SYSMOD containing a ++MOD statement to do this:

```++USERMOD(USR0001) /* User modification */.  
++VER(Z038) FMID(JXY1040) /* to user application. */.  
++MOD(IFBMOD01) /* Add this module */.  
   DISTLIB(AOSFB) /* to this DLIB at ACCEPT, */.  
   LMOD(IEEFRQ) /* to this LMOD at apply. */.  
   CSECT(IFBCST01 /* Module has two CSECTs. */.  
   IFBCST02) /* */.```

The following DD statement is needed at APPLY time to define the operating system load module library:

```//LINKLIB DD DSN=SYS1.LINKLIB,DISP=OLD```

When the SYSMOD is accepted, the following DD statement is needed to define the distribution library for this module:

```//AOSFB DD DSN=SYS1.AOSFB,DISP=OLD```

Example 2: Specifying link-edit utility attributes with LEPARM

For this example, assume you have installed a product, FXY1040, packaged in RELFILE format. The package contained a module, IFBMOD01, that was identified by inline JCLIN as being installed as follows:

1. It was copied from the distribution library, AOS12, to the target system library, LPALIB.
2. It was linked with several other modules to form a load module, IFBLMDXX, in LINKLIB.
3. It was linked, by itself, to form a load module, named IFBLMDX1, in LINKLIB.

The JCLIN in the initial function was as follows:

```//JOB JOB 'accounting info',MSGLEVEL=(1,1)  
//COPY1 EXEC PGM=IEBCOPY  
//AOS12 DD DSN=SYS1.AOS12,DISP=SHR  
//LPALIB DD DSN=SYS1.LPALIB,DISP=SHR  
//SYSIN DD +  
COPY INDD=AOS12,OUTDD=LPALIB  
SELECT M=(IFBMOD01)  
/*  
//LINK1 EXEC PGM=IEWL,PARM='REUS'  
//AOS12 DD DSN=SYS1.AOS12,DISP=SHR  
//SYSLMOD DD DSN=SYS1.LINKLIB,DISP=SHR  
//SYSLIN DD +  
   INCLUDE AOS12(IFBMOD01)  
   INCLUDE AOS12(IFBMOD0A,IFBMOD0B,IFBMOD0C)  
   ENTRY IFBMOD01  
   NAME IFBLMDXX(R)  
/*  
//LINK2 EXEC PGM=IEWL  
//AOS12 DD DSN=SYS1.AOS12,DISP=SHR  
//SYSLMOD DD DSN=SYS1.LINKLIB,DISP=SHR  
//SYSLIN DD +  
   INCLUDE AOS12(IFBMOD01)  
   NAME IFBLMDX1(R)  
/*```
The target zone now contains the following entries:

1. MOD entry for IFBMOD01, having LMOD subentries of IFBMOD01 (from the copy step), IFBLMDXX (from the first link step), and IFBLMDX1 (from the second link step).

2. LMOD entry for IFBMOD01, (created from the SELECT statement of the copy step), indicating that the load module was copied during installation. The LMOD entry does not have any link-edit utility parameters yet, because SMP/E copied IFBMOD01 from the RELFILE data sets, and, thus, had no need to obtain the link parameters.

3. LMOD entry for IFBLMDXX (created from the first link step), with link-edit attributes of REUS.

4. LMOD entry for IFBLMDX1 (created from the second link step), with link-edit attributes of STD (because no special parameters were specified).

Now assume a PTF to replace module IFBMOD01 is required. Module IFBMOD01 has link-edit utility attributes of REUS and RENT. Here is an example of a SYSMOD containing a ++MOD statement to do this:

```plaintext
++PTF(UZ12345) /* PTF */.
++VER(Z038) FMID(FXY1040) /* for this function. */.
++MOD(IFBMOD01) /* Replace IFBMOD01 */.
  DISTLIB(AOS12) /* in this DLIB. */.
  LEPARM(RENT /* Reentrant */.
       REUS) /* and reusable. */.
```

... object deck for IFBMOD01 ...

When the PTF is applied, SMP/E processes the LEPARM as follows:

1. Because LMOD IFBMOD01 was copied from the DLIB module, it is updated using the LEPARM values from the ++MOD MCS.

2. LMOD IFBLMDXX link-edit attributes remain as they are. This is because the link-edit utility attributes of each module within the load module have no bearing on the link-edit utility attributes of the load module; and, in this case, they are not the same. One of the other modules in IFBLMDXX must have a more restrictive set of link-edit attributes, thus forcing the load module to have that restrictive set of attributes.

3. LMOD IFBLMDX1 link-edit attributes remain as they are, for the same reason that load module IFBLMDXX attributes did not change. This is true even though load module IFBLMDX1 is composed of only the one DLIB module. As long as the load module was identified by a link-edit step, SMP/E assumes that the load module may contain multiple DLIB modules.

**Note:** The only way to change the link-edit attributes of a load module that was link-edited during initial installation is to provide JCLIN input to identify the new link-edit attributes.

**Example 3: Packaging a renamed module**

Suppose that, for some reason, you need to rename module IFBMOD01, which was introduced in “Example 1: Adding a new module to an existing load module” on page 82. The new name is to be IFBMODXX. You do not need to change anything else about the module. Here is an example of a SYSMOD containing the ++MOD statements needed to package this renamed module:

```plaintext
++USERMOD(USR0002) /* User modification */.
++VER(Z038) FMID(JXY1040) /* to user application. */.
  PRE(USR0001) /* Base on previous USERMOD. */.
++MOD(IFBMOD01) /* Delete the original */.
```

Chapter 2. SMP/E modification control statements 83
++MOD MCS

DELETE /* module. */.
++MOD(IFBMODXX) /* Add the renamed module */.
DISTLIB(AOSFB) /* to this DLIB at ACCEPT. */.
LMOD(IEEFRQ) /* to this LMOD at apply. */.
CSECT(IFBCST01 /* Module has two CSECTs. */.
  IFBCST02) /* */.

... object deck for IFBMODXX
...

Note: When packaging a renamed module, you must ensure that this module is included in the appropriate load modules, either through JCLIN or the LMOD operand on the ++MOD statement, as in this example.

The following DD statement is needed at APPLY time to define the operating system load module library:

//LINKLIB DD DSN=SYS1.LINKLIB,DISP=OLD

When the SYSMOD is accepted, the following DD statement is needed to define the distribution library for this module:

//AOSFB DD DSN=SYS1.AOSFB,DISP=OLD

Example 4: Deleting a module

If you need to delete a module, here is an example of a SYSMOD containing the ++MOD statement:

++USERMOD(USR0004) /* User modification */.
++VER(Z038) FMTID(XXY1040) /* to user application. */.
++MOD(IFBMOD0Z) /* Delete the module */.
  DELETE /* from DLIB AOSFB - entry */.
  DISTLIB(AOSFB) /* for DLIB must be AOSFB. */.

Note: When packaging a SYSMOD to delete a module, no object deck is required. The previous example is a complete SYSMOD, and does not require any other SMP/E elements in the SYSMOD.

For more information about the processing done when a module is deleted, see the section on deleting elements in **SMP/E Commands**.
++MOVE MCS

The ++MOVE MCS moves a macro, a module, a source, or a load module (and any known aliases) from one library to another. The associated target or distribution zone is automatically updated to show that the entry has been deleted from the old library and added to the new one.

**Note:** You cannot use ++MOVE to move a load module from one path to another path in a UNIX file system if symbolic links have been defined for that load module.

**Syntax**

The syntax to be used depends on the type of library involved:

- Moving to a different distribution library (DISTLIB)
- Moving to a different target library (SYSLIB)

### Moving to another DISTLIB

```
++MOVE MCS

++MOVE (name) DISTLIB(ddname) TODISTLIB(ddname)
```

### Moving to another SYSLIB

```
++MOVE MCS

++MOVE (name) SYSLIB(ddname) TOSYSLIB(ddname)
```

**Operands**

- **DISTLIB**
  
  specifies the ddname of the distribution library in which the member resides.

- **FMID**
  
  specifies the FMID that owns the element. This is used when the current owner of the element is different from the FMID specified on the ++VER MCS. Up to 10 SYSMOD IDs can be specified.

  **Note:** FMID is mutually exclusive with LMOD.
++MOVE MCS

MAC, MOD, SRC, or LMOD
specifies the type of member to be moved.
  • If DISTLIB and TODISTLIB are specified, MAC, MOD, or SRC is valid.
  • If SYSLIB and TOSYSLIB are specified, MAC, SRC, or LMOD is valid.

**Note:** LMOD is mutually exclusive with FMID.

name
specifies the name of the element or load module to be moved.

SYSLIB
specifies the ddname of the target library in which the member resides.

TODISTLIB
specifies the ddname of the distribution library to which the member is to be moved. This must be specified if DISTLIB is specified.

TOSYSLIB
specifies the ddname of the target library to which the member is to be moved. This must be specified if SYSLIB is specified.

**Usage Notes**

- The member and library operands are required. You must specify the member name, the set of libraries affected by the move, and the member type.
- ++MOVE statements must follow any ++VER and ++IF statements and must precede any element MCSs.
- Regardless of the order in which ++MOVE, ++RENAME, and ++DELETE statements are coded in a SYSMOD, they are processed in the following order:
  - APPLY and ACCEPT
    1. ++MOVE
    2. ++RENAME
    3. ++DELETE
  - RESTORE
    1. ++RENAME
    2. ++MOVE

Afterwards, ++JCLIN statements are processed, and then element statements.
- You must use the FMID operand to identify all possible owners of an element to be moved.
  Using the FMID operand does not imply that the owner of the element is being changed. You can change ownership only by specifying the VERSION operand on the ++VER MCS or on the element statement. Even if you are changing the owner of the element, you must specify the FMID operand, because the element is moved before its ownership is changed. For more information about the VERSION operand, see “++VER MCS” on page 118.
- A ++MOVE MCS can move a member to a given library from one library at a time. If a member exists in more than one library, you must use additional ++MOVE or ++DELETE statements to process the additional copies. For more information about ++DELETE statements, see “++DELETE MCS” on page 17.
- If a ++MOVE MCS changes the DLIB for an element that is also being replaced, the element MCS must specify the new DLIB.

**Examples**

Assume IBM ships you a PTF that moves module MODAA from its current distribution library to a new library, and also moves load module LMODA from its
current target library to a new one. Here is an example of a SYSMOD containing ++MOVE statements that makes these changes:

```plaintext
++PTF(UR01234) /* Identify the PTF number */.
++VER(Z038) FMID(HXY1300) /* for MVS function HXY1300. */.
++IF (ESY1300) THEN /* If ESY1300 is installed */
  REQ(UR12399) /* UR12399 is required. */.
++MOVE (MODAA) /* Move module MODAA */
  DISTLIB(AOS11) /* from DLIB AOS11 */
  TODISTLIB(AOSXX) MOD /* to DLIB AOSXX. */.
++MOVE (LMODA) /* Move load module LMODA */
  SYSLIB(LINKLIB) /* from LINKLIB */
  TOSYSLIB(LPALIB) LMOD /* to LPALIB. */.
++MOD(MODAA) DISTLIB(AOSXX) /* Element MCS statements. */.
```

++MOVE MCS

Chapter 2. SMP/E modification control statements 87
The ++NULL MCS is valid only in the SMPHOLD data set. It provides no SMP/E function other than allowing all service tapes to be built with the same format, even though a function may have no exception SYSMOD data for a particular month.

Syntax
++PRODUCT MCS

The ++PRODUCT MCS is used to describe information about a product. It introduces descriptive information about a new or replacement product into the global zone.

Syntax

++PRODUCT MCS

++PRODUCT(prodid,vv.rr.mm)—DESCRIPTION(description)—SREL(srel)—PRODSUP(prodid,vv.rr.mm)—URL(product_url)—VENDOR(vendor_name)—REWORK(level)

Operands

The PRODUCT is identified by the combination of the prodid and the vv.rr.mm values.

prodid

is a 1- to 8-character product identifier. It can contain uppercase alphabetic, numeric, and national ($, #, @) characters. It may also contain one or more dashes (-). For IBM products, the prodid is assumed to be the IBM program product number (5647-A01, for example).

vv.rr.mm

specifies the version, release, and modification level of this PRODUCT. It is 5 to 8 characters long in the form vv.rr.mm. The version (vv), release (rr), and modification (rrm) values must be one or two numeric characters separated by a period (‘.’). SMP/E will insert leading zeros to each section of the vv.rr.mm value that is one character long. That is, 2.5.0 will become 02.05.00 when stored.

DESCRIPTION

a text description of the product.

- DESCRIPTION can also be specified as DESC.
- The DESCRIPTION value can be in single-byte characters (such as English alphanumeric) or double-byte characters (such as Kanji).
- The DESCRIPTION value can contain up to 64 bytes of data, including blanks. (For double-byte data, the 64-byte maximum includes all shift-in and shift-out characters, as well as the double-byte characters.) Extra blanks as well as leading and trailing blanks are deleted.
- The DESCRIPTION value can span multiple 80-byte records. Data must continue up to and including column 72 and begin in column 1 of the next
++PRODUCT MCS

line. All data past column 72 is ignored. The break does not translate to a blank unless a blank is explicitly coded in column 72 of the first line or in column 1 of the second line.

- The DESCRIPTION value cannot be only blanks.
- If parentheses are included in the text, they must be in matched pairs.

**SREL**

specifies the system or subsystem releases on which the PRODUCT can be installed. Each SREL value must be four alphanumeric characters, usually one alphabetic character followed by three numeric characters. These are the systems and subsystems defined by IBM, with their SRELs:

<table>
<thead>
<tr>
<th>System</th>
<th>SREL</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB2</td>
<td>P115</td>
</tr>
<tr>
<td>CICS®</td>
<td>C150</td>
</tr>
<tr>
<td>IMS™</td>
<td>P115</td>
</tr>
<tr>
<td>MVS</td>
<td>Z038</td>
</tr>
<tr>
<td>NCP</td>
<td>P004</td>
</tr>
</tbody>
</table>

The list of SRELs is used during RECEIVE processing to determine whether a PRODUCT should be received.

**PRODSUP**

indicates which PRODUCTS are superseded (replaced) by this PRODUCT. It is a list of prodid,vv.rr.mm values for the PRODUCTS being superseded. The combination of prodid and vv.rr.mm determines the uniqueness of an entry in the PRODSUP operand list.

The PRODSUP operand must not specify a prodid,vv.rr.mm combination matching the prodid,vv.rr.mm combination of the ++PRODUCT statement on which it is specified.

**URL**

specifies a uniform resource locator (URL) that can be used to obtain additional information about this product.

- The URL can be in single-byte characters (such as English alphanumeric) or double-byte characters (such as Kanji).
- The URL can contain up to 256 bytes of data, excluding blanks. (For double-byte data, the 256-byte maximum includes all shift-in and shift-out characters, as well as the double-byte characters.) All blanks including leading and trailing blanks are deleted.
- The URL can span multiple 80-byte records. Data must continue up to and including column 72 and begin in column 1 of the next line. All data past column 72 is ignored, including blanks. The break does not translate to a blank.
- The URL cannot be only blanks.
- If parentheses are included in the text, they must be in matched pairs.

**VENDOR**

specifies the name of the vendor supplying the product.

- The VENDOR can be in single-byte characters (such as English alphanumeric) or double-byte characters (such as Kanji).
- The VENDOR can contain up to 64 bytes of data, including blanks. (For double-byte data, the 64-byte maximum includes all shift-in and shift-out characters, as well as the double-byte characters.) Extra blanks as well as leading and trailing blanks are deleted.
++PRODUCT MCS

- The VENDOR can span multiple 80-byte records. Data must continue up to and including column 72 and begin in column 1 of the next line. All data past column 72 is ignored. The break does not translate to a blank unless a blank is explicitly coded in column 72 of the first line or in column 1 of the second line.
- The VENDOR cannot be only blanks.
- If parentheses are included in the text, they must be in matched pairs.

REWORK

is the level of this ++PRODUCT MCS, which was reworked for minor changes. Up to eight numeric characters can be specified.

REWORK is generally used only for ++PRODUCT statements supplied by IBM that have been reworked for minor changes. For these ++PRODUCT statements, the REWORK level is yyggydd, which is the year followed by the Julian date (for example, 2008110).

REWORK allows an updated ++PRODUCT MCS to be automatically received again, as long as it is more recent than the version that has already been received. This takes the place of rejecting the ++PRODUCT MCS and receiving it again.

Note: If a ++PRODUCT MCS appears more than once in the SMPPTFIN data set, the first occurrence may be received. However, none of the subsequent versions of the ++PRODUCT MCS are received, even if their rework level is higher than the one for the first version of the ++PRODUCT MCS.

Usage Notes

- The ++PRODUCT statements are processed whenever the SMPPTFIN data set is processed. This is true whether only selected SYSMODs are being processed or the entire SMPPTFIN data set is being processed.
- ++PRODUCT statements are not processed when only the SMPHOLD data set is being processed.
- The prodid, vv.rr.mm, DESCRIPTION, and SREL values are required and cannot be blank or null.
- The ++PRODUCT MCS is used in the SMPPTFIN data set and can be placed between, before, or after SYSMODs, ++FEATURE MCS, or ++PRODUCT MCS. It must be followed by one of the following: a ++APAR, ++ASSIGN MCS, ++FEATURE MCS, ++FUNCTION, ++PTF, ++USERMOD, another ++PRODUCT MCS, or an end-of-file. If one of these does not follow, SMP/E does not receive the SYSMOD being processed and it skips the ++PRODUCT MCS.

Example

Here is an example of a ++PRODUCT MCS for a product called OS/390.

```plaintext
++PRODUCT(5647-A01,2.5.0) /* Product definition */
DESCRIPTION(OS/390) /* Description */
URL(http://www.S390.ibm.com/os390/) /* URL */
SREL(Z038) /* SREL value */
PRODSUP((5645-001,01.04.00)) /* Product sups */
VENDOR(IBM) /* Product Vendor */.
```
++PROGRAM MCS

++PROGRAM MCS

The ++PROGRAM MCS describes a program element (a pre-built load module or a program object). It must immediately precede the load module or program object when they are within the SYSMOD. Use the ++PROGRAM when you want to ship executables as program parts. If you want to provide the object form of the module, use the ++MOD MCS instead. JCLIN is not used to define the program element.

Syntax

The syntax to be used depends on the processing to be performed for the element:
- Adding or replacing the element
- Deleting the element

Adding or replacing a program element

Deleting a program element

Operands

**ALIAS**

specifies an alternate name for the program object or load module.

**DELETE**

specifies that the program element and all aliases are to be removed from the target library and the distribution library.
++PROGRAM MCS

Notes:
1. DELETE is mutually exclusive with all other operands except DISTLIB and
   VERSION.
2. If the element statement is in a base function, you may want to use the
   DELETE operand on the ++VER MCS to delete the previous release, rather
   than on the element statement to delete a specific element.
3. Specification of the DELETE operand results in all aliases of the program
   element being deleted along with the program element identified.

DISTLIB
specifies the ddname of the distribution library for the specified program
element. During ACCEPT processing, SMP/E installs the program element into
the distribution library as a member. (The distribution library must be a PDS
or PDSE; it cannot be part of a UNIX file system.)

Notes:
1. DISTLIB must be specified when the program element is first installed.
2. If a program element entry already exists in the target zone or distribution
   zone and the value currently in that entry does not match that specified in
   the DISTLIB operand, the SYSMOD is not applied or accepted.

FROMDS
identifies the partitioned data set that contains this element.

Note: The FROMDS operand and its DSN, NUMBER, VOL, and UNIT
suboperands are included in the MCS generated by the BUILDMCS
command. IBM does not intend the FROMDS operand to be used in
manually coded MCS.

DSN
specifies the dsname of the FROMDS data set. The specified data set name
must conform to standard data set naming conventions and cannot contain
parentheses. The maximum length of the entire name is 44 characters
(including the periods).

NUMBER
specifies a number that SMP/E is to use when assigning a name to the
SMPTLIB data set associated with this FROMDS data set. (This is similar to
the way the relative file number is used in RELFILE processing.)

VOL
specifies, for an uncataloged data set, the volume serial number of the
volume containing the FROMDS data set. If specified, this volume
identifier must be from 1 to 6 alphanumeric characters.

VOL may be omitted for a cataloged data set.

UNIT
specifies, for an uncataloged data set, the UNIT type containing the
FROMDS data set. If specified, the UNIT value must be from 1 to 8
characters and must conform to standard UNIT naming conventions.
SMP/E accepts any nonblank characters specified between the open and
close parentheses, up to a maximum length of 8.

UNIT may be omitted for a cataloged data set.

Note: FROMDS is mutually exclusive with DELETE, RELFILE, and LKLIB.

LKLIB
is the ddname of the partitioned data set containing the program element. This
++PROGRAM MCS

operand is required if the program element is provided in a data set to which
the users have access, rather than inline or in RELFILE format.

Notes:
1. SMPFLIB cannot be used as a value on the LKLIB operand.
2. LKLIB is mutually exclusive with FROMDS and RELFILE.

name
specifies the name of the program element member. The name can contain any
uppercase alphabetic, numeric, or national ($, #, @) character and can be 1 to 8
characters long.

RELFILE
identifies which relative file associated with the SYSMOD contains this
element. This operand is required if you provide the element in RELFILE
format, rather than inline or in a LKLIB data set.

Notes:
1. The RELFILE value must be a decimal number from 1 to 9999.
2. RELFILE is mutually exclusive with FROMDS and LKLIB.

RMID
specifies the last SYSMOD that replaced this program element. This operand
was used only in a service-updated function, and the specified PTF must be
integrated into the function.

SYSLIB
specifies the ddname of the target library for the program element. (The target
library must be a PDS or PDSE; it cannot be part of a UNIX file system.)

During APPLY processing, the copy utility installs the program element into
the target library. During RESTORE processing, the copy utility copies the
program element from the distribution library member into the target library.

Note: SYSLIB must be specified when the program element is first installed.

VERSION
specifies one or more function SYSMODs that currently contain the element.
The function containing the element MCS takes over ownership of the element
from the specified functions.

When VERSION is specified on an element statement, it overrides any
VERSION operand values that might be specified on the ++VER MCS.

Usage Notes
• The target and distribution libraries for a program element must be a PDS (for
pre-built load module) or a PDSE (for a program object).
• If the program element is packaged inline, it must immediately follow the
++PROGRAM MCS and must not contain any records starting with ++. Neither
FROMDS, nor RELFILE, nor LKLIB can be specified on the ++PROGRAM
MCS.
• To be packaged inline, a program element must be unloaded along with its
aliases into a sequential data set and then transformed into the required
fixed-block-80 record format before it is packaged (see "Examples" on page 95
for a discussion of doing this). Later, when SMP/E installs the element, it is
changed back to its original format. For more information about using GIMDTS,
see "GIMDTS: Data transformation service routine" on page 450.
• If the program element is packaged in a LKLIB data set, the ddname specified in the LKLIB operand is required during APPLY and ACCEPT processing.

• For information about packaging program objects or pre-built load modules, see z/OS Packaging Rules.

Examples

Suppose you have a partitioned data set whose members contain a pre-built load module (LMODABC) and two aliases (ALIAS1 and ALIAS2). Further, suppose that you want to package LMODABC and its aliases inline in a PTF. The steps to package and install pre-built load module LMODABC in a PTF are as follows:

Step 1: Unload the partitioned data set member LMODABC and its aliases to create a sequential, VS format, data set.

```
//JOBx JOB...
//STEP1 EXEC PGM=IEBCOPY,REGION=512K
//SYSPRINT DD SYSOUT=*  
//SYSUT3 DD UNIT=SYSDA,SPACE=(TRK,(5,1))
//SYSUT4 DD UNIT=SYSDA,SPACE=(TRK,(5,1))
//INPUT DD DSN=userid.TEST.LOAD,DISP=SHR
//OUTPUT DD DSN=userid.UNLOAD.DATA,DISP=(NEW,CATLG),
// SPACE=(CYL,(20,10),RLSE),UNIT=SYSDA
//SYSIN DD *
COPY OUTDD=OUTPUT,INDD=INPUT
SELECT MEMBER=LMODABC
SELECT MEMBER=ALIAS1
SELECT MEMBER=ALIAS2
```

Step 2: Transform the unloaded data set into fixed-block-80 type records by using the SMP/E provided service routine GIMDTS.

```
//JOBx JOB...
//TFORM EXEC PGM=GIMDTS
//* ----- GIMDTS IS AN SMP/E LOAD MODULE.
//SYSPRINT DD SYSOUT=*  
//*  
//SYSUT1 DD DSN=userid.UNLOAD.DATA,DISP=SHR
//*  
//SYSUT2 DD DSN=userid.FB80.DATA,DISP=OLD
```

Step 3: Place the fixed block 80 records inline in the PTF following the ++PROGRAM MCS that identifies the element and its aliases to SMP/E.

```
++PTF(PTF0001).
++VER(Z038) FMID(DC00001).
++PROGRAM(LMODABC) ALIAS(ALIAS1,ALIAS2) DISTLIB(TGTLIB)
               SYSLIB(VEND001).
:: Place the data transformed by GIMDTS here.
::
```

Step 4: APPLY the PTF to install LMODABC and its aliases into the appropriate target library.

```
SET BDY(TGT1).
APPLY PTF(PTF0001).
```
During the APPLY step, SMP/E retransforms the inline data back into a variable spanned (VS) sequential data set. SMP/E then invokes the copy utility to copy the retransformed data into the target library. The program element and aliases are copied using a COPYMOD and SELECT statements.

After the APPLY, the target library contains LMODABC and its aliases.
The ++PTF MCS identifies a service SYSMOD. This type of SYSMOD can replace or update elements in target and distribution libraries, such as for a permanent correction, or it can add new elements. All other MCSs for this SYSMOD follow this header MCS. For more information about packaging a PTF, see z/OS Packaging Rules.

Syntax

++PTF MCS

**++PTF**(sysmod_id) DESCRIPTION(description) FILES(number) RFDSNPFX(relfile_prefix) REWORK(level)

Operands

DESCRIPTION

specifies a descriptive name to be associated with this SYSMOD. z/OS Packaging Rules manual.

- DESCRIPTION can also be specified DESC.
- The DESCRIPTION value can be in single-byte characters (such as English alphanumeric) or double-byte characters (such as Kanji).
- The DESCRIPTION value can contain up to 64 bytes of data, including blanks. (For double-byte data, the 64-byte maximum includes all shift-in and shift-out characters, as well as the double-byte characters.) Extra blanks, as well as leading and trailing blanks are deleted.
- The DESCRIPTION value can span multiple 80-byte records. Data must continue up to and including column 72 and begin in column 1 of the next line. All data past column 72 is ignored. The break does not translate to a blank unless a blank is explicitly coded in column 72 of the first line or in column 1 of the second line.
- If DESCRIPTION is specified, it must contain at least one non-blank character.
- If parentheses are included in the text, they must be in matched pairs.

FILES

specifies the number of relative files belonging to this PTF. It can be a decimal number from 1 to 9999. For information about packaging SYSMODs in relative files, see z/OS Packaging Rules.

Notes:

1. Although SMP/E allows you to package PTFs in relative files, they are not generally packaged in this format.
2. If a packager uses a high-level qualifier on RELFILE data sets, the RFDSNPFX operand on the header MCS (not the RFPREFIX operand on the RECEIVE command) must be used to identify that high-level qualifier.
++PTF MCS

REWORK specifies the level of this SYSMOD, which has been reworked for minor changes. Up to eight numeric characters can be specified.

For SYSMODs supplied by IBM, the REWORK level is \texttt{yyyyddd}, where \texttt{yyyy} is the year the SYSMOD was reworked and \texttt{ddd} is the Julian date.

REWORK allows an updated SYSMOD to be automatically received again, as long as it is more recent than the version that has already been received. This takes the place of rejecting the SYSMOD and receiving it again.

\textbf{Note:} If a SYSMOD appears more than once in the SMPPTFIN data set, the first occurrence may be received. However, none of the subsequent versions of the SYSMOD are received, even if their rework level is higher than the one for the first version of the SYSMOD. (Message GIM40001E is issued for each of the subsequent versions of the SYSMOD.)

RFDSNPFX identifies to SMP/E the prefix used in the relative file data set names for this SYSMOD. SMP/E uses this prefix when allocating data set names for the SYSMOD's relative files during RECEIVE processing.

- This operand can be specified only if the FILES operand is also specified.
- The RFDSNPFX value specified on the MCS statement must match the actual prefix used in the data set names for the associated relative files.

For example, if the names of the relative files created for a SYSMOD start with “IBM”, as in \texttt{IBM.sysmod_id.F1}, the header MCS statement for the SYSMOD must specify \texttt{RFDSNPFX(IBM)} so SMP/E knows which prefix to use when allocating the data set names for the SYSMOD's relative files during RECEIVE processing.

- Following standard data set naming conventions, the prefix can be from 1 to 8 alphanumeric or national ($, #, @) characters or a dash (–).

To enable full Security Server protection for tape data sets and to keep the tape header within the 17-character limit (including periods), you should limit the prefix to 1 to 3 characters. If the name exceeds the 17-character limit, only the rightmost 17 characters are written to the tape header label.

\texttt{sysmod_id} specifies a unique 7-character system modification identifier for the PTF. For more information, see “Naming conventions for SYSMODs” on page 517.

Usage Notes

If you want to update IBM-supplied code, you should use the ++USERMOD MCS rather than the ++PTF MCS. For more information, see “++USERMOD MCS” on page 115.

Examples

A PTF is required that replaces module IFBMOD01 for function FXY1040. The prerequisite service SYSMOD for the module is PTF UZ00004. The APAR incident fixed by this PTF is AZ12345. Here is an example of a SYSMOD containing a ++PTF statement to accomplish this:

\begin{verbatim}
++PTF(UZ00006) /* Identify the PTF number. */.
++VER(Z038) FMID(FXY1040) /* for MVS function FXY1040 */
PRE(UZ00004) /* with a prerequisite. */.
SUP(AZ12345) /* Fixes one APAR. */.
++MOD(IFBMOD01) /* This module */.
\end{verbatim}
++PTF MCS

DISTLIB(AOSFB) /* from this DLIB. */.

... object deck for IFBMOD01

...
The ++RELEASE MCS removes a previously held SYSMOD from exception SYSMOD status. ++RELEASE statements are processed by the RECEIVE command.

Syntax

```
++RELEASE MCS
```

```
++RELEASE(sysmod_id)
   ERROR
   FIXCAT
   USER
   REASON(reason_id)
   FMID(fmid)
   DATE(yyddd)

```

SYSTEM reason IDs used by IBM:

- ACTION
  - AD
  - DB2BIND
  - DDDEF
  - DELETE
  - DEP
  - DOC
  - DOWNLO
  - EC
  - ENH
  - EXIT
  - EXRF
  - FULLGEN
  - IGEN
  - IPL
  - MSGSKEL
  - MULTSYS
  - RESTART

Operands

- ERROR, FIXCAT, SYSTEM, or USER
  - Specifies the hold category from which the SYSMOD is to be removed. You must specify one of these categories.
  - ERROR
    - An APAR reported an error in the SYSMOD. The SYSMOD should not be applied or accepted until the APAR is resolved. A PTF held for this reason is also called a program error PTF, or PE-PTF. SMP/E automatically resolves the APAR and allows the SYSMOD to be applied or accepted when a SYSMOD that either matches or supersedes the APAR is also applied or accepted. Error holds can be read only from the SMPHOLD data set.
++RELEASE MCS

Note: ERROR can also be specified as ERR.

**FIXCAT**
An APAR provides a fix for the held SYSMOD and the fix is associated with one or more Fix Categories. It is optional whether the APAR will affect processing for the held SYSMOD, based on the APAR's Fix Categories and the Fix Categories of interest specified by the user. If any one or more Fix Categories for the APAR match any of those of interest to the user, then the held SYSMOD will not be applied or accepted until the APAR is resolved. The APAR is resolved when a SYSMOD that matches the APAR name, or a SYSMOD that supersedes the APAR, is applied or accepted. FIXCAT holds can be read only from the SMPHOLD data set.

See the FIXCAT operand for APPLY and ACCEPT command processing for details of specifying the Fix Categories of interest.

**SYSTEM**
Special action outside normal SMP/E processing is required for the SYSMOD. Examples are SYSMODs requiring a SYSGEN after they are installed, or SYSMODs requiring the installation of an associated engineering change (EC) level. System holds can appear in the SYSMOD itself or in the SMPHOLD data set.

Note: SYSTEM can also be specified as SYS.

**USER**
The SYSMOD requires special processing because of a decision you have made. User holds can be read only from the SMPHOLD data set.

**DATE**
specifies the date that the ++HOLD MCS was generated.

**FMID**
specifies the FMID to which the held SYSMOD is applicable. For external HOLDDATA (a ++HOLD statement not within a SYSMOD), this information allows SMP/E to receive only those statements associated with FMIDs defined in the user's global zone. This operand is required.

**REASON**
identifies the HOLD reason ID that is to be removed from the SYSMOD. This field is required.

- **An error reason ID** is the number of the APAR that caused the SYSMOD to be placed in exception status.
- **A fix category reason ID** is the SYSMOD ID for the APAR that caused the SYSMOD to be placed into exception status.
- **A system reason ID** is a 1- to 7-character string used to identify some special processing the SYSMOD requires. These are the specific values currently used by IBM:

<table>
<thead>
<tr>
<th>ID</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTION</td>
<td>The SYSMOD needs special handling before or during APPLY processing, ACCEPT processing, or both.</td>
</tr>
<tr>
<td>AO</td>
<td>The SYSMOD may require action to change automated operations procedures and associated data sets and user exits in products or in customer applications. The PTF cover letter describes any changes (such as to operator message</td>
</tr>
</tbody>
</table>
### ++RELEASE MCS

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB2BIND</td>
<td>A DB2 application REBIND is required for the SYSMOD to become effective.</td>
</tr>
<tr>
<td>DDDEF</td>
<td>Data set changes or additions as required.</td>
</tr>
<tr>
<td>DELETE</td>
<td>The SYSMOD contains a ++DELETE MCS, which deletes a load module from the system.</td>
</tr>
<tr>
<td>DEP</td>
<td>The SYSMOD has a software dependency.</td>
</tr>
<tr>
<td>DOC</td>
<td>The SYSMOD has a documentation change that should be read before the SYSMOD is installed.</td>
</tr>
<tr>
<td>DOWNLD</td>
<td>Code that is shipped with maintenance that needs to be downloaded.</td>
</tr>
<tr>
<td>DYNACT</td>
<td>The changes supplied by the SYSMOD may be activated dynamically without requiring an IPL. The HOLD statement describes the instructions required for dynamic activation. If those instructions are not followed, then an IPL is required for the SYSMOD to take effect.</td>
</tr>
<tr>
<td>EC</td>
<td>The SYSMOD needs a related engineering change.</td>
</tr>
<tr>
<td>ENH</td>
<td>The SYSMOD contains an enhancement, new option or function. The HOLD statement provides information to the user regarding the implementation and use of the enhancement.</td>
</tr>
<tr>
<td>EXIT</td>
<td>The SYSMOD contains changes that may affect a user exit. For example, the interface for an exit may be changed, an exit may need to be reassembled, or a sample exit may be changed.</td>
</tr>
<tr>
<td>EXRF</td>
<td>The SYSMOD must be installed in both the active and the alternative Extended Recovery Facility (XRF) systems at the same time to maintain system compatibility. (If you are not running XRF, you should bypass this reason ID.)</td>
</tr>
<tr>
<td>FULLGEN</td>
<td>The SYSMOD needs a complete system or subsystem generation to take effect.</td>
</tr>
<tr>
<td>IOGEN</td>
<td>The SYSMOD needs a system or subsystem I/O generation to take effect.</td>
</tr>
<tr>
<td>IPL</td>
<td>The SYSMOD requires an IPL to become effective. For example, the SYSMOD may contain changes to LPA or NUCLEUS, the changes may require a CLPA, or a failure to perform an IPL might lead to catastrophic results, such as could be caused by activation of a partial fix.</td>
</tr>
</tbody>
</table>

**Note:** If you plan to perform an IPL with CLPA after the SYSMOD has been applied, then no further investigation of the HOLD is required; simply bypass the IPL reason ID. However, if you are not planning to perform an IPL with CLPA, then the details of the HOLD statement must be investigated to determine what kind of actions are required to activate the SYSMOD.
++RELEASE MCS

MSGSKEL  This SYSMOD contains message changes that must be compiled for translated versions of the message changes to become operational on extended TSO consoles.

If you want to use translated versions of the messages, you must run the message compiler once for the library containing the English message outlines, and once for each additional language your system. For details, see z/OS MVS Planning: Operations.

If you want to use only the English version of the messages, you do not need to run the message compiler. You should bypass this reason ID.

MULTSYS  Identifies fixes that need to be applied to multiple systems, in one of three cases: preconditioning, coexistence, or exploitation.

RESTART  To become effective, the SYSMOD requires a special subsystem restart operation. The HOLD statement contains information regarding the required restart actions.

- A user reason ID is defined by the user.

For additional information, see “Naming conventions for HOLD reason IDs and HOLD classes” on page 514.

sysmod_id specifies that SMP/E is to remove the identified SYSMOD from exception SYSMOD status. This operand is required.

Usage Notes
- ++RELEASE statements are not allowed within a SYSMOD. They are allowed only in SMPHOLD.
- ++RELEASE statements unconditionally remove a SYSMOD from exception status and should, therefore, be used with caution. To install a SYSMOD that is currently in exception status, you should probably not create and process a ++RELEASE statement, but rather use the appropriate BYPASS operand of the APPLY or ACCEPT command.
- ++RELEASE statements do not affect ++HOLD statements within a SYSMOD (internal HOLDDATA). However, SMP/E can ignore this internal HOLDDATA during APPLY or ACCEPT processing if BYPASS(HOLDSYS) or BYPASS(HOLDUSER) is specified.

Examples
The following examples are provided to help you use the ++RELEASE MCS:

Example 1: Removing a SYSMOD from HOLDUSER status
Here is an example of a ++HOLD statement that holds the PTF until after some event (such as new hardware) occurs:

```
++HOLD  (UZ12345)  /* Put this PTF */
         FMID(FXY1040)  /* for this function */
         USER  /* into hold user status */
         REASON(CPU0A)  /* for CPU 0A update. */
         COMMENT(I DO NOT WANT THIS TO GO ON
               UNTIL AFTER THE CPU CHANGE)  /* */.
```

When the CPU change is made, the following sample ++RELEASE statement allows the PTF to be installed:

```
++RELEASE MCS
```
++RELEASE MCS

++RELEASE (UZ12345) /* Remove this PTF */
FMID(FXY1040) /* for this function */
USER /* from hold user status */
REASON(CPU0A) /* for CPU 0A update. */.

That PTF is now eligible for normal installation.

Example 2: Incorrect use of ++RELEASE
Assume the following ++HOLD MCS was processed as part of the normal preventive service installation:

++HOLD (UZ12345) /* Put this PTF */
FMID(FXY1040) /* for this function */
ERROR /* into hold error status */
REASON(AZ12345) /* for the APAR. */
COMMENT(WHEN RUNNING PRODUCT XYZ and OC4 ABEND MAY OCCUR) /* */.

You are running one system with product XYZ installed and one without product XYZ. The PTF provides a fix for another problem you are encountering on the system without product XYZ; so you want to install this PTF on that system. Here is an example of a ++RELEASE statement that lets you apply PTF UZ12345 without having to use the BYPASS operand:

++RELEASE (UZ12345) /* Remove this PTF */
FMID(FXY1040) /* for this function */
ERROR /* from hold error status */
REASON(AZ12345) /* for the APAR. */.

The risk with this method of processing is that SMP/E no longer has any record of PTF UZ12345 being in exception status. Therefore, the next time any modifications are installed on the system with product XYZ installed, the PTF is installed, introducing a potential OC4 problem into that system.

The correct way to install the PTF on the system without product XYZ is to use the following command:

SET BDY(MVSTST1) /* Process MVSTST1 tgt zone. */.
APPLY S(UZ12345) /* Process the PTF. */
BYPASS(HOLDERR(AZ12345)) /* Bypass known error. */.

Now the PTF is installed on one system, but SMP/E still remembers that it is in hold error status and does not allow it to be installed on any other system.

Note: Any other applicable operand (FORFMID, SOURCEID, and so on) can be used in place of the SELECT or S operand.

Example 3: A ++RELEASE statement with a FIXCAT HOLD
Here is an example of the use of the FIXCAT hold category on a ++RELEASE statement:

++RELEASE (HBB7720) /* Remove this PTF */
FMID(HBB7720) /* for this function */
REASON(AK18603) /* for this APAR */
FIXCAT /* from fix category hold status */
DATE(06231) /* on this date */.

Note: Any other applicable operand (FORFMID, SOURCEID, and so on) can be used in place of the SELECT or S operand.
During APPLY processing, the ++RENAME MCS renames a load module in all the target libraries and the LMOD entry in the target zone. It also updates the LMOD subentry of the MOD entry for each module in the load module. For load modules having a CALLLIBS subentry (or having a SYSLIB allocation), SMP/E also renames the base version of the load module in the SMPLTS library. For load modules having a side deck library subentry, SMP/E also renames the definition side deck in the side deck library.

During ACCEPT processing, if inline JCLIN was processed, the distribution zone entries are updated to reflect the ++RENAME changes. The target libraries and target zone entries are not updated.

Notes:
1. You cannot use ++RENAME to rename a load module for which symbolic links have been defined.
2. You cannot use ++RENAME to rename macros, modules, source, or other elements. SMP/E does, however, support these ways to rename an element:
   - For elements other than modules, you must use an element MCS statement to delete the original element, and another element MCS statement to reintroduce it using the new name. You can do both of these steps in the same SYSMOD. For examples, see the descriptions of the various element MCS statements in this chapter.
   - For modules, you must use a ++MOD statement to delete the original module, and another ++MOD statement to reintroduce it using the new name. In addition, you must ensure that the renamed module is included in the appropriate load modules, either through JCLIN or the LMOD operand on the ++MOD statement. You can do all of these steps in the same SYSMOD. For an example, see "Example 3: Packaging a renamed module" on page 83 in this chapter.

Syntax

```
++RENAME MCS

>>>++RENAME(old_name)TONAME(new_name)<<<
```

Operands

`old_name`

specifies the name of the load module to be renamed.

`TONAME`

specifies the new name for the load module.

Usage Notes

- There are no optional operands. You must specify the load module name and its new name.
- ++RENAME statements must follow any ++VER and ++IF statements, and must precede any element MCSs.
Regardless of the order in which ++MOVE, ++RENAME, and ++DELETE statements are coded in a SYSMOD, they are processed in the following order:

- **APPLY** and **ACCEPT**
  1. ++MOVE
  2. ++RENAME
  3. ++DELETE
- **RESTORE**
  1. ++RENAME
  2. ++MOVE

++JCLIN statements are processed next, followed by element statements.

**Examples**

Here is an example of a SYSMOD containing a ++RENAME statement renaming load module LMODA:

```plaintext
++PTF(UR01234) /* Identify the PTF number. */
++VER(Z038) FMID(HXY1300) /* For MVS function HXY1300.*/
++IF (ESY1300) THEN /* If ESY1300 is installed */
  REQ(UR12399) /* UR12399 is required. */
++RENAME (LMODA) /* Rename load module LMODA */
  TONAME(LMODBB) /* to LMODBB. */
++JCLIN /* JCLIN follows. */
++MOD(MODAA) DISTLIB(AOSXX) /* Element MCS statements. */
```
++SRC MCS

The ++SRC MCS describes a single source replacement. It must immediately precede the source definition statements when they are within the SYSMOD.

You should use the ++SRC MCS when you want to provide the source form of a module and have it get assembled when the SYSMOD is installed. If you want to provide the object form of the module, use the ++MOD MCS instead.

Syntax
The syntax to be used depends on the processing to be done for the element:
- Adding or replacing the element
- Deleting the element

Adding or replacing source

```plaintext
++SRC MCS
++SRC(name)
  DISTLIB(ddname)
  DISTMOD(ddname)
  RELFILE(number)
  TXLIB(ddname)
  SSI(code)
  FROMDS(DSN(dsn)NUMBER(n)
    VOL(vol)
    UNIT(unit))
  SYSLIB(ddname)
  UMID(sysmod_id)
  VERSION(sysmod_id)
```

Deleting source

```plaintext
++SRC MCS
++SRC(name)
  DELETE
  DISTLIB(ddname)
  VERSION(sysmod_id)
```

Operands

DELETE
indicates that the source is to be removed from the target libraries, distribution libraries, and SMP/E data sets.

Note: DELETE is mutually exclusive with all other operands except DISTLIB and VERSION.

DISTLIB
specifies the ddname of the distribution library for the specified source.
**Note:** DISTLIB must be specified if the source has not been previously recorded on the target zone or distribution zone. If a SRC entry already exists in the target zone or distribution zone, and the value currently in that entry does not match that specified in the DISTLIB operand, the SYSMOD is not applied or accepted, unless that SYSMOD also used the ++MOVE MCS to change the DISTLIB to that new value.

**DISTMOD**

specifies the ddname of the link-edit distribution library for the assembled source code. During ACCEPT processing, the object code from the assembler is link-edited to the library specified.

**FROMDS**

identifies the partitioned data set that contains this element.

**Note:** The FROMDS operand and its DSN, NUMBER, VOL, and UNIT suboperands are included in the MCS generated by the BUILDMCS command. IBM does not intend the FROMDS operand to be used in manually coded MCS.

**DSN**

specifies the dsname of the FROMDS data set. The specified data set name must conform to standard data set naming conventions and cannot contain parentheses. The maximum length of the entire name is 44 characters (including the periods).

**NUMBER**

specifies a number that SMP/E is to use when assigning a name to the SMPTLIB data set associated with this FROMDS data set. (This is similar to the way the relative file number is used in RELFILE processing.)

**VOL**

specifies, for an uncataloged data set, the volume serial number of the volume containing the FROMDS data set. If specified, this volume identifier must be from 1 to 6 alphanumeric characters.

VOL may be omitted for a cataloged data set.

**UNIT**

specifies, for an uncataloged data set, the UNIT type containing the FROMDS data set. If specified, the UNIT value must be from 1 to 8 characters and must conform to standard UNIT naming conventions. IBM SMP/E for z/OS, V3R6 accepts any nonblank characters specified between the open and close parentheses, up to a maximum length of 8.

UNIT may be omitted for a cataloged data set.

**Note:** FROMDS is mutually exclusive with DELETE, RELFILE, SSI, and TXLIB.

**name**

specifies the name of the source in the distribution library and, optionally, in the target library. The name can contain any alphanumeric characters and $, #, @, or hex C0.

**RELFILE**

identifies which relative file associated with the SYSMOD contains this element. This operand is required if you provide the element in RELFILE format, rather than inline or in a TXLIB data set.
++SRC MCS

Notes:
1. SMPTLIB cannot be used as a value on the TXLIB operand.
2. RELFILE is mutually exclusive with FROMDS and TXLIB.

RMID
specifies the last SYSMOD that replaced this source. This operand can be used only in a service-updated function, and the specified PTF must be integrated into the function.

SSI
specifies eight hexadecimal digits of system status information. This information is placed in the directory of the target system library or SMPMTS or SMPSTS during APPLY processing, and in the distribution library during ACCEPT processing, as four packed hexadecimal bytes of user data. See the IEBUPDTE program description in z/OS DFSMSdfp Utilities, SC26-7414.

Note: This operand is ignored if text is located in a library, as is the case when either the FROMDS, RELFILE, or TXLIB operand is specified.

SYSLIB
specifies the ddname of the target library, if the source module exists in one. APPLY and RESTORE processing update this library.

Note: If a SRC entry already exists in the target zone or distribution zone and the value currently in that entry does not match that specified in the SYSLIB operand, SMP/E ignores the SYSLIB value in the SYSMOD being installed, unless that SYSMOD also used the +MOVE MCS to change the SYSLIB to that new value.

TXLIB
is the ddname of the partitioned data set containing the source. This operand is required if the module is provided in a data set the users have access to, rather than inline or in RELFILE format.

Notes:
1. SMPTLIB cannot be used as a value on the TXLIB operand.
2. TXLIB is mutually exclusive with FROMDS and RELFILE.

UMID
specifies the UMIDs of the source. This operand can be used only in function SYSMODs and specifies the PTF service level of the source (the set of SYSMODs that have updated this source since it was last replaced).

VERSION
specifies one or more function SYSMODs that currently contain the element. The function containing the ++SRC MCS takes over ownership of the element from the specified functions.

When VERSION is specified on an element statement, it overrides any VERSION operand values that might be specified on the ++VER MCS.

Usage Notes
- If the source is packaged inline, it must immediately follow the ++SRC MCS and must not contain any records starting with ++. Neither FROMDS, nor RELFILE, nor TXLIB can be specified.
- If the source is in a TXLIB data set, the ddname specified on the TXLIB operand is required during APPLY and ACCEPT processing.
Examples

The following examples are provided to help you use the ++SRC MCS:

Example 1: Adding a new source to the system

A replacement for the source IFBSRC01 is in a partitioned data set referred to by the ddname REPLACE. The distribution library for the source is SYS1.IFBSRC; SYS1.AOS23 is the distribution library for the module, IFBSRC01, resulting from the assembly of the source, IFBSRC01.

Here is an example of a SYSMOD containing a ++SRC statement that causes SMP/E to install the source code, assemble it, and save the resulting module in the DLIBs:

```
++USERMOD(USR0001) /* User modification */.
++VER(Z038) FMID(FXY1040) /* for user function in MVS. */.
++SRC(IFBSRC01) /* Replace source */.
    DISTLIB(IFBSRC) /* in this DLIB. */
    DISTMOD(AOS23) /* MOD goes in this DLIB. */
    TXLIB(REPLACE) /* Replacement SRC is here. */.
```

The following DD statements are required when the SYSMOD is applied:

```
//REPLACE DD DSN=...
//SMPSTS DD DSN=SYS1.SMPSTS,DISP=OLD
```

plus whatever other DD statements are required based on which load modules the assembled source is to be linked to. These load modules should be identified by JCLIN, either as a separate step or within the SYSMOD itself. Assuming the load module was composed of only this one module, the following ++JCLIN MCS can be added to the SYSMOD after the ++VER statement.

```
++JCLIN. /* JCLIN to get SRC linked. */
    //JOB1 JOB 'accounting info',MSGLEVEL=(1,1)
    //STEP1 EXEC PGM=IEBCOPY
    //AOS23 DD DSN=SYS1.AOS23,DISP=SHR
    //LPALIB DD DSN=SYS1.LPALIB,DISP=SHR
    //SYSIN DD *
    COPY INDD=AOS23,OUTDD=LPALIB
    SELECT M=(IFBSRC01)
    /*

In this case, you also need the following DD statement when the SYSMOD is applied:

```
//LPALIB DD DSN=SYS1.LPALIB,DISP=OLD
```

The following DD statements are required when the SYSMOD is accepted:

```
//REPLACE DD DSN=...
//IFBSRC DD DSN=SYS1.IFBSRC,DISP=OLD
//AOS23 DD DSN=SYS1.AOS23,DISP=OLD
```

Example 2: Packaging renamed source

Suppose that, for some reason, you need to rename source IFBSRC01, which was introduced in "Example 1: Adding a new source to the system." The new name is to be USRSRCXX. You do not need to change anything else about the source. Because the original source was linked by JCLIN, you must provide updated
JCLIN to link the renamed version of the source. Here is an example of a SYSMOD containing the ++SRC statements and JCLIN needed to package this renamed source:

```plaintext
++USERMOD(USR0002) /* User modification */.
++VER(Z038) FMID(FXY1040) /* for user function in MVS. */.
    PRE(USR0001) /* Base on previous USERMOD. */.
++SRC(IFBSRC01) /* Delete the original */.
    DELETE /* source. */.
++SRC(USRSRCXX) /* Add renamed source */.
    DISTLIB(IFBSRC) /* to this DLIB. */.
    DISTMOD(AOS23) /* MOD goes in this DLIB. */.
    TXLIB(REPLACE) /* Replacement SRC is here. */.
...
++JCLIN. /* JCLIN to get SRC linked. */.
//JOB1 JOB 'accounting info',MSGLEVEL=(1,1)
//STEP1 EXEC PGM=IEBCOPY
//A0523 DD DSN=SYS1.A0523,DISP=SHR
//LPALIB DD DSN=SYS1.LPALIB,DISP=SHR
//SYSIN DD *
    COPY INDD=A0523,OUTDD=LPALIB

SELECT M=(USRSRCXX)
/*

You must ensure that the renamed source is in the TXLIB library (REPLACE) used to provide SMP/E with the element.

The following DD statements are required when the SYSMOD is applied:

```
//REPLACE DD DSN=...
//SMPSTS DD DSN=SYS1.SMPSTS,DISP=OLD
//LPALIB DD DSN=SYS1.LPALIB,DISP=OLD
```

The following DD statements are required when the SYSMOD is accepted:

```
//REPLACE DD DSN=...
//IFBSRC DD DSN=SYS1.IFBSRC,DISP=OLD
//A0523 DD DSN=SYS1.A0523,DISP=OLD
```
++SRCUPD MCS

++SRCUPD MCS describes a single set of source update statements within a PTF, an APAR fix, or a USERMOD. It must immediately precede the source update statements within the SYSMOD.

Syntax

```
++SRCUPD MCS

++SRCUPD(name)

DISTLIB(ddname)

DISTMOD(ddname)

SYSLIB(ddname)
```

Operands

**name**

specifies the name of the source in the distribution library and, optionally, in the target library. The name can contain any alphanumeric characters and $, #, @, or hex C0.

**DISTLIB**

specifies the ddname of the distribution library for the specified source.

**Note:** DISTLIB must be specified if the source has not been previously recorded on the target zone or distribution zone. If a SRC entry already exists in the target zone or distribution zone, and the value currently in that entry does not match that specified in the DISTLIB operand, the SYSMOD is not applied or accepted.

**DISTMOD**

specifies the ddname of the link-edit distribution library for the assembled source code. During ACCEPT processing, the object code from the assembler is link-edited to the library specified.

**SYSLIB**

specifies the ddname of the target library, if the source exists in one. APPLY and RESTORE processing update this library.

Usage Notes

- If a SYSMOD containing a ++SRCUPD statement attempts to change the ownership (FMID) of the element (with the VERSION operand), the SYSMOD cannot be installed.
- The changes for the source must immediately follow the ++SRCUPD MCS and must not contain any records starting with ++.
- The only IEBUPDTE control statements allowed in a SYSMOD are ./ CHANGE and ./ ENDUP.
- The only IEBUPDTE CHANGE operand that SMP/E checks is NAME, which must specify the same element as the ++SRCUPD MCS. Other CHANGE operands may produce undesired results and are used at your own risk. For example, if you code UPDATE=INPLACE, SMP/E may update the distribution library. Once the distribution libraries are changed, there is no way to remove the updates.
- When processing multiple updates to the same lines in a given source module, SMP/E uses the ./ CHANGE statement from the last update to the lines.
- SMP/E does not support a continuation of the ./ CHANGE statement.

Examples
The following examples are provided to help you use the ++SRCUPD MCS:

Example 1: Updating an existing source
Here is an example of a SYSMOD containing a ++SRCUPD statement to make a modification to an IBM source module, in this case module JESMOD01, which is part of product EJS1102:

```plaintext
++USERMOD(MY00001) /* User modification */.
++VER(Z038) FMID(EJS1102) /* for MVS JES. */.
PRE(UZ12345) /* Current service level. */.
++SRCUPD(JESMOD01) /* Update this JES MOD. */.
/* DISTLIB already known. */.
./ CHANGE NAME=JESMOD01
LA R1,MYPARM 00001000
BALR MYPGM 00001100
MYPGM EQU * 00500000
... 00500100
... 00500200
B R14 00500300
```

Example 2: Making subsequent source updates
Assume that, after installing the modification given in the first example, you need to make another modification. Assume the following lines had to be changed:

```plaintext
... 00500150
BALR MYPGM2 00500160
MYPGM2 EQU * 00600000
... 00600100
... 00600200
B R14 00600300
```

You have two choices as to the method of building the second modification:
1. The first method is to build the second SYSMOD to contain only the newly changed lines of code, and then specify the PRE operand in both the current IBM service level and the previous user modification.
2. The second method is to build the second SYSMOD to contain all the user modifications to this module; then specify the PRE operand in the current IBM service level and supersede the previous user modification.

Here is an example of a SYSMOD for the first method:

```plaintext
++USERMOD(MY00002) /* User modification */.
++VER(Z038) FMID(EJS1102) /* for MVS JES. */.
PRE(UZ12345) /* Current service level */.
MY00001 /* plus previous USERMOD. */.
++SRCUPD(JESMOD01) /* Update this JES MOD. */.
/* DISTLIB already known. */.
./ CHANGE NAME=JESMOD01
... 00500150
BALR MYPGM2 00500160
MYPGM2 EQU * 00600000
... 00600100
... 00600200
B R14 00600300
```

Here is an example of a SYSMOD for the second method:
The advantages to the second method are that all modifications to one source are contained in a single SYSMOD, and if that SYSMOD has to be restored or re-applied, processing is much more efficient.

**Note:** The SMP/E ++SRCUPD MCS is the same for both methods; only the ++VER MCS and the actual update control cards change.
++USERMOD MCS

The ++USERMOD MCS identifies a user modification. This type of SYSMOD can be used to add user-defined functions or to replace or update elements for IBM-supplied code in the target or distribution libraries. All other MCSs for this SYSMOD follow this header MCS. For more information about packaging a USERMOD, see SMP/E User's Guide.

Syntax

```
++USERMOD MCS
++USERMOD(sysmod_id) DESCRIPTION(description)
FILES(number) RFDSNPFX(relfile_prefix) REWORK(level)
```

Operands

**DESCRIPTION**

specifies a descriptive name to be associated with this SYSMOD. z/OS Packaging Rules manual.

- DESCRIPTION value can also be specified DESC.
- The DESCRIPTION value can be in single-byte characters (such as English alphanumeric) or double-byte characters (such as Kanji).
- The DESCRIPTION value can contain up to 64 bytes of data, including blanks. (For double-byte data, the 64-byte maximum includes all shift-in and shift-out characters, as well as the double-byte characters.) Extra blanks, as well as leading and trailing blanks are deleted.
- The DESCRIPTION value can span multiple 80-byte records. Data must continue up to and including column 72 and begin in column 1 of the next line. All data past column 72 is ignored. The break does not translate to a blank unless a blank is explicitly coded in column 72 of the first line or in column 1 of the second line.
- If DESCRIPTION is specified, it must contain at least one non-blank character.
- If parentheses are included in the text, they must be in matched pairs.

**FILES**

specifies the number of relative files belonging to this USERMOD. It can be a decimal number from 1 to 9999. For information about packaging SYSMODs in relative files, see z/OS Packaging Rules.

Notes:

1. Although SMP/E allows you to package USERMODs in relative files, they are not generally packaged in this format.
2. If a packager uses a high-level qualifier on RELFILE data sets, the RFDSNPFX operand on the header MCS (not the RFPREFIX operand on the RECEIVE command) must be used to identify that high-level qualifier.
**++USERMOD MCS**

**REWORK**
specifies the level of this SYSMOD, which was reworked for minor changes. Up to eight numeric characters can be specified.

For SYSMODs supplied by IBM, the REWORK level is `yyyyddd`, where `yyyy` is the year the SYSMOD was reworked and `ddd` is the Julian date.

REWORK allows an updated SYSMOD to be automatically received again, as long as it is more recent than the version that has already been received. This takes the place of rejecting the SYSMOD and receiving it again.

**Note:** If a SYSMOD appears more than once in the SMPPTFIN data set, the first occurrence may be received. However, none of the subsequent versions of the SYSMOD are received, even if their rework level is higher than the one for the first version of the SYSMOD. (Message GIM40001E is issued for each of the subsequent versions of the SYSMOD.)

**RFDSNPFX**
identifies to SMP/E the prefix used in the relative file data set names for this SYSMOD. SMP/E uses this prefix when allocating data set names for the SYSMOD's relative files during RECEIVE processing.

- This operand can be specified only if the FILES operand is also specified.
- The RFDSNPFX value specified on the MCS statement must match the actual prefix used in the data set names for the associated relative files.

For example, if the names of the relative files created for a SYSMOD start with "IBM", as in `IBM.sysmod_id.F1`, the header MCS statement for the SYSMOD must specify `RFDSNPFX(IBM)` so SMP/E knows which prefix to use when allocating the data set names for the SYSMOD's relative files during RECEIVE processing.

- Following standard data set naming conventions, the prefix can be from 1 to 8 alphanumeric or national ($, #, @) characters or a dash (-).

To enable full Security Server protection for tape data sets and to keep the tape header within the 17-character limit (including periods), you should limit the prefix to 1 to 3 characters. If the name exceeds the 17-character limit, only the rightmost 17 characters are written to the tape header label.

**sysmod_id**
specifies a unique 7-character system modification identifier for the USERMOD. For more information, see “Naming conventions for SYSMODs” on page 517.

**Usage Notes**
If you have updated an element, SYSMODs other than functions cannot replace it unless you explicitly allow them to by bypassing MODID checking. However, if you install a function, it may overlay your user modifications. SMP/E issues a warning message when it detects this condition.

**Examples**
A source (IFBSRC02) owned by function SYSMOD FXY1040 is to be modified. Your modification requires a service level provided UZ00007; you are only updating, not replacing, the source. You have chosen a SYSMOD ID of MY00001. Here is an example of a SYSMOD containing a ++SRCUPD statement that accomplishes this:

```
++USERMOD(MY00001) /* USERMOD number */.
++VER(Z038) FMID(FXY1040) /* for function FXY1040 */.
PRE(UZ00007) /* at this service level. */.
```
++SRCUPD(IFBSRC02) /* Update this source */
DISTLIB(IFBSRC) /* in this DLIB. */
./ CHANGE NAME=IFBSRC02
...
... update control cards
...

For additional examples of USERMODs, see "++SRCUPD MCS" on page 112 and
SMP/E User's Guide
+++VER MCS

+++VER MCS

The +++VER MCS describes the environment required for receiving and installing a SYSMOD. A SYSMOD must contain a separate +++VER MCS for each environment to which it applies. At least one +++VER MCS must be present in a SYSMOD, and a maximum of 255 +++VER statements are allowed for each SYSMOD.

Syntax

```
+++VER MCS

+++VER(srel)

DELETE(sysmod_id)

FMID(sysmod_id)

NPRE(sysmod_id)

PRE(sysmod_id)

REQ(sysmod_id)

SUP(sysmod_id)

VERSION(sysmod_id)
```

Operands

**DELETE**

Indicates which function SYSMODs should be deleted when this function is installed. These functions are permanently deleted and cannot be restored.

**DELETE** can be specified only in function SYSMODs.

The same SYSMOD can be specified on both DELETE and SUP. This cleans up entries for the deleted function, and, at the same time, allows SYSMODs that name the deleted function as a requisite to still be installed.

SYSMODs specified in the DELETE operand do not have to be specified in VERSION operands of +++VER, +++MAC, +++SRC, or +++MOD statements.

For additional information about the effect of DELETE during APPLY and ACCEPT processing, see [SMP/E Commands](#).

**FMID**

FMID identifies the function to which a SYSMOD applies. **FMID** must be specified for all SYSMODs except base functions.

The following considerations relate to the FMID operand:

- Unlike prerequisites specified by the PRE operand, the functional prerequisite specified by the FMID operand is satisfied only by the specified SYSMOD. It is not satisfied by another SYSMOD that supersedes that function.
++VER MCS

- When specified on the ++VER MCS for a function, FMID defines the function as a dependent function. In this case, FMID indicates that the elements supplied by the dependent function SYSMOD are functionally higher than the specified base function.

  A function cannot be both a base function and a dependent function. Therefore, if a base function contains more than one ++VER MCS, none of them can specify the FMID operand. Likewise, if a dependent function contains more than one ++VER MCS, all of them must specify the FMID operand.

- When specified on the ++VER MCS for a non-function SYSMOD, FMID indicates the functional level of all elements in the SYSMOD.

- SMP/E RECEIVE processing does not receive a dependent function unless the FMID of the base is already present in the global zone or BYPASS(FMID) is specified. For more information about packaging of dependent SYSMODs, refer to RECEIVE processing in *SMP/E Commands*.

NPRE

indicates which function SYSMODs cannot exist in the same zone as this function. These are negative prerequisite SYSMODs. The current SYSMOD cannot be applied or accepted if any of the listed SYSMODs are already present.

This operand has no effect on RECEIVE eligibility.

NPRE can only be specified within a function SYSMOD.

PRE

indicates which SYSMODs are prerequisites for this SYSMOD. A prerequisite SYSMOD must either be already installed, or must be installed concurrently with this SYSMOD. For additional information about how prerequisites are resolved during APPLY and ACCEPT processing, see *SMP/E Commands*.

If a SYSMOD replaces an existing element, the PRE operand must specify the previous SYSMOD that replaced the element (RMID) and all the SYSMODs that have updated the element (UMIDs) since it was last replaced.

If a SYSMOD updates an existing element, the PRE operand must specify the previous SYSMOD that replaced the element. It should also specify the last SYSMOD that updated the element since then.

This operand has no effect on RECEIVE eligibility.

REQ

indicates which SYSMODs are requisites for this SYSMOD. The specified SYSMOD must either be already installed, or must be installed concurrently with this SYSMOD. If the specified SYSMOD also specifies this SYSMOD as a requisite, these two SYSMODs are corequisites, and neither can be installed independently; they must be installed within the same APPLY and ACCEPT command. For additional information about how requisites are resolved during APPLY and ACCEPT processing, see *SMP/E Commands*.

This operand has no effect on RECEIVE eligibility.

srel

specifies the system or subsystem release on which the SYSMOD can be installed. The SREL must contain four alphanumeric characters, usually one alphabetic character followed by three numeric characters. These are the systems and subsystems defined by IBM, with their SRELS:

<table>
<thead>
<tr>
<th>System</th>
<th>SREL</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB2</td>
<td>P115</td>
</tr>
</tbody>
</table>
++VER MCS

CICS C150
IMS P115
MVS Z038
NCP P004

The SREL is used during RECEIVE processing to determine whether a SYSMOD should be received. For more information about how the SREL operand is processed, see [SMP/E Commands](#).

**SUP**

indicates which SYSMODs are superseded (contained in and replaced) by this SYSMOD. For example, it may specify one or more APARs fixed in the element modifications supplied with this SYSMOD.

For functions, the same SYSMOD can be specified on both DELETE and SUP. This cleans up entries for the deleted function, and, at the same time, allows SYSMODs that name the deleted function as a requisite to still be installed.

For PTFs, APARs, and USERMODs to properly supersede another SYSMOD, all the elements in the superseded SYSMOD must be contained in either the superseding SYSMOD or a SYSMOD from the requisite set for the superseding SYSMOD (unless the element is being deleted by the superseding SYSMOD).

**VERSION**

indicates functions whose elements should be considered functionally lower than the elements contained in this SYSMOD. It specifies one or more function SYSMODs that currently contain the element. The function containing the ++VER MCS takes over ownership of all the elements from the specified functions.

When VERSION is specified on an element statement, it overrides any VERSION operand values specified on the ++VER MCS.

**Note:** A SYSMOD containing an element update (++MACUPD, ++SRCUPD, or ++ZAP) cannot change the ownership of the element. The ownership can be changed (with the ++VER VERSION operand) only if the SYSMOD provides a replacement for the element.

**Usage Notes**

- You can build a SYSMOD that can be processed by previous versions of SMP/E, as well as this version of SMP/E. For service SYSMODs, this construction requires at least two ++VER statements, one processable by previous versions of SMP/E and the other processable by this version of SMP/E. The SRELs in these ++VER statements must be different, so the SYSMOD can be processed correctly by the applicable version of SMP/E.

- A SYSMOD cannot contain multiple ++VER statements that have identical SREL and FMID values, because SMP/E would not be able to determine which ++VER MCS to use in doing the remaining applicability checking during APPLY and ACCEPT processing.

- You cannot specify the same SYSMOD more than once on a single ++VER operand. Likewise, you generally cannot specify the same SYSMOD on more than one operand. However, you can specify the same SYSMOD on VERSION and another operand (except FMID). You can also specify the same SYSMOD on the DELETE and SUP operands.

- Corequisite SYSMODs (which are related through the REQ operand) that are applicable to the same FMID cannot have elements in common. Because the REQ operand implies no service hierarchy, SMP/E cannot determine which SYSMOD has the highest service level of the common elements. When the
relationship between the SYSMODs containing the common elements is defined through the REQ operand, SMP/E issues an error or warning message.

- If the requisites have an element in common and each contains a replacement for the element, ID check processing fails and neither of the requisites is installed, unless **BYPASS(ID)** is specified.

- If the requisites have an element in common and each contains an update for the element, ID check processing issues a warning message for the requisites. Generally, the requisites are both installed. However, SMP/E does not allow multiple ZAPs for the same module to be processed by the same APPLY command. If this is the case, neither of the requisites is installed; they must be processed by separate APPLY commands.

**Examples**

The following examples are provided to help you use the ++VER MCS:

**Example 1: Defining base and dependent functions**

Assume you want to package one of your user applications as a function so you can use SMP/E in installing and maintaining it. You also have an optional enhancement to that product you want to package. The base function has two modules, USRMOD01 and USRMOD02, which are in link-edit format and reside in the library pointed to by the USRLIBXX DD statement. The optional enhancement changes USRMOD02 and depends on the base application being present. Here are examples of SYSMODs for these functions:

```clike
++FUNCTION(EUSR001) /* Base application */.
++VER(Z038) /* for MVS version. */.
++MOD(USRMOD01) /* Has this MOD */
DISTLIB(AOS12) /* in this DLIB. */
LKLIB(USRLIBXX) /* Replacement is here */.
++MOD(USRMOD02) /* Has this MOD */
DISTLIB(AOS12) /* in this DLIB. */
LKLIB(USRLIBXX) /* Replacement is here. */.
++FUNCTION(FUSR011) /* Dependent function */.
++VER(Z038) /* for MVS version. */
FMID(EUSR001) /* Base function must be present. */.
++MOD(USRMOD02) /* Has this MOD */
DISTLIB(AOS12) /* in this DLIB. */
LKLIB(USRLIBXX) /* Replacement is here. */.
++MOD(USRMOD03) /* Has this MOD */
DISTLIB(AOS12) /* in this DLIB. */
LKLIB(USRLIBXX) /* Replacement is here. */.
```

The dependent function, FUSR011, specifies the base function on its FMID operand. This means the base function must be present in order for the dependent function to be installed. The FMID operand also indicates that if the two functions have any elements in common, the version in the dependent product is the one that is to be installed. The dependent product has now assumed ownership of those elements.

**Example 2: Defining intersecting dependent functions**

Assume you want to add another dependent function for the base function defined in the preceding example. It works whether or not the other dependent function is installed. Here is an example of a SYSMOD containing a ++VER statement to define the relationship between the dependent functions:

```clike
++FUNCTION(FUSR012) /* Dependent function */.
++VER(Z038) /* for MVS version. */
FMID(EUSR001) /* Base function must be present. */.
```
Example 3: Deleting a previous level of a function

Assume you want to provide a new level of the base function defined in the first example. It includes both of the dependent functions for the original base function. Here is an example of a SYSMOD containing a ++VER statement deleting the previous level of the function:

```
++FUNCTION(EUSR002) /* New base application */.
++VER(Z038) /* for MVS version. */.
DELETE(EUSR001) /* Delete prev level. */.
++MOD(USRMOD01) /* Has this MOD */.
DISTLIB(AOS12) /* in this DLIB. */.
LKLIB(USRLIBXX) /* Replacement is here. */.
++MOD(USRMOD02) /* Has this MOD */.
DISTLIB(AOS12) /* in this DLIB. */.
LKLIB(USRLIBXX) /* Replacement is here. */.
++MOD(USRMOD03) /* Has this MOD */.
DISTLIB(AOS12) /* in this DLIB. */.
LKLIB(USRLIBXX) /* Replacement is here. */.
++MOD(USRMOD04) /* Has this MOD */.
DISTLIB(AOS12) /* in this DLIB. */.
LKLIB(USRLIBXX) /* Replacement is here. */.
++MOD(USRMOD04) /* Has this MOD */.
DISTLIB(AOS12) /* in this DLIB. */.
LKLIB(USRLIBXX) /* Replacement is here. */.
```

After this function is installed, module USRMOD02 is the version from function FUSR012, no matter what the former functional level was.

Example 3: Deleting a previous level of a function

Assume you want to provide a new level of the base function defined in the first example. It includes both of the dependent functions for the original base function. Here is an example of a SYSMOD containing a ++VER statement deleting the previous level of the function:

```
++FUNCTION(EUSR002) /* New base application */.
++VER(Z038) /* for MVS version. */.
DELETE(EUSR001) /* Delete prev level. */.
++MOD(USRMOD01) /* Has this MOD */.
DISTLIB(AOS12) /* in this DLIB. */.
LKLIB(USRLIBXX) /* Replacement is here. */.
++MOD(USRMOD02) /* Has this MOD */.
DISTLIB(AOS12) /* in this DLIB. */.
LKLIB(USRLIBXX) /* Replacement is here. */.
++MOD(USRMOD03) /* Has this MOD */.
DISTLIB(AOS12) /* in this DLIB. */.
LKLIB(USRLIBXX) /* Replacement is here. */.
++MOD(USRMOD04) /* Has this MOD */.
DISTLIB(AOS12) /* in this DLIB. */.
LKLIB(USRLIBXX) /* Replacement is here. */.
```

After this function is installed, module USRMOD02 is the version from function FUSR012, no matter what the former functional level was.

Example 3: Deleting a previous level of a function

Assume you want to provide a new level of the base function defined in the first example. It includes both of the dependent functions for the original base function. Here is an example of a SYSMOD containing a ++VER statement deleting the previous level of the function:

```
++FUNCTION(EUSR002) /* New base application */.
++VER(Z038) /* for MVS version. */.
DELETE(EUSR001) /* Delete prev level. */.
++MOD(USRMOD01) /* Has this MOD */.
DISTLIB(AOS12) /* in this DLIB. */.
LKLIB(USRLIBXX) /* Replacement is here. */.
++MOD(USRMOD02) /* Has this MOD */.
DISTLIB(AOS12) /* in this DLIB. */.
LKLIB(USRLIBXX) /* Replacement is here. */.
++MOD(USRMOD03) /* Has this MOD */.
DISTLIB(AOS12) /* in this DLIB. */.
LKLIB(USRLIBXX) /* Replacement is here. */.
++MOD(USRMOD04) /* Has this MOD */.
DISTLIB(AOS12) /* in this DLIB. */.
LKLIB(USRLIBXX) /* Replacement is here. */.
```

After the new function is installed, all references to SYSMODs EUSR001, FUSR011, and FUSR012 are deleted from the target zone and distribution zone (except the SYSMOD entry for EUSR001, which indicates it was deleted by EUSR002).

This SYSMOD does not require the previous level of the base function to be installed. The DELETE operand just says that if that previous function was installed, SMP/E should delete it before installing the new level. For more information about delete processing for the APPLY and ACCEPT commands, see [SMP/E Commands](#).

Example 4: Deleting a function without replacing it (dummy delete)

Assume you no longer need a particular function, and you want to delete it from your system. First, you must make sure that no other functions depend on the function you want to delete. Once you have done this, you need to define a dummy function SYSMOD that deletes the function you want to delete. You then receive, apply, and accept the dummy function, and run UCLIN to delete the SYSMOD entries for the deleted function and for the dummy function.

For example, assume you are ready to delete function MYFUNC1 using dummy function DELFUNC. MYFUNC1 is applicable to SREL Z038 and is installed in target zone TGT1 and distribution zone DLIB1. Here is an example of the dummy function:

```
++FUNCTION(DELFUNC) /* Any valid unique SYSMOD ID. */.
++VER(Z038) /* For SREL Z038 (MVS products). */.
DELETE(MYFUNC1) /* Deletes MYFUNC1. */.
```

After the new function is installed, all references to SYSMODs EUSR001, FUSR011, and FUSR012 are deleted from the target zone and distribution zone (except the SYSMOD entry for EUSR001, which indicates it was deleted by EUSR002).

This SYSMOD does not require the previous level of the base function to be installed. The DELETE operand just says that if that previous function was installed, SMP/E should delete it before installing the new level. For more information about delete processing for the APPLY and ACCEPT commands, see [SMP/E Commands](#).

Example 4: Deleting a function without replacing it (dummy delete)

Assume you no longer need a particular function, and you want to delete it from your system. First, you must make sure that no other functions depend on the function you want to delete. Once you have done this, you need to define a dummy function SYSMOD that deletes the function you want to delete. You then receive, apply, and accept the dummy function, and run UCLIN to delete the SYSMOD entries for the deleted function and for the dummy function.

For example, assume you are ready to delete function MYFUNC1 using dummy function DELFUNC. MYFUNC1 is applicable to SREL Z038 and is installed in target zone TGT1 and distribution zone DLIB1. Here is an example of the dummy function:

```
++FUNCTION(DELFUNC) /* Any valid unique SYSMOD ID. */.
++VER(Z038) /* For SREL Z038 (MVS products). */.
DELETE(MYFUNC1) /* Deletes MYFUNC1. */.
```

After the new function is installed, all references to SYSMODs EUSR001, FUSR011, and FUSR012 are deleted from the target zone and distribution zone (except the SYSMOD entry for EUSR001, which indicates it was deleted by EUSR002).

This SYSMOD does not require the previous level of the base function to be installed. The DELETE operand just says that if that previous function was installed, SMP/E should delete it before installing the new level. For more information about delete processing for the APPLY and ACCEPT commands, see [SMP/E Commands](#).

Example 4: Deleting a function without replacing it (dummy delete)

Assume you no longer need a particular function, and you want to delete it from your system. First, you must make sure that no other functions depend on the function you want to delete. Once you have done this, you need to define a dummy function SYSMOD that deletes the function you want to delete. You then receive, apply, and accept the dummy function, and run UCLIN to delete the SYSMOD entries for the deleted function and for the dummy function.

For example, assume you are ready to delete function MYFUNC1 using dummy function DELFUNC. MYFUNC1 is applicable to SREL Z038 and is installed in target zone TGT1 and distribution zone DLIB1. Here is an example of the dummy function:

```
++FUNCTION(DELFUNC) /* Any valid unique SYSMOD ID. */.
++VER(Z038) /* For SREL Z038 (MVS products). */.
DELETE(MYFUNC1) /* Deletes MYFUNC1. */.
```
These are the commands you use to receive and install the dummy function, and to delete the SYSMOD entries for the dummy function and the deleted function:

```
SET BDY(GLOBAL) /* Set to global zone. */.
RECEIVE S(DELFUNC) /* Receive the function. */.
SET BDY(TGT1) /* Set to applicable target. */.
APPLY S(DELFUNC) /* Apply to delete old */
     /* function. */.
SET BDY(DLIB1) /* Set to applicable DLIB. */.
ACCEPT S(DELFUNC) /* Accept to delete old */
     /* function. */.
SET BDY(TGT1) /* Set to applicable target. */.
UCLIN.
DEL SYSMOD(DELFUNC) /* Delete SYSMOD entries for */.
DEL SYSMOD(MYFUNC1) /* dummy and old function. */.
ENDUCL.
SET BDY(DLIB1) /* Set to applicable DLIB. */.
UCLIN.
DEL SYSMOD(DELFUNC) /* Delete SYSMOD entries for */.
DEL SYSMOD(MYFUNC1) /* dummy and old function. */.
ENDUCL /* */.
```

When you accept the dummy function, SMP/E automatically deletes the associated SYSMOD entry from the global zone and the MCS entry from the SMPPTS.

To complete the cleanup, you should also use the REJECT command to delete any SYSMODs and HOLDDATA applicable to the dummy function and the old function. In addition, you should delete the FMIDs from the GLOBALZONE entry to prevent SMP/E from receiving any SYSMODs or HOLDDATA applicable to either of those functions. Here are examples of the commands you can use to do this.

```
SET BDY(GLOBAL) /* Set to global zone. */.
REJECT HOLDDATA NOFMID /* Reject SYSMODs, HOLDDATA */
     /* PRODUCT information */.
     /* for the deleted functions. */
     /* Delete the FMIDs from the */
     /* GLOBALZONE entry. */.
```

**Note:** If you delete a function that used totally copied libraries, there will be DLIB entries left in the zone after the deletion, which may cause problems if this function is later reinstalled. You should run UCLIN to delete the leftover DLIB entries to ensure that SMP/E will create new DLIB entries that point to the proper libraries when a new copy of the function is installed.
++ZAP MCS

The ++ZAP MCS describes a module update within a PTF, APAR fix, or USERMOD. It must precede the IMASPZAP statements within the SYSMOD.

Syntax

```
++ZAP

++ZAP(name)

DALIAS(alias)

DISTLIB(ddname)

TALIAS(alias)
```

Operands

**DALIAS**

is the alias name of a module that has an alias in the distribution library, but not in the target library. This might be used if the module is included under its alias name during system generation.

Note: DALIAS is mutually exclusive with TALIAS.

**DISTLIB**

specifies the ddname of the distribution library for the specified module.

Note: This operand must be specified if the module has not been previously recorded on the target zone or distribution zone. If a MOD entry already exists in the target zone or distribution zone and the value currently in that entry does not match that specified in the DISTLIB operand, the SYSMOD is not applied or accepted.

**name**

specifies the name of the module member in the distribution library and, optionally, in the target system library. The name can contain any alphanumeric characters and $, #, @, or hex C0.

Note: If the module to be updated has been assembled, specify the name of the assembled module, not the CSECT name.

**TALIAS**

identifies all the alias names of a module that has aliases in both the target and distribution libraries.

You can use TALIAS for a module that was copied from a distribution library into a target library (defined by JCLIN data as a copied module), but not for one that is link-edited (defined by JCLIN data as a link-edited module).

TALIAS must be specified on the ++MOD MCS even if ALIAS was specified on the COPY SELECT statement.

Note: TALIAS is mutually exclusive with DALIAS.
Usage Notes

- If a SYSMOD containing a ++ZAP statement attempts to change the ownership (FMID) of the element (with the VERSION operand), the SYSMOD cannot be installed.
- The changes for the module must immediately follow the ++ZAP MCS and must not contain any records that start with ++.
- The only IMASPZAP control statements allowed in a SYSMOD are:

```
ABSDUMP   DUMP     NAME     VER
ABSDUMPT  DUMPT    REP      VERIFY
BASE      IDRDATA  SETSSI  *(comment)
```

- An EXPAND control statement in link-edit utility format can be placed within IMASPZAP input to allow lengthening of control sections. The EXPAND statement must follow the NAME statement. For the syntax and description of the EXPAND statement, see [z/OS MVS Program Management: User’s Guide and Reference, SA22-7643](http://www.ibm.com). All control statements must begin in or after column 2.
- Expand type IMASPZAP processing cannot be performed against a noneditable (NE) module.
- Any SETSSI statements placed in the input stream for expand-type IMASPZAP processing must be in a form acceptable to both IMASPZAP and the link-edit utility; that is, they must begin in column 2 or after. The SSI statements must follow the EXPAND statements.
- The name specified on the ++ZAP MCS must be the same as the name of the distribution library module. The CSECT value on the IMASPZAP NAME statement must be the same as the load module's CSECT name. That CSECT name is usually the same as the distribution library name, but it can be different. For example, if the module to be updated has been assembled, the ++ZAP statement should specify the name of the assembled module, not the CSECT name.

The LIST LMOD statement produces a target zone listing of link-edit utility control statements that might have changed the CSECT name of the member. A link-edit map may be helpful in other cases where the names differ.
- The IMASPZAP NAME statement can optionally be coded as follows:

```
NAME csect-name
or
NAME lmod-name csect-name
or
NAME lmod-name csect-name class-name
```

The coding of one operand assumes that operand to be a CSECT name for the module referred to in the ++ZAP statement. In this case, all load modules containing the module named in the ++ZAP statement are processed by IMASPZAP.

Two operands can be specified, in which case the second operand is assumed to be a CSECT name. The first operand is assumed to be a valid load module containing the module named in the ++ZAP statement. In this case, only the indicated load module is processed by IMASPZAP.

Three operands can also be specified, in which case the first two operands are as described in the previous paragraphs. The third operand indicates, for program objects only, the class of text that you want to modify. The default is B_text.
you want to omit the CSECT name and supply a class-name, code a single asterisk for the CSECT name, followed by the class-name.

- When using IMASPZAP on an assembled module, be careful: The modification identifier is updated, but not the modification of any associated macros. It is not recommended that you use IMASPZAP to modify assembled modules. An assembled module modified by IMASPZAP does not cause updating of the distribution library during accept processing; therefore, a subsequently generated system does not contain the IMASPZAP modification.

A more satisfactory method of updating assembled modules is to update the macros that generate them.

- SMP/E processing does not save a backup copy of the nucleus during apply processing when the nucleus is modified by a SYSMOD containing a non-expand-type IMASPZAP modification.

- Only one ZAP can be applied to a module by a single APPLY command. If you need to install several ZAPs for a given module, each one must be packaged separately and installed by a separate APPLY command.

- If you need to install a ++ZAP change on a live system library and that library uses library lookaside (LLA), virtual lookaside facility (VLF), or both, you must do some setup work before installing the change. Otherwise, the change might not take effect, even after installation. See z/OS MVS Initialization and Tuning Guide for guidance on:
  - Removing libraries from LLA and VLF control
  - Refreshing LLA and VLF

Note: Installing a change on a live system is not recommended.

Examples

The examples in this section are based on the load module structure shown in Figure 2.

<table>
<thead>
<tr>
<th>Load Module Name</th>
<th>Load Module Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module Name</td>
<td>Module Name</td>
</tr>
<tr>
<td>CSECT Name</td>
<td>CSECT Name</td>
</tr>
<tr>
<td>LMODA</td>
<td>LMODB</td>
</tr>
<tr>
<td>MOD1</td>
<td>MOD1</td>
</tr>
<tr>
<td>CSECT1</td>
<td>CSECT1</td>
</tr>
<tr>
<td>CSECT2</td>
<td>CSECT2</td>
</tr>
<tr>
<td>CSECT3</td>
<td>CSECT3</td>
</tr>
<tr>
<td>MOD02</td>
<td>MOD02</td>
</tr>
<tr>
<td>MOD03</td>
<td>MOD03</td>
</tr>
<tr>
<td>MOD04</td>
<td>MOD04</td>
</tr>
</tbody>
</table>

Figure 2. Load module structure for ++ZAP examples

The following examples show how ++ZAP statements can be used to update modules, load modules, and CSECTs within modules:
Example 1: Changing all load modules that contain the same module
Assume you want to change CSECT2 in module MOD1, which is in both LMODA and LMODB. Here is an example of a SYSMOD that accomplishes this by specifying the CSECT name on the NAME statement without including any load module names:

```
++USERMOD(MYMOD01).
++VER(Z038) FMID(FXY1000).
++ZAP(MOD1).
  NAME CSECT2
  VER 0000 FF4160
  REP 0000 FE4160
```

Example 2: Changing the only load module that contains a given module
Assume you want to change CSECT MOD3 in module MOD3, which is only in LMODA. Here is an example of a SYSMOD that accomplishes this by specifying the CSECT name on the NAME statement without including any load module names:

```
++USERMOD(MYMOD02).
++VER(Z038) FMID(FXY1000).
++ZAP(MOD3).
  NAME MOD3
  VER 000A 00
  REP 000A FF
```

Example 3: Changing one of several load modules that contain a given module
Assume you want to change CSECT2 in module MOD1, which is in both LMODA and LMODB. You want to update only the version in LMODB. Here is an example of a SYSMOD that accomplishes this by specifying both the load module name and the CSECT name on the NAME statement.

```
++USERMOD(MYMOD03).
++VER(Z038) FMID(FXY1000).
++ZAP(MOD1).
  NAME LMODB CSECT2
  VER 0000 00
  REP 0000 FF
```

Example 4: Expanding a module
Assume you want to update CSECT3 with an EXPAND request. CSECT3 is in module MOD1, which is in both LMODA and LMODB. Here is an example of a SYSMOD that accomplishes this by specifying the CSECT name on the NAME statement and on an EXPAND statement.

```
++USERMOD(MYMOD04).
++VER(Z038) FMID(FXY1000).
++ZAP(MOD1).
  NAME CSECT3
  VER 0000 FF
  REP 0000 FE
  EXPAND CSECT3(4)
```
++ZAP MCS
Chapter 3. Defining control statements in SMPPARM members

The SMPPARM data set can contain members that allow you to customize SMP/E as follows:

<table>
<thead>
<tr>
<th>Member name</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIMDDALC</td>
<td>Defines data sets to be dynamically allocated</td>
</tr>
<tr>
<td>GIMEXIT</td>
<td>Defines exit routines</td>
</tr>
<tr>
<td>GIMOPCDE</td>
<td>Defines macro and assembler operation codes</td>
</tr>
</tbody>
</table>

**GIMDDALC control statements**

SMPPARM member GIMDDALC is used to specify data sets that are to be dynamically allocated by SMP/E. The following types of data sets may be specified in GIMDDALC:

- Data sets to be allocated to a SYSOUT class (or to the terminal for foreground execution)
- Data sets to be allocated as temporary data sets
- SMPTLIB data sets

During any attempt to allocate a data set, SMP/E first looks for a DD statement specified in the job. If no DD statement was specified, SMP/E then looks for a DDDEF entry in the zone. If no DDDEF entry was found, SMP/E then checks to see if there is an SMPPARM data set. If so, and there is a GIMDDALC member within it, SMP/E looks for a GIMDDALC control statement for the corresponding ddname.

**Syntax**

**SYSOUT data sets**

```
DD(ddname) SYSOUT(class, TERM)
```

**SMPWRKKn and SYSUTn data sets**

```
DD(ddname) BLOCK(size) CYLINDERS TRACKS
   UNIT(type) VOLUME(volid) DATACLASS(name) MGMTCLASS(name)
   STORCLASS(name) DSTYPE(LIBRARY)
```
GIMDDALC statements

SMPTLIB data sets

<table>
<thead>
<tr>
<th>DD(SMPTLIB)</th>
<th>TRACKS</th>
<th>DIR(nn)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPACE(primary,secondary)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Operands**

**DD** identifies the ddname of the data set to be allocated.
- *ddname* can be from 1 to 8 alphanumeric (A through Z, and 0 through 9), or national characters (@,#, or $), and must start with an alphabetic or national character.
- The DD operand must be the first operand specified on a GIMDDALC control statement.

**BLOCK(size), CYLINDERS, or TRACKS**

specifies the space units to be used in allocating the data set: blocks, cylinders, or tracks. *size* is the size of each block to be allocated, and can be from 1 to 5 decimal digits. To specify the number of these units, use the SPACE operand.

**BLOCK** can be specified as BLK, **CYLINDERS** can be specified as CYL, and **TRACKS** can be specified as TRK.

**DATACLAS**

specifies the name of a data class to be used for allocating a data set managed by SMS.

The data class can be from 1 to 8 alphanumeric characters (A through Z and 0 through 9) or national characters (@, #, $).

**DIR**

specifies the number of directory blocks to allocate.

The number can contain from 1 to 4 decimal digits.

**DSNTYPE**

specifies the type of partitioned data set to be created.
- **LIBRARY** indicates a PDSE is to be created.
- **PDS** indicates a PDS is to be created.

**MGMTCLAS**

specifies the name of a management class to be used for allocating a data set managed by SMS.

The management class can be from 1 to 8 alphanumeric characters (A through Z and 0 through 9) or national characters (@, #, $).

**SPACE**

specifies the primary and secondary space allocation for a data set. To specify the units for the space allocation, use one of the **BLOCK, CYLINDERS, or TRACKS** operands.
- The *primary* and *secondary* values can contain from 1 to 4 decimal digits, and the two values must be separated by a comma or a blank.
- The units of *primary* and *secondary* space allocation for SMPTLIB data sets is always tracks.
STORCLAS
specifies the name of a storage class to be used for allocating a data set
managed by SMS.

The storage class can be from 1 to 8 alphanumeric characters (A through Z and
0 through 9) or national characters (@, #, $).

SYSOUT
specifies the output class for SYSOUT data sets.
• The class value must be 1 alphabetic or numeric character (A through Z and
0 through 9).
• An asterisk (*) indicates the class from the OUTPUT DD statement is to be
used. If no OUTPUT DD statement is found, the system default message
class is used.
• TERM indicates if SMP/E is executing in the foreground, the data set is to
be allocated to the terminal.
• SYSOUT is mutually exclusive with all other operands except DD.
• You can not specify SYSOUT for SMPTLIB data sets.

UNIT
specifies the UNIT type the data set is to reside on.
UNIT can be any nonblank 1 to 8 character string.

VOLUME
specifies the volume serial number of the volume the data set is to reside on.

The volume serial number can be from 1 to 6 alphanumeric characters (A
through Z and 0 through 9).

Syntax notes

1. GIMDDALC control statements must each start on a new line.
2. GIMDDALC control statements may be continued on more than one line.
   SMP/E assumes a statement is continued if it did not find a period (.) before
column 73.
3. SMP/E ignores columns 73 through 80. If data, such as a period, is specified
   beyond column 72, SMP/E will ignore it and indicate an error in the control
   statement following the one containing the data.
4. GIMDDALC control statements may contain comments. Comments start with /*
   and end with */. The first */ encountered after the initial /* will end the
   comment. A comment can appear anywhere within a statement. A comment
   after the ending period must start on the same line as the period, and you
   cannot start any additional comments after that final comment. For example,
you can code a comment like this:
   DD(SMPTLIB) SPACE(5,5) DIR(20). /* Comment after period continued
       on subsequent lines is OK. */

   However, you cannot code a comment like this:
   DD(SMPTLIB) SPACE(5,5) DIR(20). /* Comment after period OK,
       */
   /* but starting another comment
       causes a syntax error. */

   This causes a syntax error at the start of the second comment after the period.
5. Only one GIMDDALC control statement is allowed for a single data set.
6. At least one operand (other than DD) must be specified on each GIMDDALC
   control statement.
7. For SMPTLIB, only the **SPACE**, **TRACKS**, and **DIR** operands are allowed.

**Sample GIMDDALC member**

The following is a sample GIMDDALC member. The member is named GIMDDALC and is shipped with SMP/E in the SAMPLIB target library.

```c
/**
 * **************************************************************************
 * * Licensed Materials - Property of IBM                                   *
 * * 5694-A01 5655-G44                                                    *
 * * (C) Copyright IBM Corp. 1999, 2001                                   *
 * * **************************************************************************
 * GIMDDALC -- Sample GIMDDALC member for the SMPPARM data set            *
 * **************************************************************************

This sample member is a model to assist the SMP/E user in constructing a GIMDDALC member specifically for their use. A user should copy this sample into the GIMDDALC member of a data set. That data set is then defined to SMP/E using the SMPPARM ddname either using a DD statement in the SMP/E job, or using a DDDEF entry.

The GIMDDALC member contains information used to dynamically allocate three kinds of data sets:

1. Data sets to be allocated to a SYSOUT class (or to the terminal for foreground execution),
2. Data sets to be allocated as temporary data sets, and
3. SMPTLIB data sets.

During any attempt to allocate a ddname, SMP/E will first look for a DD statement specified in the job. If no DD statement was specified, SMP/E will look for a DDDEF entry in the zone. If no DDDEF entry was found, SMP/E will then go to member GIMDDALC in the SMPPARM data set. If there is an SMPPARM data set, and there is a GIMDDALC member within it, SMP/E will look for an appropriate control statement.

**NOTE:** The allocation values in the control statements of this sample member are not intended to be recommendations for the actual values you should use and will not be appropriate for all users and all environments.

See the "SMP/E Reference" for details on the syntax of the control statements and on the use of member GIMDDALC.
```

Figure 3. Sample GIMDDALC member provided in SAMPLIB (Part 1 of 2)
GIMEXITS control statements

SMP/E provides support for two exit points in SMP/E command processing. You can write exit routine programs that receive control at those exit points to:

- process statements from SMPPTFIN during the RECEIVE command, and
- control retry processing when data sets run out of space during ACCEPT, APPLY, GZONEMERGE, LINK LMODS, LINK MODULE, RECEIVE, or RESTORE processing.
GIMEXITS Statements

The GIMEXITS member of the SMPPARM data set contains control statements used to identify the exit routine programs to be given control at those SMP/E exit points. It is the presence of the control statement that indicates to SMP/E that an exit routine program is to be called at the appropriate exit point during SMP/E processing. The statements not only identify the exit routine program, but also allow you to identify the data set that contains the exit routine program. This eliminates the need to copy the exit routines into the new MIGLIB or LINKLIB data sets; the exit routines can reside in almost any authorized data set you desire.

Syntax

```
GIMEXITS control statement

EXIT(RECEIVE MODNAME(name) DATASET(dataset)).
```

Operands

EXIT
identifies the SMP/E exit point that will pass control to the exit routine defined on the MODNAME operand of this control statement. Valid values are:

RECEIVE indicates the RECEIVE command exit point will pass control to the exit routine.

RETRY indicates the RETRY exit point will pass control to the exit routine.

The EXIT operand must be the first operand specified on a GIMEXITS control statement.

MODNAME
specifies the name of the exit routine to receive control during SMP/E execution at the identified exit point. The exit routine must be a load module residing in an authorized library. If the DATASET operand is not specified, SMP/E uses the system search order for programs to locate the exit routine. See the topic “The Search Order the System uses for Programs” in z/OS MVS Initialization and Tuning Guide for more information.

The name can be from 1 to 8 alphanumeric characters (A through Z and 0 through 9) or national characters (@, #, $). MODNAME is a required operand.

DATASET
specifies the data set name of a load library in which the exit routine resides. The data set must be cataloged and it must be an authorized library. If the DATASET operand is not specified, SMP/E uses the system search order for programs to locate the exit routine. See the topic “The Search Order the System uses for Programs” in z/OS MVS Initialization and Tuning Guide for more information.

The data set name must conform to standard naming conventions for data sets. Each part of the name must contain from 1 to 8 alphanumeric characters (A through Z and 0 through 9) or dash (-), separated from the other parts by a period (.), with no intervening blanks. The first character of each part must not be numeric or dash, and the maximum length of the entire name is 44 characters (including the periods).
GIMEXITS Statements

Syntax notes

1. GIMEXITS control statements must each start on a new line.
2. GIMEXITS control statements may be continued on more than one line. SMP/E assumes a statement is continued if it did not find a period (.) before column 73.
3. SMP/E ignores columns 73 through 80. If data, such as a period, is specified beyond column 72, SMP/E will ignore it and indicate an error in the control statement following the one containing the data.
4. GIMEXITS control statements may contain comments. Comments start with /* and end with */. The first */ encountered after the initial /* will end the comment. A comment can appear anywhere within a statement. A comment after the ending period must start on the same line as the period, and you cannot start any additional comments after that final comment. For example, you can code a comment like this:

   EXIT(RECEIVE) MODNAME(MYRECEX). /* Comment after period continued on subsequent lines is OK. */

   However, you cannot code a comment like this:

   EXIT(RECEIVE) MODNAME(MYRECEX). /* Comment after period OK, */ /* but starting another comment causes a syntax error. */

   This causes a syntax error at the start of the second comment after the period.
5. Only one GIMEXITS control statement is allowed for a single exit point.

Sample GIMEXITS member

The following is a sample GIMEXITS member. The member is named GIMEXITS and is shipped with SMP/E in the SAMPLIB target library.
GIMEXITS Statements

This sample member is a model to assist the SMP/E user in constructing
a GIMEXITS member specifically for their use. A user should copy this
sample into the GIMEXITS member of a data set. That data set is then
defined to SMP/E using the SMPPARM ddname either using a DD statement
in the SMP/E job, or using a DDDEF entry.

The GIMEXITS member contains information used to identify the exit
routines which are to get control during SMP/E processing at specific
exit points:

RECEIVE The RECEIVE exit point allows you to scan statements in the
SMPPTFIN data set during RECEIVE command execution.

RETRY The RETRY exit point enables you to control SMP/E's RETRY
processing when a data set runs out of space during APPLY,
ACCEPT, RESTORE, and LINK command execution.

See the "SMP/E Reference for details on the syntax of the control
statements and on the use of member GIMEXITS.

CHANGE_ACTIVITY

CHANGE_FLAG KEY: M@PNC

WHERE:

M= MULTIPLE P= DCRS = D,E,F,G N= 0-9,A-Z C= A = ADD
FOR DELETE RMPS = H,I,J,K (USE '0' ONLY C = CHANGE
FLAGS ONLY LINE = L,M,N,O WHEN ELEMENT P = COPY
PTMS = P,Q,R,S IS BEING M = MOVE
APARS = 0-9 CREATED) D = DELETE

EXIT(RECEIVE) MODNAME(MYRECEX) DATASET(SMPE.EXITS.LOAD). /*
Define exit routine MYRECEX in data set SMPE.EXITS.LOAD
to get control during SMP/E RECEIVE command processing.
*/
EXIT(RETRY) MODNAME(MYRTYEX) DATASET(SMPE.EXITS.LOAD). /*
Define exit routine MYRTYEX in data set SMPE.EXITS.LOAD
to get control during SMP/E RETRY processing.
*/
OPCODE control statements

During JCLIN processing, SMP/E scans the SMPJCLIN input, which consists of various job steps calling system utilities. To determine the structure of the target system, SMP/E specifically looks at copy steps, link-edit steps, and assembly steps.

As SMP/E scans inline assembly steps, it looks at each assembler instruction to determine whether the instruction is a macro invocation or an OPCODE. If SMP/E determines that this is a macro invocation, it builds a MAC entry in the target zone and defines the connection between the macro and the assembly in which the macro was found. As a result, when that macro is later changed by the installation of a SYSMOD, SMP/E can cause all the assemblies that used that macro to be redone.

SMP/E uses GIMOPCDE members to determine whether an assembler instruction is a macro invocation or an OPCODE.

SMP/E provides you with a default set of OPCODE definitions that identifies all the known standard assembler OPCODEs. You can define your own members to identify additional OPCODE values or macro names by using the sample GIMOPCDE member SMP/E supplied for you. If you define your own member, you must allocate the SMPPARM data set. You can specify the name of one of these user-defined GIMOPCDE members on the JCLIN command or on the ++JCLIN statement to have SMP/E pick up this additional information. The data from this other GIMOPCDE member is merged with the data from the default set of OPCODE definitions. If duplicate data is specified, the user-specified member has priority.

SMP/E uses the following method to determine whether an assembler instruction is an OPCODE or a macro:

- SMP/E looks for a user-allocated SMPPARM data set.
- If SMPPARM is not found, SMP/E uses the default set of OPCODE definitions.
  
  If SMPPARM is found and there is a user-defined GIMOPCDE member specified on the JCLIN statement or in the ++JCLIN MCS, then SMP/E searches for the specified member in SMPPARM. If it finds that member, it will look first in that member for a definition of the character string.
  
  - If the user-defined GIMOPCDE member specified is not found, SMP/E searches SMPPARM for the GIMOPCDE member. If it finds the GIMOPCDE member, it will look only in that member for a definition of the character string. (The GIMOPCDE member, if it exists in SMPPARM, will completely override SMP/E’s default set of OPCODE definitions.)
  
  - If the GIMOPCDE member is not found, SMP/E uses the default set of OPCODE definitions.
  
  - If the character string is not defined in either the SMPPARM data set or the default set, SMP/E considers it to be a macro.

For additional information about how SMP/E processes assembler input as JCLIN, see “++JCLIN MCS” on page 60.
OPCODE statements

Syntax

```
GIMOPCDE member control statements

KEY=name
    TYPE=OPCODE
    TYPE=MACRO
```

Operands

**KEY**

identifies a character string for SMP/E to check for when scanning assembler entries during JCLIN processing, either during the JCLIN command or when applying a SYSMOD with inline JCLIN.

*name* can be any alphanumeric character string from one to eight characters in length.

**TYPE**

specifies how SMP/E treats that character string when it is encountered. The following values may be specified:

- **OPCODE**
  - specifies that SMP/E treats the specified character string as a valid assembler OPCODE.
  
  **Note:** OPCODE can also be specified as OP.

- **MACRO**
  - indicates that SMP/E treats the specified character string as the name of a macro.
  
  **Note:** MACRO can also be specified as MAC.

**Notes:**

1. The TYPE operand can be specified only once per control statement.
2. If the TYPE operand is not specified, the default is TYPE=OPCODE.

Usage notes

- Either a blank or a comma can be used to separate the operands.
- Comments are permitted, subject to the next restriction.
- An OPCODE control statement cannot span multiple lines. The complete control statement, including comments, must be contained on a single line.
- If, during JCLIN processing, SMP/E encounters a string specified in one of the OPCODE control statements, and the OPCODE control statement specifies that the string is to be treated as a macro, SMP/E builds a MAC entry in the target zone and connects that MAC entry to the ASSEM entry built for the current assembly. For a further description of JCLIN processing as related to the GIMOPCDE members, see [SMP/E Commands](#).

Examples

The following examples are provided to help you use OPCODE statements.
**Example 1: Defining a new OPCODE for special assemblers**

Some licensed programs use a version of the assembler supplied with their product rather than the standard assembler. Often, this assembler recognizes special OPCODEs. When SMP/E is processing JCLIN containing assembler steps that use these OPCODEs, you do not want SMP/E creating target zone macro entries for them. To avoid this situation, create a special GIMOPCDE member containing appropriate control statements to define those OPCODEs to SMP/E.

The following is an example of a GIMOPCDE member containing three control statements to define the three-character strings, *OP1*, *OPCD1*, and *OPCODE01* as special assembler OPCODEs:

```
KEY=OP1    TYPE=OPCODE.
KEY=OPCD1   TYPE=OP.
KEY=OPCODE01 TYPE=OPCODE.
```

**Example 2: Overriding an SMP/E-defined OPCODE**

You may provide your own GIMOPCDE member to override SMP/E’s default set of OPCODE definitions by using the sample GIMOPCDE member provided for you. This sample member contains a control statement identifying *PUNCH* as a character string that should be treated as an assembler OPCODE. You may have constructed a *PUNCH* macro and now want SMP/E to recognize *PUNCH* as a macro name. This can be done by adding the following control statement to an GIMOPCDE member and then specifying that member on the JCLIN command or +JCLIN statement:

```
KEY=PUNCH    TYPE=MACRO    /* override type to macro */.
```
OPCODE statements
Chapter 4. SMP/E data sets and files

This chapter describes the data sets that are used to process SMP/E commands. These data sets are listed by the ddnames for the data sets required by each SMP/E command. According to the type of data set, you can define it with a DDDEF entry, a DD statement, or with GIMDDALC control statements in SMPPARM member GIMDDALC. For more information about DDDEF entries, see "DDDEF entry (distribution, target, and global zone)” on page 197. For more information about SMPPARM member GIMDDALC, see Chapter 3, “Defining control statements in SMPPARM members,” on page 129. For more information about dynamic allocation, see SMP/E User’s Guide.

CLIENT

ddname
Specified by the user.

Use
The CLIENT data set contains information about the local z/OS system such as how to navigate a local FTP firewall and HTTP proxy server, in addition to the location of Java application classes and Java debug options. This information is used by the RECEIVE FROMNETWORK and RECEIVE ORDER commands, as well as the GIMGTPKG service routine. The contents of the CLIENT data set are described in the RECEIVE chapter in SMP/E Commands, SA22-7771.

Attributes
Sequential or member of a partitioned data set; LRECL=80, RECFM=F or FB.

Device
Direct access.

Note: This data set may reside in a UNIX file system. Specify PATHOPTS(ORDONLY) and either FILEDATA=TEXT or FILEDATA=BINARY on the DD statement if it is to reside in a UNIX file system. For simplicity, FILEDATA=TEXT is preferred, and ensure the newline character (X'15') marks the end of each line. If you use FILEDATA=BINARY, you must ensure that each line is padded with blanks to the 80-byte record length because there is no end of record marker in binary data.

Distribution library (DLIB)

ddname
The ddname for a distribution library should match the low-level qualifier of the data set name. For example, the ddname for SYS1.AMACLIB should be AMACLIP.

Use
Distribution libraries (DLIBs) contain updated versions of macros, source, and modules that were shipped by IBM and stored during ACCEPT processing. They are used during system generation to build the target libraries, so you should keep them at a tested functional and service level. They are also used during RESTORE processing to replace elements in the target libraries. You must provide a DDDEF entry or DD statement for each distribution library that is being processed.

Attributes
Partitioned.
**Distribution library (DLIB)**

| Device | Direct access only. |

**INFILE**

**ddname** The ddname for an INFILE data set cannot be the same as the ddname for any other data sets used by SMP/E. Other than this, there are no restrictions.

**Use** The INFILE data set is a sequential data set containing a zone that is to be loaded by the ZONEIMPORT command. The ddname of this sequential data set is specified on the INFILE operand of the ZONEIMPORT command. The zone in the INFILE data set can be used in two ways:

- To recreate a zone that was destroyed
- To recreate a zone on another CSI data set

You must use the ZONEEXPORT command to create the INFILE data set. (This same data set is called the OUTFILE data set when it is created by the ZONEEXPORT command.)

**Attributes** Sequential; no DCB parameters are required.

If the INFILE data set is on tape, there may be more than one volume for an exported data set. Remember to specify all the volume serial numbers on the INFILE DD statement.

**Device** Tape or direct access.

**Note:** BLKSIZE must not exceed 32760.

**Link library (LKLIB)**

**ddname** The ddname for a link library must match the LKLIB value on the element MCS. For example, the ddname for the link library on statement `++MOD(MODA) LKLIB(LIBX)` must be LIBX.

**Use** Link libraries contain replacements for object modules in link-edited format. They are used when the modules are provided in partitioned data sets rather than inline or in relative files.

**Attributes** Partitioned.

**Device** Direct access only.

**ORDERSERVER**

**ddname** Specified by the user.

**Use** The ORDERSERVER data set contains information about the IBM Automated Delivery Request server used by the RECEIVE ORDER command. The contents of the ORDERSERVER are described in the RECEIVE chapter in [SMP/E Commands, SA22-7771](#).

**Attributes** Sequential or member of a partitioned data set; LRECL=80, RECFM=F or FB.

**Device** Direct access.

**Note:** This data set may reside in a UNIX file system. Specify `PATHOPTS(ORDONLY)` and either `FILEDATA=TEXT` or `FILEDATA=BINARY` on the DD statement if it is to reside in a UNIX file.
### OUTFILE

**ddname**

The ddname for an OUTFILE data set cannot be the same as the ddname for any other data sets used by SMP/E. Other than this, there are no restrictions.

**Use**

The OUTFILE data set is a sequential data set containing a zone copied by the ZONEEXPORT command. The ddname of this sequential data set is specified on the OUTFILE operand. The zone in the OUTFILE data set can be used in two ways:

- As a backup copy, to recreate a zone that was destroyed
- As a transportable copy, to recreate a zone on another CSI data set

You must use the ZONEIMPORT command to process the OUTFILE data set. (This same data set is called the INFILE data set when it is processed by the ZONEIMPORT command.)

**Attributes**

Sequential; no DCB parameters are required.

**Device**

Tape or direct access.

**Note:** BLKSIZE must not exceed 32760.

### SERVER

**ddname**

Specified by the user.

**Use**

The SERVER data set contains information about a TCP/IP connected host running an FTP server.

**Attributes**

Sequential or member of a partitioned data set; LRECL=80, RECFM=F or FB.

**Device**

Direct access.

**Note:** This data set may reside in a UNIX file system. Specify PATHOPTS(ORDONLY) and either FILEDATA=TEXT or FILEDATA=BINARY on the DD statement if it is to reside in a UNIX file system. For simplicity, FILEDATA=TEXT is preferred, and ensure the newline character (X'15') marks the end of each line. If you use FILEDATA=BINARY, you must ensure that each line is padded with blanks to the 80-byte record length because there is no end of record marker in binary data.

### SMPCLNT

**ddname**

SMPCLNT

**Use**

The SMPCLNT data set contains information about the local z/OS system such as how to navigate a local FTP firewall, the location of Java application classes and Java debug options. This information is used by the GIMGTPKG service routine, and has the same
**SMPCLNT**

Contents as the CLIENT data set. The contents of the CLIENT data set are described in the RECEIVE chapter in [SMP/E Commands](#). Attributes

Sequential or member of a partitioned data set; LRECL=80, RECFM=F or FB.

Device  

Direct access.

**Note:** This data set may reside in a UNIX file system. Specify

**PATHOPTS(ORDONLY)** and either **FILEDATA=TEXT** or **FILEDATA=BINARY** on the DD statement if it is to reside in a UNIX file system. For simplicity, **FILEDATA=TEXT** is preferred, and ensure the newline character (X'15') marks the end of each line. If you use **FILEDATA=BINARY**, you must ensure that each line is padded with blanks to the 80-byte record length because there is no end of record marker in binary data.

**SMPCNTL**

<table>
<thead>
<tr>
<th>ddbname</th>
<th>SMPCNTL</th>
</tr>
</thead>
</table>

**Use**  
The SMPCNTL data set contains the SMP/E commands to be processed.

**Attributes**  
Sequential; LRECL=80, BLKSIZE=multiple of 80, RECFM=FB.

**Device**  
Card, tape, direct access, or terminal.

**Notes:**

1. BLKSIZE must not exceed 32760.
2. This data set may reside in a UNIX file system. Specify **PATHOPTS(ORDONLY)** and either **FILEDATA=TEXT** or **FILEDATA=BINARY** on the DD statement if it is to reside in a UNIX file system. For simplicity, **FILEDATA=TEXT** is preferred, and ensure the newline character (X'15') marks the end of each line. If you use **FILEDATA=BINARY**, you must ensure that each line is padded with blanks to the 80-byte record length because there is no end of record marker in binary data.

**SMPCPATH**

<table>
<thead>
<tr>
<th>ddbname</th>
<th>SMPCPATH</th>
</tr>
</thead>
</table>

**Use**  
The SMPCPATH (SMP/E classpath) is a directory in the UNIX file system that contains SMP/E Java application classes. The classes are used by RECEIVE ORDER processing to communicate with the IBM Automated Delivery Request server, and by the RECEIVE command and GIMGTPKG, GIMZIP, and GIMUNZIP service routines to calculate SHA-1 hash values when ICSF is not available. The default location for SMPCPATH is /usr/lpp/smp/classes/.

**Attributes**  
Existing directory in the UNIX file system.

**Device**  
Direct access only.

**Notes:**

1. SMPCPATH can be specified using a DD statement or a DDDEF entry. If a value is not specified, the default path of /usr/lpp/smp/classes/ is used.
2. For additional information about how to specify the location of the Java runtime, see “Options that affect Java,” in the “Preparing to use Internet service retrieval” chapter in [SMP/E User’s Guide](#).

### SMPCSI

**ddname**  
SMPCSI.

**Use**  
The data set specified by the ddname SMPCSI is the CSI containing the global zone. (This data set is also known as the master CSI.) The CSI is the database used by SMP/E to record status and other information about the various target and distribution libraries being supported.

**Attributes**  
VSAM; RECORDSIZE(24 143), KEYS(24 0).

**Device**  
Direct access only.

**Notes:**

1. The low-level qualifier of the data set name must be CSI.

2. If you have used IBM SMP/E for z/OS, V3R6 to update a CSI data set, you might not be able to process that data set with previous releases of SMP/E. For more information, see the migration section in [SMP/E User’s Guide](#).

3. When running on systems with the required level of Data Facility Product (DFP), SMP/E automatically takes advantage of the local shared resource (LSR) feature of VSAM. This reduces the number of times SMP/E must access data when it is reading CSI data sets. As a result, SMP/E performance is improved for commands such as APPLY, APPLY CHECK, ACCEPT, ACCEPT CHECK, and especially LIST.

4. CSI data sets should usually be allocated dynamically. However, you may want to use the batch local shared resources (BLSR) subsystem with expanded storage hiperspaces (instead of SMP/E’s implementation of LSR) to improve SMP/E performance during APPLY and ACCEPT processing for a large number of changes.

To do this, you use JCL statements instead of dynamic allocation to define the CSI data sets containing the zones required for processing. For each CSI data set, you need to provide two DD statements:

- The first DD statement is used by SMP/E to enqueue on the data set (to protect it from simultaneous updates) and to trigger batch LSR. This DD statement uses the DDNAME parameter to point to the second DD statement.
- The second DD statement is used by batch LSR to open the CSI data set.

Both DD statements must specify the same DSN value. The following example shows how this can be done:

```bash
//SMPCSI DD DSN=dataset1.CSI,DISP=SHR,
// SUBSYS=(BLSR,'DDNAME=MYSMPCSI',
// 'HBUFND=value','HBUFNI=value')
//MYSMPCSI DD DSN=dataset1.CSI,DISP=SHR
//
//tgtzone DD DSN=dataset2.CSI,DISP=SHR,
// SUBSYS=(BLSR,'DDNAME=MYTGT1',
// 'HBUFND=value','HBUFNI=value')
//MYTGT1 DD DSN=dataset2.CSI,DISP=SHR
```

In this example, note that:

- If the target zone and global zone exist in the same CSI, dataset1 and dataset2 refer to the same data set.
HBUFND and HBUFNI are used instead of BUFND and BUFNI to indicate the use of hiperspace in expanded storage (if available).

tgtzone is the name of the target zone specified on the SET BDY command to be processed.

The DSN parameter must be specified on all DD statements to ensure internal enqueue protection.

5. For more information about using the EXEC statement to specify the CSI data set containing the global zone, see SMP/E User’s Guide.

6. For information about the target and distribution zones in a CSI data set, see “Zone statement” on page 169.

**SMPDATA1**

ddname SMPDATA1.

Use The SMPDATA1 data set is used to store library change file records. The library change file records are written by SMP/E during APPLY or RESTORE processing when the CHANGEFILE subentry of the OPTIONS entry is set to YES. For more information about library change file records, see Chapter 8, “Library change file records.”

Attributes Sequential; BLKSIZE=1069–32670, RECFM=VB, DISP=MOD.

- To have the system determine the optimal block size for the data set, specify BLKSIZE=0.
- If you specify a non-zero BLKSIZE that is less than 1069, SMP/E uses a default of 8800.
- DISP=MOD must be specified to maintain a cumulative history of SMP/E ‘delta’ processing.
- If a DISP other than MOD is specified by the user, SMP/E uses it.
- SMP/E uses LRECL=1065 in allocating the SMPDATA1 data set.

Device Direct access only.

Notes:
1. The SMPDATA1 data set can only be defined to SMP/E with a DD statement or a DDDEF.
2. The SMPDATA1 DD statement or DDDEF must be defined to each target zone to enable ‘delta’ processing to occur for that target zone.
3. Do not concatenate SMPDATA1 data sets.
4. Do not allocate the SMPDATA1 data set as a path in a UNIX file system.

The SMPDATA1 DDDEF can also be defined with the SMP/E Administration dialogs.

**SMPDATA2**

ddname SMPDATA2.

Use The SMPDATA2 data set is used to store the library change file records when the SMPDATA1 data set becomes full. This type of processing is called spill processing. The library change records are
SMPDATA2

written by SMP/E during APPLY or RESTORE processing. For more information about library change file records, see Chapter 8, “Library change file records.”

Attributes

Sequential; BLKSIZE=1069–32760, RECFM=VB, DISP=MOD.

- To have the system determine the optimal block size for the data set, specify BLKSIZE=0.
- If you specify a non-zero BLKSIZE that is less than 1069, SMP/E uses a default of 8800.
- DISP=MOD must be specified.
- If a DISP other than MOD is specified, SMP/E will use it.
- SMP/E will use LRECL=1065 in allocating the SMPDATA2 data set.

Device

Direct access only.

Notes:

1. The SMPDATA2 data set can only be defined to SMP/E with a DD statement or a DDDEF.
2. The SMPDATA2 DD statement or DDDEF must be defined to each target zone to enable spill processing to occur for that target zone.
3. Do not concatenate SMPDATA2 data sets.
4. Do not allocate the SMPDATA2 data set as a path in a UNIX file system.
5. The size of the SMPDATA2 data set will vary depending on how the user intends to manage it.
6. As is the case with SMPLOG data set, the user is responsible for managing the contents and space.

The SMPDATA2 DDDEF can also be defined with the SMP/E Administration dialogs.

SMPDEBUG

ddname SMPDEBUG.

Use

The SMPDEBUG data set contains a dump that was requested by the DEBUG command. Depending on the operands specified, it may contain (1) a dump of SMP/E control blocks and storage areas associated with the specified dump points, or (2) a dump of the VSAM RPL control block and additional RPL information for the specified SMP/E function.

Attributes

Sequential; LRECL=121, BLKSIZE=multiple of 121, RECFM=FBA, DISP=MOD.

Device

SYSOUT, printer, direct access, tape, or terminal.

Notes:

1. BLKSIZE must not exceed 32760.
2. This data set may reside in a UNIX file system. Specify FILEDATA=TEXT and PATHOPTS(OWRONLY,OAPPEND) on the DD statement for this data set, if it is to reside in a UNIX file system.

SMPDIR

ddname SMPDIR.

Use

The SMPDIR data set contains a dump that was requested by the DEBUG command. Depending on the operands specified, it may contain (1) a dump of SMP/E control blocks and storage areas associated with the specified dump points, or (2) a dump of the VSAM RPL control block and additional RPL information for the specified SMP/E function.

Attributes

Sequential; LRECL=121, BLKSIZE=multiple of 121, RECFM=FBA, DISP=MOD.

Device

SYSOUT, printer, direct access, tape, or terminal.

Notes:

1. BLKSIZE must not exceed 32760.
2. This data set may reside in a UNIX file system. Specify FILEDATA=TEXT and PATHOPTS(OWRONLY,OAPPEND) on the DD statement for this data set, if it is to reside in a UNIX file system.
**SMPDIR**

**Use**
The SMPDIR directory identifies the name of a path to be allocated in a UNIX file system. The name is not a complete pathname; it is the directory under which the files produced by GIMZIP are stored.

**Attributes**
Existing directory within a UNIX file system.

**Device**
Direct access.

**Notes:**
1. The SMPDIR directory must be defined with a DD statement.
2. The SMPDIR directory must be defined as a directory in a UNIX file system.
3. The size of the SMPDIR directory depends on the size of the package files produced by GIMZIP.

**SMPDUMMY**

**ddname**
SMPDUMMY

**Use**
SMPDUMMY is intended for use as the definition side deck library for a load module when the IMPORT statements associated with a DLL are not wanted. It allows the SYSDEFSD DD statement to be allocated as a DUMMY data set.

When any of the following SYSDEFSD DD statements is encountered in the link edit JCLIN input stream:

- //SYSDEFSD DD DSN=SMPDUMMY,DISP=xxx
  (where SMPDUMMY may be the lowest-level qualifier of a multi-qualifier data set name)
- or-
- //SYSDEFSD DD DSN=NULLFILE
  (where NULLFILE must be a single-word parameter)
- or-
- //SYSDEFSD DD DUMMY

the SIDE DECK LIBRARY subentry for the load module will be set to SMPDUMMY. When needed for processing, SMP/E will dynamically allocate SMPDUMMY as a DUMMY data set. Any existing SMPDUMMY DDDEF entry or GIMDDALC statement will be ignored. If SMPDUMMY was previously allocated outside of SMP/E, SMP/E will free the SMPDUMMY DD and reallocate it as a DUMMY data set.

**Attributes**
DUMMY.

**Device**
None.

**Note:** The SMPDUMMY ddname is intended only for association with the SIDE DECK LIBRARY subentry for load modules. SMP/E does not prohibit its specification for other library ddnames with the various input methods (MCS, UCLIN, JCLIN, and so on). However, SMPDUMMY will always be allocated by SMP/E as "DD DUMMY". Therefore, the results are unpredictable, in that they depend on each utility's ability to handle a DUMMY data set.

**SMPHOLD**

**ddname**
SMPHOLD.
SMPHOLD

This may refer to an actual data set, or it may refer to a file on a tape (such as file 4 on an ESO tape).

Use

SMPHOLD contains ++HOLD and ++RELEASE statements to be processed by the RECEIVE command.

Attributes

Sequential; LRECL=80, BLKSIZE=multiple of 80, RECFM=FB.

Device

Direct access or tape.

Notes:

1. BLKSIZE must not exceed 32760.
2. This data set may reside in a UNIX file system. Specify
   PATHOPTS(ORDONLY) and either FILEDATA=TEXT or FILEDATA=BINARY
   on the DD statement if it is to reside in a UNIX file system. For simplicity,
   FILEDATA=TEXT is preferred, and ensure the newline character (X'15') marks
   the end of each line. If you use FILEDATA=BINARY, you must ensure that
   each line is padded with blanks to the 80-byte record length because there is no
   end of record marker in binary data.

SMHRPRT

ddname

SMHRPRT

Use

The SMHRPRT data set is an alternate report data set. If SMHRPRT is allocated, the HOLD reports that are generated during RECEIVE, APPLY, ACCEPT and REPORT SYSMODS processing are directed to this data set while other reports are directed to SMPRPT.

Attributes

Sequential; LRECL=121, BLKSIZE=multiple of 121, RECFM=FBA, DISP=MOD.

Device

SYSOUT, printer, direct access, tape, or terminal.

Notes:

1. BLKSIZE cannot exceed 32760.
2. If SMHRPRT is not allocated, all report output goes to the SMPRPT data set. If SMPRPT is not allocated, all report output goes to the SMPOUT data set.
3. If SMHRPRT is allocated to a data set, the disposition must be MOD, because SMP/E opens and closes the SMHRPRT DD statement at each SET command. If the disposition is SHR or OLD, SMHRPRT contains only the reports from the last set of commands that is processed before the end of SMP/E processing.
4. The SMHRPRT ddname can be allocated to a file in the UNIX file system. Specify FILEDATA=TEXT and PATHOPTS(OWRONLY, OAPPEND) on the DD statement if the SMHRPRT ddname is allocated to a file in the UNIX file system. If OAPPEND is not specified, SMHRPRT contains only the reports from the last set of commands that is processed before the end of SMP/E processing.

SMPJCLIN

ddname

SMPJCLIN.

Use

The SMPJCLIN data set contains a job stream of assembly, link-edit, and copy job steps. This data is typically the stage 1 output from the most recent full or partial system generation, but it may be other data in a similar format, such as output from the SMP/E GENERATE command. This job stream is used as input to the JCLIN command to update or create entries in a target zone.
SMPJCLIN

Attributes
Sequential; LRECL=80, BLKSIZE=multiple of 80, RECFM=FB.

Device
Card, tape, direct access, or terminal.

Notes:
1. BLKSIZE must not exceed 32760.
2. This data set may reside in a UNIX file system. Specify PATHOPTS(ORDONLY) and either FILEDATA=TEXT or FILEDATA=BINARY on the DD statement if it is to reside in a UNIX file system. For simplicity, FILEDATA=TEXT is preferred, and ensure the newline character (X'15') marks the end of each line. If you use FILEDATA=BINARY, you must ensure that each line is padded with blanks to the 80-byte record length because there is no end of record marker in binary data.

SMPJHOME

ddbname SMPJHOME

Use
SMPJHOME (SMP/E JAVAHOME) is a directory in the UNIX file system that contains the Java runtime. SMP/E uses Java classes during RECEIVE ORDER processing to communicate with the IBM Automated Delivery Request server, and by the RECEIVE command and GIMGTPKG, GIMZIP, and GIMUNZIP service routines to calculate SHA-1 hash values when ICSF is not available.

SMP/E requires the Java runtime directory when either of the following conditions applies.

• When SMP/E is installing ++JARUPD elements, or
• When a UNIX shell script is invoked during the installation of a file system element, and the shell script issues a Java command.

No default location is assumed by SMP/E for SMPJHOME.

Attributes
Existing directory in the UNIX file system.

Device
Direct access only.

Notes:
1. SMPJHOME can be specified using a DD statement or a DDDEF entry.
2. For additional information about how to specify the location of the SMP/E application classes, see “Options that affect Java,” in the “Preparing to use internet service retrieval” chapter in SMP/E User's Guide.

SMPLIST

ddbname SMPLIST.

Use
The SMPLIST data set contains the output of all LIST commands.

Attributes
Sequential; LRECL=121, BLKSIZE=multiple of 121, RECFM=FBA, DISP=MOD.

Device
SYSOUT, printer, direct access, tape, or terminal.

Notes:
1. BLKSIZE must not exceed 32760.
2. If SMPLIST is not defined, all LIST output goes to the SMPOUT data set.
SMPLIST

3. If SMPLIST is allocated to a data set, the disposition must be MOD, because 
SMP/E opens and closes the SMPLIST DD statement at each SET command. If 
the disposition is SHR or OLD, SMPLIST contains only the output from the last 
set of commands processed before the end of SMP/E processing.

4. This data set may reside in a UNIX file system. Specify FILEDATA=TEXT and 
PATHOPTS(OWRONLY,OAPPEND) on the DD statement for this data set, if it 
is to reside in a UNIX file system. If OAPPEND is not specified, SMPLIST 
contains only the output from the last set of commands processed before the 
end of SMP/E processing.

SMPLOG

<table>
<thead>
<tr>
<th>ddname</th>
<th>SMPLOG.</th>
</tr>
</thead>
</table>
| Use     | The SMPLOG data set (LOG) contains time-stamped records of 
          SMP/E processing. The records in this data set can be written 
          automatically by SMP/E or added by the user through the LOG 
          command. The data set also contains messages issued by SMP/E, 
          as well as detailed information about data set allocation. |
| Attributes | Sequential; BLKSIZE=514-32000, RECFM=VB, DISP=MOD. |
|          | • The BLKSIZE value determines how many records are written to 
          the LOG at one time. As a block is filled, it is written to 
          SMPLOG. |
|          | • DISP=MOD must be specified to maintain a cumulative history 
          of SMP/E processing. |
|          | • SMP/E uses LRECL=510 in allocating the SMPLOG data set. |
| Device   | Tape or direct access. |
| Notes:   | 1. Important: If the data set allocated to the SMPLOG DD statement has the 
          wrong data set attributes, SMP/E will open the data set for output processing 
          with acceptable attributes. This action will overlay the contents of the data set. |
|          | 2. Each zone should have its own SMPLOG data set. |
|          | 3. A secondary log data set (SMPLOGA) should be defined to hold log data when 
          the SMPLOG data set is full. Otherwise, the extra log data is written to the 
          SMPOUT data set, with the date and time stamp encrypted. |
|          | 4. SMPLOG should be updated only by the LOG command or by processing for 
          other SMP/E commands. |
|          | 5. Because some messages are longer than in earlier releases of SMP/E, you may 
          need to increase the size of any data sets used for SMP/E messages (such as 
          SMPOUT). How much more space you need depends on the current size of 
          these data sets and which messages are issued. To start, you may want to 
          allocate new data sets twice the size of the old ones. |

SMPLOGA

<table>
<thead>
<tr>
<th>ddname</th>
<th>SMPLOGA.</th>
</tr>
</thead>
</table>
| Use     | SMPLOGA is a backup LOG data set. If SMPLOGA is defined, it is 
          used automatically when the SMPLOG data set is full. The LOG 
          data set contains time-stamped records of SMP/E processing. The 
          records in this data set can be written automatically by SMP/E or 
          added by the user through the LOG command. The data set also |
**SMPLOGA**

contains messages issued by SMP/E, as well as detailed information about data set allocation.

**Attributes**

Sequential; BLKSIZE=514-32000, RECFM=VB, DISP=MOD.

- The BLKSIZE value determines how many records are written to the LOG at one time. As a block is filled, it is written to SMPLOG.

  If BLKSIZE is less than 514, SMP/E uses the BLKSIZE in effect for the SMPLOG data set.

  The BLKSIZE for the SMPLOGA data set must match the BLKSIZE for the related SMPLOG data set.

- **DISP=MOD** must be specified to maintain a cumulative history of SMP/E processing.

- SMP/E uses LRECL=510 in allocating the SMPLOGA data set.

**Device**

Tape or direct access.

**Notes:**

1. **Important:** If the data set allocated to the SMPLOGA DD statement has the wrong data set attributes, SMP/E will open the data set for output processing with acceptable attributes. This action will overlay the contents of the data set.

2. Each zone should have its own SMPLOGA data set.

3. If the SMPLOGA data set is full, the extra log data is written to the SMPOUT data set, with the date and time stamp encrypted.

4. SMPLOGA should be updated only by the LOG command or by processing for other SMP/E commands.

5. Because some messages are longer than in earlier releases of SMP/E, you may need to increase the size of any data sets used for SMP/E messages (such as SMPOUT). How much more space you need depends on the current size of these data sets and which messages are issued. To start, you may want to allocate new data sets twice the size of the old ones.

---

**SMPLTS**

**ddname**

SMPLTS.

**Use**

The SMPLTS data set is used to maintain the base versions of load modules and program objects for which SYSLIB allocations have been specified. A base version of each load module or program object includes only the modules that were explicitly defined for it by SMP/E MCS statements and JCLIN. Other modules that might be implicitly included in the execution-time load modules and program objects when they are bound are not stored in the base version copy in the SMPLTS.

SMP/E saves the base version of a load module or program object in SMPLTS only when the load module or program object is defined to SMP/E with CALLLIBS and the JCLIN specifies SYSLIB allocations (which are used to create the CALLLIBS subentry lists in the corresponding LMOD entries) and:

- the load module or program object contains XZMOD subentries (indicating cross-zone modules), or

- the target zone was created before SMP/E V3R2 and the UPGRADE command has not yet been used to upgrade it.
SMP/E uses the load modules and program objects in the SMPLTS as input when binding the execution-time load modules and program objects into their specified target libraries.

In some cases, SMP/E may also need to create a temporary member in the SMPLTS to resolve certain warning conditions identified by the binder. This temporary member (if created) is deleted from the SMPLTS after a successful link-edit into the target library.

**Attributes**

Partitioned (DSNTYPE=PDS or DSNTYPE=LIBRARY); RECFM=U, DISP=OLD, BLKSIZE greater than or equal to 6144.

If you allocate the SMPLTS as a PDS, IBM recommends a block size of 32760 to minimize the use of DASD space for this data set.

**Device**

Direct access only.

**Notes:**

1. Each target zone must have its own SMPLTS data set that is not shared by any other target zone.
2. The SMPLTS data set is required for APPLY, LINK LMODS, LINK MODULE, and RESTORE processing.
3. The BLKSIZE of the SMPLTS data set needs to be 1024 or greater to support load modules that specify the SCTR attribute. A large BLKSIZE is recommended to provide maximum space efficiency.
4. The SMPLTS data set is eligible for RETRY processing, in the same manner as other target libraries, after an x37 ABEND. The size of an SMPLTS data set varies depending on the number of load modules that specify a CALLLIBS subentry list in the target zone associated with the SMPLTS.
5. If the SMPLTS data set is to contain program objects, it needs to be allocated as a PDSE.

---

**SMPMTS**

**ddname**

SMPMTS.

**Use**

The SMPMTS data set (MTS) is a target library for macros that exist only in a distribution library (such as macros in SYS1.AMODGEN). This data set allows the current version of these macros to be used for assemblies during APPLY processing. For more information about MTS entries, see "MTSMAC entry (SMPMTS)" on page 290.

**Attributes**

Partitioned; LRECL=80, BLKSIZE=multiple of 80, RECFM=FB, DISP=OLD.

**Device**

Direct access only.

**Notes:**

1. Each target zone must have its own SMPMTS data set that is not shared by any other target zone. This SMPMTS data set may be used with the related distribution zone.
2. For APPLY processing, the SMPMTS data set must be allocated with enough space to hold all the system generation macros, as well as any other macros that do not reside in a target library. This is because functions are now packaged with a complete set of system generation macros.
3. The SMPMTS data set must be in the SYSLIB concatenation for APPLY and RESTORE processing. It can be in the SYSLIB concatenation for ACCEPT processing. For information about SYSLIB concatenation requirements, see SMP/E User’s Guide.

**SMPNTS**

ddname  
SMPNTS.

**Use**

The SMPNTS (SMP/E Network Temporary Store) is a directory of UNIX file system files that are used for temporary storage of network transported packages that were received during SMP/E RECEIVE processing.

**Attributes**

Existing directory within a UNIX file system. A UNIX file system directory name is concatenated with the appropriate subdirectories and file names to create complete pathnames.

- The directory name can be from 1 to 255 characters.
- The directory name must begin and end with a slash (/).
- In addition to the required delimiters (/), a directory name must also be enclosed in single apostrophes (’) if any of the following is true:
  - The directory name contains lowercase alphabetic characters
  - The directory name contains a character that is not uppercase alphabetic, numeric, or national ($, #, or @), slash (/), plus (+), hyphen, period, or ampersand (&).
- The apostrophes must be outside the required delimiters, as in '/directory name/', not '/directory name'/.
- The single apostrophes used to enclose the directory name (the delimiters) do not count as part of the 255-character limit.
- Any apostrophes specified as part of the directory (not the delimiters) must be doubled. Double apostrophes count as two characters in the 255-character limit.
- The directory name can include characters through X’40’ and X’FE’.
- Do not use symbolic substitution.

**Device**

Direct access only.

**Notes:**

1. The SMPNTS can be defined to SMP/E only with a DD statement or a DDDEF.
2. Do not allocate the SMPNTS as anything other than a directory in a UNIX file system.
3. The size of the SMPNTS directory depends on the size of the packages received from the network and stored there.

**SMPOBJ**

ddname  
SMPOBJ.

**Use**

The SMPOBJ data set is used primarily for source-maintained products. It contains preassembled modules that can be used instead of reassembling those modules. These modules must be in
load module format—that is, in the same format as modules residing in the distribution library.

Attributes Partioned; RECFM=U, DISP=SHR.

Device Direct access only.

### SMPOUT

**ddname** SMPOUT.

**Use** The SMPOUT data set contains messages issued during SMP/E processing, as well as dumps of the VSAM RPL, if any dumps were taken. It might also contain LIST output and reports if the SMPHRPT, SMPLIST, and SMPRPT data sets are not defined.

**Attributes** Sequential; LRECL=121, BLKSIZE=multiple of 121, RECFM=FBA, DISP=MOD.

**Device** SYSOUT, printer, direct access, tape, or terminal.

**Notes:**
1. BLKSIZE must not exceed 32760.
2. If SMPOUT is allocated to a data set, the disposition must be MOD, because SMP/E opens and closes the SMPOUT DD statement at each SET command. If the disposition is SHR or OLD, SMPOUT contains only the output from the last set of commands processed before the end of SMP/E processing.
3. If SMPOUT is being allocated for the GIMGTPKG, GIMUNZIP, GIMXSID, or GIMZIP service routine, and you want the resulting SMP/E messages to be formatted to an 80–character length for easier browsing on a terminal, then specify LRECL=81 and BLKSIZE as a multiple of 81.
4. This data set may reside in a UNIX file system. Specify FILEDATA=TEXT and PATHOPTS(OWRONLY,OAPPEND) on the DD statement for this data set, if it is to reside in a UNIX file system. If OAPPEND is not specified, SMPOUT contains only the output from the last set of commands processed before the end of SMP/E processing.

### SMPPARM

**ddname** SMPPARM.

**Use** The SMPPARM data set contains members that allow you to customize SMP/E as follows:

<table>
<thead>
<tr>
<th>Member name</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIMDDALC</td>
<td>Defines data sets to be dynamically allocated</td>
</tr>
<tr>
<td>GIMEXITS</td>
<td>Defines exit routines</td>
</tr>
<tr>
<td>GIMOPCDE</td>
<td>Defines macro and assembler operation codes</td>
</tr>
</tbody>
</table>

**Note:** The File Allocation Reports for the APPLY and ACCEPT commands include status information for the SMPPARM data set, even if an SMPPARM data set is not allocated. SMP/E does this as a reminder to users who may have intended to supply one or more SMPPARM members. If you do not intend to supply any SMPPARM members, you may ignore this status information for the SMPPARM data set.

- For the GIMDDALC member:
SMPPARM

During SET processing, SMP/E determines whether SMPPARM member GIMDDALC was provided. If so, it processes the GIMDDALC control statements contained within the member. For more information about GIMDDALC members, see Chapter 3, “Defining control statements in SMPPARM members,” on page 129.

- For the GIMEXITS member:
  During RECEIVE, APPLY, ACCEPT, RESTORE and LINK processing, SMP/E will determine if SMPPARM member GIMEXITS has been provided. If so, it will process the EXITS control statements contained within the member. For more information about GIMEXITS members, see Chapter 3, “Defining control statements in SMPPARM members,” on page 129.

- For the GIMOPCDE member:
  During JCLIN processing, SMP/E determines whether assembler instructions are macro invocations or OPCODEs. SMP/E contains a default set of OPCODE definitions, which identify standard assembler OPCODEs. If you do not want to use this default set, you can define your own by using the sample GIMOPCDE member supplied to you. For more information about GIMOPCDE members, see Chapter 3, “Defining control statements in SMPPARM members,” on page 129.

Attributes
Partitioned; LRECL=80, BLKSIZE=multiple of 80, RECFM=FB.

Device
Direct access only.

SMPPTFIN

ddname SMPPTFIN.

This ddname can refer to an actual data set or to a file on a tape (such as file 1 on an ESO tape).

Use
The SMPPTFIN data set contains SYSMODs and ++ASSIGN statements to be processed by the RECEIVE command.

Attributes
Sequential; LRECL=80, BLKSIZE=multiple of 80, RECFM=FB.

Device
Card, tape, direct access, or terminal.

Notes:
1. BLKSIZE must not exceed 32760.
2. If the SMPPTFIN data set is inline, you must make sure that the combination of characters used for the delimiter does not occur in the input itself.
   For example, if DD * is specified for SYSIN and the SMPPTFIN data set contains the characters $$, then $$ cannot be used as the default delimiter for the input or for the delimiter specified on the DLM parameter. (The output of SMP/E service routine GIMDTS contains the characters $$.)
3. If you want to receive from multiple product tapes, you cannot concatenate them on a single SMPPTFIN DD statement. Instead, you must process each tape in a separate step, using separate SMPPTFIN DD statements.
   Refer to the documentation supplied with the tape for exact information about how to code the SMPPTFIN DD statement. For example, with a product tape, the program directory contains this information.
4. This data set may reside in a UNIX file system. Specify PATHOPTS(ORDONLY) and either FILEDATA=TEXT or FILEDATA=BINARY.
on the DD statement if it is to reside in a UNIX file system. For simplicity, 
FILEDATA=TEXT is preferred, and ensure the newline character (X'15') marks 
the end of each line. If you use FILEDATA=BINARY, you must ensure that 
each line is padded with blanks to the 80-byte record length because there is no 
end of record marker in binary data.

SMPPTS

ddname  SMPPTS.

Use  The SMPPTS data set (PTS) is used as a repository for SYSMODs. 
It contains one member for each SYSMOD that was received. Each 
member is called an MCS entry and is an exact copy of the 
SYSMOD as it was received from the SMPPTFIN data set. The 
name of an MCS entry matches the SYSMOD ID of the SYSMOD it 
contains. For more information, see “MCS entry (SMPPTS)” on page 273.

Note: SYSMODs in the SMPPTS data set may be stored by SMP/E 
in a compact format. Specifically, inline element data within 
SYSMODs may be compacted during RECEIVE or 
GZONEMERGE command processing. This compact format 
helps to reduce the space requirements of the SMPPTS data 
set. The compacted data is automatically expanded when 
needed during APPLY and ACCEPT command processing. 
See the description of the COMPACT subentry in “OPTIONS 
entry (global zone)” on page 292 for information about how 
to specify whether SYSMODs should be compacted.

To view the original uncompacted form of any SYSMOD in 
the SMPPTS data set, you can use the SMP/E Query Dialogs 
to display the MCS entry in the Global zone for a SYSMOD. 
You can also use the GIMCPTS service routine to expand 
one or more compacted SYSMODs (see “GIMCPTS 
SYSMOD compaction service routine” on page 447 for more 
information about GIMCPTS).

Attributes  Partitioned; LRECL=80, BLKSIZE=multiple of 80, RECFM=FB, 
DISP=OLD.

It is recommended that SMPPTS be allocated as a PDSE 
(DSNTYPE=LIBRARY).

If you plan to run concurrent jobs, specify DISP=SHR instead of 
DISP=OLD.

Unless SMPPTS spill data sets are defined, the SMPPTS must be 
large enough to contain all the SYSMODs that are to be received 
from the SMPPTFIN data set.

Do not concatenate SMPPTS data sets. If you need multiple data 
sets for the SMPPTS, use SMPPTS spill data sets.

You must specify a valid data set name for the SMPPTS. 
NULLFILE and DD DUMMY are invalid for the SMPPTS.

Device  Direct access only.
**SMPPTS spill data set**

**ddname** SMPPTS1 through SMPPTS99.

**Use** SMPPTS spill data sets can be used to store SYSMODs when the SMPPTS data set becomes full. This type of processing is called *spill processing*. SMPPTS spill data sets are used by SMP/E in the same way as the primary SMPPTS data set is used. For more information, see “SMPPTS” on page 157.

**Attributes** Partitioned; LRECL=80, BLKSIZE=multiple of 80, RECFM=FB, DISP=OLD.

It is recommended that SMPPTS spill data sets be allocated as a PDSE (DSNTYPE=LIBRARY).

If you plan to run concurrent jobs, specify DISP=SHR instead of DISP=OLD.

Do not concatenate SMPPTS spill data sets.

The first SMPPTS spill data set must be specified with a ddname of SMPPTS1, the second SMPPTS2, and so on, up to a maximum of SMPPTS99. Do not skip any ddnames in this sequence; if a spill data set is omitted, SMP/E ignores any data sets that may follow the omitted data set. (For example, if you specify only SMPPTS1 and SMPPTS3, then SMP/E uses only SMPPTS1 and ignores SMPPTS3.)

You must specify a valid data set name for an SMPPTS data set. NULLFILE and DD DUMMY are invalid for SMPPTS spill data sets.

**Device** Direct access only.

---

**SMPPUNCH**

**ddname** SMPPUNCH.

**Use** The SMPPUNCH data set contains output from various SMP/E commands. This output generally consists of commands or control statements.

- For BUILDMCS, it contains the complete superseding function SYSMODs.
- For GENERATE, it contains a job stream for building target libraries.
- For REPORT CROSSZONE, it contains commands for installing cross-zone requisites.
- For REPORT ERRSYSMODS, it contains commands for installing SYSMODs that resolve the error hold reason IDs for exception SYSMODs.
- For REPORT SOURCEID, it contains commands for listing SYSMODs associated with the source IDs that were found in the specified zones.
- For REPORT SYSMODS, it contains commands for installing SYSMODs from the input zone that are applicable to the comparison zone.
• For UNLOAD, it contains UCLIN statements for recreating the entries that were unloaded.

Attributes
Sequential; LRECL=80, BLKSIZE=multiple of 80, RECFM=FB, DISP=MOD.

Device
Card, tape, or direct access.

Notes:
1. BLKSIZE must not exceed 32760.
2. If SMPPUNCH is allocated to a data set, the disposition must be MOD, because SMP/E opens and closes the SMPPUNCH DD statement at each SET command. If the disposition is SHR or OLD, SMPPUNCH contains only the output from the last set of commands processed before the end of SMP/E processing.
3. For the UNLOAD command, SMPPUNCH should be allocated to a direct access data set or to tape, because the volume of output is large. You may also want a large BLKSIZE; the larger the BLKSIZE, the fewer the times SMP/E must do I/O.
4. This data set may reside in a UNIX file system. Specify FILEDATA=TEXT and PATHOPTS(OWRONLY,OAPPEND) on the DD statement for this data set, if it is to reside in a UNIX file system. If OAPPEND is not specified, SMPPUNCH contains only the output from the last set of commands processed before the end of SMP/E processing.

SMPRPT

dname
SMPRPT.

Use
The SMPRPT data set contains the reports produced during SMP/E processing.

Attributes
Sequential; LRECL=121, BLKSIZE=multiple of 121, RECFM=FBA, DISP=MOD.

Device
SYSOUT, printer, direct access, tape, or terminal.

Notes:
1. BLKSIZE must not exceed 32760.
2. If SMPRPT is not defined, all report output goes to the SMPOUT data set.
3. If SMPRPT is allocated to a data set, the disposition must be MOD, because SMP/E opens and closes the SMPRPT DD statement at each SET command. If the disposition is SHR or OLD, SMPRPT contains only the reports from the last set of commands processed before the end of SMP/E processing.
4. This data set may reside in a UNIX file system. Specify FILEDATA=TEXT and PATHOPTS(OWRONLY,OAPPEND) on the DD statement for this data set, if it is to reside in a UNIX file system. If OAPPEND is not specified, SMPRPT contains only the reports from the last set of commands processed before the end of SMP/E processing.

SMPSCDS

dname
SMPSCDS.

Use
The SMPSCDS data set (SCDS) contains backup copies of target zone entries that are created during APPLY processing. These backup copies are made before the entries are (1) changed by inline JCLIN, a ++MOVE MCS, or a ++RENAME MCS, or (2) deleted by
SMPPSCDS

an element MCS with the DELETE operand. The backup copies are used during RESTORE processing to return the entries to the way they were before APPLY processing. For more information about BACKUP entries, see “BACKUP entries (SMPPSCDS)” on page 188.

Attributes Partitioned; LRECL=80, BLKSIZE=multiple of 80, RECFM=FB, DISP=OLD.

Device Direct access only.

Note: Each target zone must have its own SMPPSCDS data set; that data set must be unique to that target zone. This SMPPSCDS data set must also be used with the related distribution zone.

SMPPSNAP

**ddname** SMPPSNAP.

**Use** The SMPPSNAP data set is used for snap dump output. When a severe error occurs, such as an abend or severe VSAM return code, SMP/E requests a snap dump of its storage before doing any error recovery. In addition, the DEBUG command may request a snap dump of SMP/E storage when specified messages are issued, or it may request a snap dump of control blocks and storage areas associated with a specified dump point.

Attributes Sequential.

Device SYSOUT, printer, direct access, tape, or terminal.

Note: BLKSIZE must not exceed 32760.

SMPPSRVR

**ddname** SMPPSRVR

**Use** The SMPPSRVR data set contains information about a TCP/IP connected host running an FTP server. It is used by the GIMGTPKG service routine.

Attributes Sequential or member of a partitioned data set; LRECL=80, RECFM=F or FB.

Device Direct access.

Note: This data set may reside in a UNIX file system. Specify PATHOPTS(ORDONLY) and either FILEDATA=TEXT or FILEDATA=BINARY on the DD statement if it is to reside in a UNIX file system. For simplicity, FILEDATA=TEXT is preferred, and ensure the newline character (X'15') marks the end of each line. If you use FILEDATA=BINARY, you must ensure that each line is padded with blanks to the 80-byte record length because there is no end of record marker in binary data.

SMPPSTS

**ddname** SMPPSTS.

**Use** The SMPPSTS data set (STS) is a target library for source that exists
only in a distribution library. This data set allows the current version of these modules to be used for assemblies during APPLY processing.

Attributes
Partitioned; LRECL=80, BLKSIZE=multiple of 80, RECFM=FB, DISP=OLD.

Device
Direct access only.

Note: Each target zone must have its own SMPSTS data set, which may not be shared by any other target zone. This SMPSTS data set can also be used with the related distribution zone.

SMPTLIB

dname
SMPTLIB.

Use
SMPTLIB data sets (TLIBs) are used as temporary storage for relative files that are loaded from SMPPTFIN during RECEIVE processing. They are deleted when the associated SYMMOD is deleted by REJECT, RESTORE, or ACCEPT processing.

You can have SMP/E dynamically allocate the TLIB data sets, or you can allocate them yourself before RECEIVE processing. For information about how SMP/E allocates the TLIBs, see the RECEIVE Command chapter in [SMP/E Commands](#). (Regardless of how the SMPTLIB data sets are allocated, they do not appear in the File Allocation report.)

Notes:
1. No DD statement or DDDEF entry is required for the SMPTLIB data sets if they are preallocated and cataloged.
2. If you allocate and catalog the SMPTLIB data sets yourself, make sure they are allocated on the volume specified in the catalog.
3. If you allocate the SMPTLIB data sets yourself but do not catalog them, make sure they are allocated on the volume specified in the SMPTLIB DD statement or DDDEF entry being used for the command you are processing.
4. If you need to specify a unit that is not SYSALLDA, and the unit is not set by use of a STORCLAS or an ACS filter routine, then you must use a DDDEF entry instead of a DD statement to allocate SMPTLIB data sets and specify the unit value in the DDDEF entry.
5. If you are using SMS to manage your data sets, you can set up the unit, volume, and space allocation through a STORCLAS or an ACS filter routine, instead of specifying them on a DD statement or DDDEF entry.
6. If you are using SMS to manage your data sets, do not specify dummy volumes on a DD statement or in a DDDEF entry for the SMPTLIB allocation. Dummy volumes used as indicators to ACS routines for SMS class selection can cause operator mount messages to be issued for the non-existent dummy volumes. This is because SMP/E first attempts to allocate the SMPTLIB data sets as if they already exist on the volume, which can cause operator mount messages to be issued for the dummy volume.
SMPTLIB

7. SMPTLIB data sets should not be allocated as PDSEs, because IEBCOPY does not support copying an unloaded PDS load library from tape to a PDSE load library on DASD.

8. DDDEFs are not required in target or distribution zones for SMPTLIB data sets, unless you have uncataloged the SMPTLIB data sets from the global zone (this is not recommended). In this case, only the VOLUME and UNIT subentries are accepted as input on UCLIN statements for target or distribution zone SMPTLIB DDDEFs.

Attributes

Here are two examples of SMPTLIB DD statements:

/SMPTLIB DD UNIT=SYSALLDA, VOL=SER=SMPVOL
//SMPTLIB DD UNIT=SYSALLDA,
//   VOL=SER=(SMPVL1, SMPVL2)

If you use DDDEFs to have SMP/E dynamically allocate the SMPTLIB data sets, you cannot specify the initial or final disposition. SMP/E determines the disposition based on the command it is processing.

Device

Direct access only.

SMPTLOAD

ddname SMPTLOAD.

Use

SMP/E may need to use an SMPTLOAD data set when installing program elements that were packaged inline. SMPTLOAD is required when applying or accepting a program element and either:

- the destination data set (target or distribution library) is a PDS and the unloaded inline data represents a PDSE that contains program elements
- the destination data set is a PDSE and the unloaded inline data set has RECFM=U and is not a PDSE.

SMP/E dynamically allocates a new SMPTLOAD data set whenever one is needed.

Attributes

Partitioned (DSNTYPE=PDS or DSNTYPE=LIBRARY), DISP=(NEW,DELETE).

Device

Direct access.

Notes:

1. If the unloaded inline data represents a PDSE, SMP/E allocates SMPTLOAD with DSNTYPE=LIBRARY. Otherwise, SMP/E allocates SMPTLOAD with DSNTYPE=PDS.

2. All other allocation information for the SMPTLOAD is obtained from the SMPTLOAD DDDEF or, if a DDDEF does not exist, from GIMDDALC control statements found in SMPPARM member GIMDDALC. If a DD statement is supplied for SMPTLOAD, it is freed and the allocation for SMPTLOAD is done using the DDDEF or the GIMDDALC control statement found in SMPPARM member GIMDDALC.

SMP/E uses the following information, if it is supplied by the user, to dynamically allocate an SMPTLOAD data set:
SMPTLOAD

- Primary and secondary space amounts
- Allocation units for primary and secondary space
- Number of directory blocks
- A unit designation
- A volume designation
- A STORCLAS designation
- A MGMTCLAS designation
- A DATACLAS designation

If SMP/E cannot get the required allocation information from a DDDEF or a GIMDDALC control statement in the GIMDDALC member of SMPPARM, then the allocation fails.

3. If there is an SMPTLOAD DDDEF, SMP/E tries to allocate the SMPTLOAD data set, even if the DDDEF does not supply any of the previously listed information. This is because SMP/E always tries to allocate the SMPTLOAD data set new with either a DSNTYPE of PDS or LIBRARY, which may be sufficient if an ACS routine exists to supply other needed allocation values.


5. SMP/E will not use an SMPTLOAD data set specified with JCL. If an SMPTLOAD DD statement is specified, SMP/E dynamically frees the SMPTLOAD ddname prior to allocating a new SMPTLOAD data set.

SMPWKDIR

ddname

SMPWKDIR.

Use

The SMPWKDIR directory identifies the name of a directory in a UNIX file system. SMPWKDIR is used to store temporary files created during SMP/E processing. If SMPWKDIR is not specified, SMP/E will use one of the following directories for temporary files, depending on the command or service:

- APPLY and ACCEPT commands: the system /tmp directory
- RECEIVE command: the system /tmp directory and the package directory of the SMPNTS directory specified on a DD statement or DDDEF entry.
- GIMZIP and GIMUNZIP service routines: the package directory specified on the SMPDIR DD statement.

Attributes

Existing directory in a UNIX file system.

Device

Direct access only.

Note: The SMPWKDIR can be defined for the RECEIVE, APPLY, and ACCEPT commands using either a DD statement or a DDDEF entry. It can be defined for the GIMZIP and GIMUNZIP service routines using a DD statement only.

SMPWRK1

ddname

SMPWRK1.

Use

The SMPWRK1 data set is used as temporary storage for macro updates and replacements that will be processed by the IEBUPDTE and IEBCOPY programs. During APPLY and ACCEPT processing, SMP/E places the input in this data set before calling the utility.
SMPWRK1

Attributes
Partitioned; LRECL=80, BLKSIZE=multiple of 80, RECFM=FB, DISP=(NEW,DELETE).

Specifying **DISP=(NEW,DELETE)** minimizes space loss problems. SMPWRK1 is generally needed only for the duration of the SMP/E job step. To keep the data set longer than that, you must use a different DISP value and compress the data set to reclaim space.

The SMS-related options DATACLAS, MGMTCLAS, STORCLAS, and DSNTYPE provide additional data set support. For more information, refer to [z/OS DFSMS Using Data Sets](https://www.ibm.com).  

Device
Direct access only.

Notes:
1. BLKSIZE must not exceed 32760.
2. If the BLKSIZE value is omitted or inapplicable, SMP/E uses a default of 0 which allows the system to determine the most appropriate block size.

---

SMPWRK2

**ddname**
SMPWRK2.

**Use**
The SMPWRK2 data set is used as temporary storage for source updates and source replacements that will be processed by the IEBUPDTE and IEBCOPY programs. During APPLY and ACCEPT processing, SMP/E places the input in this data set before calling the utility.

**Attributes**
Partitioned; LRECL=80, BLKSIZE=multiple of 80, RECFM=FB, DISP=(NEW,DELETE).

Specifying **DISP=(NEW,DELETE)** minimizes space loss problems. SMPWRK2 is generally needed only for the duration of the SMP/E job step. To keep the data set longer than that, you must use a different DISP value and compress the data set to reclaim space.

The SMS-related options DATACLAS, MGMTCLAS, STORCLAS, and DSNTYPE provide additional data set support. For more information, refer to [z/OS DFSMS Using Data Sets](https://www.ibm.com).

**Device**
Direct access only.

**Notes:**
1. BLKSIZE must not exceed 32760.
2. If the BLKSIZE value is omitted or inapplicable, SMP/E uses a default of 0 which allows the system to determine the most appropriate block size.

---

SMPWRK3

**ddname**
SMPWRK3.

**Use**
The SMPWRK3 data set is used as temporary storage for object modules supplied by a SYSMOD, object modules created by assemblies, and IMASPZAP control cards following ++ZAP statements.

**Attributes**
Partitioned; LRECL=80, BLKSIZE=multiple of 80, RECFM=FB, DISP=(NEW,DELETE).
Specifying `DISP=(NEW,DELETE)` minimizes space loss problems. SMPWRK3 is generally needed only for the duration of the SMP/E job step. To keep the data set longer than that, you must use a different `DISP` value and compress the data set to reclaim space.

If you want to save assembled object modules until SMP/E has successfully applied or accepted the SYSMODs that caused the assemblies, specify `DISP=(NEW,KEEP)`. This allows SMP/E to reuse the assembled object modules if the APPLY or ACCEPT command fails. For more information about reusing assemblies, see `SMP/E Commands`.

**Note:** If SMPWRK3 is a permanent data set, make sure to specify `OLD` as the initial disposition. Do not specify `SHR`. An initial disposition of `SHR` may cause an abend.

The SMS-related options DATACLAS, MGMTCLAS, STORCLAS, and DSNTYPE provide additional data set support. For more information, refer to `z/OS DFSMS Using Data Sets`.

**Device**
Direct access only

**Notes:**
1. `BLKSIZE` must not exceed 32760.
2. If the `BLKSIZE` value is omitted or inapplicable, SMP/E uses a default of 0 which allows the system to determine the most appropriate block size.

---

**SMPWRK4**

**ddname** SMPWRK4.

**Use**
The SMPWRK4 data set is used as temporary storage for IMASPZAP and link-edit input containing EXPAND control statements.

**Attributes**
Partitioned; `LRECL=80, BLKSIZE=multiple of 80, RECFM=FB, DISP=(NEW,DELETE)`.

Specifying `DISP=(NEW,DELETE)` minimizes space loss problems. SMPWRK4 is generally needed only for the duration of the SMP/E job step. To keep the data set longer than that, you must use a different `DISP` value and compress the data set to reclaim space.

The SMS-related options DATACLAS, MGMTCLAS, STORCLAS, and DSNTYPE provide additional data set support. For more information, refer to `z/OS DFSMS Using Data Sets`.

**Device**
Direct access only.

**Notes:**
1. `BLKSIZE` must not exceed 32760.
2. If the `BLKSIZE` value is omitted or inapplicable, SMP/E uses a default of 0 which allows the system to determine the most appropriate block size.

---

**SMPWRK6**

**ddname** SMPWRK6.

**Use**
The SMPWRK6 data set is used during APPLY and ACCEPT processing as temporary storage for inline replacements for data...
SMPWRK6

Elements. It is also used as temporary storage for inline updates for JAR elements. SMP/E places the input in this data set so it can be directly accessed and installed by the copy utility or SMP/E.

**Attributes**
Partitioned; LRECL=80, BLKSIZE=multiple of 80, RECFM=FB, DISP=(NEW,DELETE).

Specifying **DISP=(NEW,DELETE)** minimizes space loss problems. SMPWRK6 is generally needed only for the duration of the SMP/E job step. To keep the data set longer than that, you must use a different DISP value and compress the data set to reclaim space.

The SMS-related options DATACLAS, MGMTCLAS, STORCLAS, and DSNTYPE provide additional data set support. For more information, refer to [z/OS DFSMS Using Data Sets](#).

**Device**
Direct access only.

**Notes:**
1. BLKSIZE must not exceed 32760.
2. If the BLKSIZE value is omitted or inapplicable, SMP/E uses a default of 0 which allows the system to determine the most appropriate block size.

SMPnnnnn

**ddname**
SMPnnnnn, where nnnnn is a number from 00000 through 99999.

**Use**
SMP/E allocates SMPnnnnn data sets for its own internal processing. To avoid processing errors, do not assign such a ddname to any data set.

SYSIN

**ddname**
SYSIN.

**Use**
Contains package control statements used as input to the GIMZIP and GIMUNZIP service routines.

**Attributes**
Sequential or a member of a partitioned data set; LRECL=80

**Device**
Direct access or SYSIN file.

**Notes:**
1. SYSIN can be defined to GIMZIP and GIMUNZIP only with a DD statement.
2. The size of the input data set will vary depending on how you intend to manage it.
3. You are responsible for managing the contents and space of the input data set.
4. This data set may reside in a UNIX file system. Specify **PATHOPTS(ORDONLY)** and either **FILEDATA=TEXT** or **FILEDATA=BINARY** on the DD statement if it is to reside in a UNIX file system. For simplicity, **FILEDATA=TEXT** is preferred, and ensure the newline character (X'15') marks the end of each line. If you use **FILEDATA=BINARY**, you must ensure that each line is padded with blanks to the 80-byte record length because there is no end of record marker in binary data.

SYSLIB

**ddname**
SYSLIB.
SYSLIB

Use

The SYSLIB data set is a concatenation of macro libraries that are to be used by the assembler utility.

For APPLY and RESTORE processing, the data sets should be concatenated in this order:

1. SMPMTS
2. MACLIB
3. MODGEN
4. Target system macro libraries (such as libraries specified for SYSLIB on the ++MAC statement)
5. Distribution macro libraries (such as libraries specified for DISTLIB on the ++MAC statement)

For more information about the proper SYSLIB concatenation for ACCEPT processing, see the SMP/E User’s Guide.

Attributes

Partitioned; LRECL=80, BLKSIZE=multiple of 80, RECFM=FB.

Device

Direct access only.

SYSPRINT

ddname

SYSPRINT.

You can specify different SYSPRINT data sets for each of the utilities that SMP/E calls. This can be done with the PRINT value in the appropriate UTILITY entry. For more information, see “UTILITY entry (global zone)” on page 350.

Use

The SYSPRINT data set contains output from the utilities called by SMP/E.

Attributes

Sequential; DISP=MOD.

Do not specify LRECL, BLKSIZE, or RECFM unless they are compatible with the attributes used by the utilities called.

If SYSPRINT is allocated to a data set, the disposition must be MOD, because SMP/E opens and closes the SYSPRINT DD statement at each SET command. If the disposition is SHR or OLD, SYSPRINT contains only the output from the last set of commands processed before the end of SMP/E processing.

Device

SYSOUT, printer, direct access, or tape.

SYSOUT or a tape is recommended, because SYSPRINT might be opened with different DCB attributes by the various utilities and service aids called by SMP/E.

Notes:

1. BLKSIZE must not exceed 32760.
2. How you specify the SYSPRINT data set can affect whether a listing of the utility output is produced. For example, no listing is produced if the PRINT value in the UTILITY entry specifies either of the following:
   • A DDDEF for a DUMMY data set
   • A DDDEF for a data set that is sent to a SYSOUT class that suppresses output

Chapter 4. SMP/E data sets and files 167
SYSPUNCH

ddname SYSPUNCH.
Use The SYSPUNCH data set is a temporary data set containing object modules assembled by running the job stream produced by system generation or the GENERATE command. These modules are not installed in the distribution libraries at ACCEPT time.
Attributes Partitioned; LRECL=80, BLKSIZE=multiple of 80, RECFM=FB.
Do not specify a DISP value. Instead, let SMP/E use its defaults:
• DISP=(NEW,PASS) for the first job generated
• DISP=(OLD,PASS) for any subsequent jobs
Device Direct access only.

SYSUT1, SYSUT2, and SYSUT3

ddname SYSUT1, SYSUT2, and SYSUT3.
Use These are scratch data sets for SMP/E and the utilities it calls. They can be used instead of the following data sets when certain utilities are called:
• SYSIN
  – For invocations of the copy utility
  – For some invocations of update utility
  – For some invocations of the x37 RETRY COMPRESS utility
• SYSLIN for invocations of the link-edit utility
• SYSUT2 for invocations of the assembler utility
SYSUT1 and SYSUT2 are also used by the GIMDTS service routine:
• SYSUT1 points to the data set containing the input in its original format.
• SYSUT2 points to the data set containing the transformed output.
Attributes Sequential; DISP=(NEW,DELETE)
Device Direct access only.
Notes:
1. BLKSIZE must not exceed 32760.
2. Do not specify BLKSIZE=0. SMP/E does not support system-determined block size (SDB) for this data set.
3. If the BLKSIZE value is omitted or invalid, SMP/E uses a default of 3200.
4. When processing partitioned data sets, GIMUNZIP dynamically allocates SYSUT1 as a large format sequential data set (DSNTYPE=LARGE).

SYSUT4

ddname SYSUT4.
Use This data set can be used instead of the SYSIN data sets when certain utilities are called:
• For invocations of the x37 RETRY COMPRESS utility
• For some invocations of the assembler utility

Notes:
1. BLKSIZE must not exceed 32760.
2. Do not specify BLKSIZE=0. SMP/E does not support system-determined block size (SDB) for this data set.
3. If the BLKSIZE value is omitted or invalid, SMP/E uses a default of 3200.
4. When processing partitioned data sets, GIMUNZIP dynamically allocates SYSUT1 as a large format sequential data set (DSNTYPE=LARGE).
• For RECEIVE FROMNETWORK and RECEIVE FROMNTS command processing

**Attributes**
Sequential; DISP=(NEW,DELETE)

**Device**
Direct access only.

**Notes:**
1. BLKSIZE must not exceed 32760.
2. Do not specify BLKSIZE=0. SMP/E does not support system-determined block size (SDB) for this data set.
3. If the BLKSIZE value is omitted or invalid, SMP/E uses a default of 3200.

---

### Target library

**ddname**
The ddname for a target library should match the low-level qualifier of the data set name. For example, the ddname for SYS1.LINKLIB should be LINKLIB.

**Use**
Target libraries contain updated versions of macros, source modules, and load modules that were stored during APPLY or RESTORE processing. They are the libraries used for your running system. You must provide a DDDEF entry or DD statement for each target library that is being processed.

**Attributes**
Partitioned.

**Device**
Direct access only.

---

### Text library (TXLIB)

**ddname**
The ddname for a text library must match the TXLIB value on the ++JCLIN or element MCS. For example, the ddname for the text library on statement ++MOD(MODA) TXLIB(LIBX) must be LIBX.

**Use**
Text libraries contain JCLIN input or replacements for macros, source, or object modules that have not been link-edited. They are used when the JCLIN or elements are provided in partitioned data sets rather than inline or in relative files.

**Attributes**
Partitioned.

**Device**
Direct access only.

---

### Zone statement

**ddname**
The ddname must match the name of the target or distribution zone.

**Use**
The zone DD statement is used by SMP/E to access an individual target or distribution zone in a CSI data set. For example, to have SMP/E access target zone MVSTGT1, you can provide a DD statement like this one:

```plaintext
//MVSTGT1 DD DSN=SMPE.SMPCSI.CSI,DISP=SHR
```

If you do not provide a DD statement for a target or distribution zone, SMP/E allocates the zone dynamically using the ZONEINDEX information in the GLOBALZONE entry. Also note that, while DD statements may be used to override the
Zone statement

ZONEINDEX information, they are not a substitute for a zoneindex. A zoneindex is always required for a zone.

Attributes

VSAM; RECORDSIZE(24 143), KEYS(24 0).

Device

Direct access only.

Notes:

1. The low-level qualifier of the data set name must be CSI.
2. If you have used IBM SMP/E for z/OS, V3R6 to update a zone, you might not be able to process that zone with previous releases of SMP/E. For more information, see the migration section in [SMP/E User’s Guide](#).
3. When running on systems with the required level of DFP, SMP/E automatically takes advantage of the local shared resource (LSR) feature of VSAM. This reduces the number of times SMP/E must access data when it is reading CSI data sets. As a result, SMP/E performance is improved for commands such as APPLY, APPLY CHECK, ACCEPT, ACCEPT CHECK, and especially LIST.
4. CSI data sets should usually be allocated dynamically. However, you may want to use the batch local shared resources (BLSR) subsystem with expanded storage hiperspaces (instead of SMP/E's implementation of LSR) to improve SMP/E performance during APPLY and ACCEPT processing for a large number of changes. For more information about BLSR, see the notes section in [“SMPCSNI” on page 145](#).
5. For information about the CSI data set containing the global zone, see [“SMPCSNI” on page 145](#).
Chapter 5. SMP/E data set entries

This chapter describes the entries in the various data sets SMP/E uses. It discusses the following:
- How these data sets are organized
- How the entries in these data sets are organized
- How to create, update, and obtain information about these entries

**Note:** The LIST, UCLIN, and UNLOAD commands used to process these entries are described in [SMP/E Commands](#).

How the data sets are organized

SMP/E uses the following data sets as a database for its processing: the CSI, PTS, SCDS, MTS, and STS. It is important to understand how SMP/E uses each of these data sets and how they are related.

The following is a description of each data set:

- The CSI data set is a VSAM data set that serves as the primary data set for SMP/E. SMP/E divides the CSI into multiple partitions using the VSAM key structure. Each partition is referred to as a zone.
  
  There are three types of zones:
  
  1. **Global zone.** A single global zone is used to record information about SYSMODs that have been received. The global zone also contains information enabling SMP/E to access the other two types of zones, and information allowing you to tailor parts of SMP/E processing.
  2. **Target zones.** One or more target zones are used to record information about the status and structure of the operating system (or target) libraries.
  3. **Distribution zones.** One or more distribution zones are used to record information about the status and structure of the distribution libraries.

  In [Figure 5 on page 172](#), the CSI is pictured as one data set. In fact, you can group all your zones within one VSAM data set (up to 32766 zones per data set), or divide them up into multiple VSAM data sets, even one zone per data set.

  The choice is yours; it is based on such factors as:
  
  - Which packs the associated target and distribution libraries are on. It is advisable to keep the associated zone on the same pack as the libraries controlled from the zone, so when the pack is dumped, the data and description are kept synchronized.
  
  - The organization of your shop. Totally separate service organizations for different products may require many separate data sets.

  Figure 6 on page 173 illustrates the same zone relationship as Figure 5 on page 172, with the CSI spread across multiple VSAM data sets.

- The PTS is used strictly as a storage data set for SYSMODs. A SYSMOD is read from the SMPPTFIN data set and stored directly on the PTS without any modifications or SMP/E information. It is, therefore, related to the global zone in that both data sets contain information about the received SYSMODs. You can, therefore, look at the global zone and the PTS as a pair of data sets that must be processed (such as scratched, saved, or modified) concurrently.
Data set entries

- The **SCDS** is used by SMP/E to store backup copies of modified target zone entries during apply processing of a SYSMOD with inline JCLIN. Thus, the SCDS is directly related to a specific target zone, and each target zone must have its own SCDS.

- The **MTS** is a library in which SMP/E stores updated copies of macros during apply when no other target macro library is identified. The MTS is, therefore, related to a specific target zone, and each target zone must have its own MTS data set.

- The **STS** is a library in which SMP/E stores updated copies of source during APPLY when no other target source library is identified. The STS is, therefore, related to a specific target zone, and each target zone must have its own STS data set.

The relationships among SMP/E data sets and zones are shown in Figure 5 for a single CSI data set, and in Figure 6 on page 173 for multiple CSI data sets.

![Figure 5. Single-CSI structure](image-url)
How data set entries are organized

Within the global zone, target zone, and distribution zone, SMP/E keeps many types of entries. These can be divided into two categories:

- Those that are used to control SMP/E processing. These consist of:
  - GLOBALZONE definition entry
  - TARGETZONE definition entry
  - DLIBZONE definition entry
  - OPTIONS entries
  - UTILITY entries
  - DDDEF entries
  - FMIDSET entries
  - ZONESET entries

- Those that are used to define the status and structure of the target libraries and distribution libraries. These consist of:
  - ASSEM entries

Figure 6. Multiple-CSI structure
Data set entries

- Data element entries
- DLIB entries
- hierarchical file system element entries
- LMOD entries
- MAC entries
- MOD entries
- SRC entries
- SYSMOD entries

Note: The SYSMOD entries also contain information referred to as HOLDDATA.

The following sections describe the entry relationships for the entries that control processing, and entries that define status and structure.

Entries that control processing

The starting point for all SMP/E processing is the SMPCSI DD statement. This DD statement directs SMP/E to the CSI data set containing the GLOBALZONE entry. The next step after obtaining the GLOBALZONE entry depends on which zone you direct SMP/E to process. (For more information about identifying which zone SMP/E is to process, see the SET command chapter in SMP/E Commands.

Processing the global zone

Once the GLOBALZONE entry has been obtained and you direct SMP/E to process the global zone (for instance to receive SYSMODs), the OPTIONS subentry in the GLOBALZONE entry directs SMP/E to the correct OPTIONS entry to use. The OPTIONS entry contains information about the SMP/E processing options to be used. During processing, SMP/E may have to invoke one of the system utilities. Another entry, the UTILITY entry, is used to define to SMP/E information about the utility program to invoke, the parameters to pass to it, and the return code to expect from it. SMP/E finds the UTILITY entry through the OPTIONS entry that names the UTILITY entry. Thus, for SMP/E to use the correct utility program, you must define both the UTILITY entry that describes the utility program and the OPTIONS entry that names that UTILITY entry.

The other processing entries in the global zone are the DDDEF, FMIDSET, and ZONESET entries.

- DDDEF entries provide information used by SMP/E to dynamically allocate data sets. These entries are not connected to any other processing entry. They are accessed individually, by name, as SMP/E needs information to dynamically allocate a specific data set. They are not shown in Figure 7 on page 175.
- FMIDSET entries define sets of FMIDs. Various other commands, such as APPLY and ACCEPT, can then process SYSMODs by FMIDSET.
- ZONESET entries define sets of zones. The REPORT command can then check for SYSMODs in the zones defined in the ZONESET.

Each of these entries (GLOBALZONE, OPTIONS, UTILITY, DDDEF, FMIDSET, and ZONESET) exists in the global zone.

Figure 7 on page 175 illustrates how the various global zone processing entries are related to one another. Subsequent sections of this chapter provide a more detailed explanation of each entry and the data it contains.
Processing a target zone or a distribution zone

A target zone and a distribution zone are processed in very similar ways. The only difference is the name of the controlling entry. Therefore, they are covered together, and the differences are pointed out as appropriate.

When you direct SMP/E to process a target zone (for instance to apply a PTF) or a distribution zone (for instance to accept that PTF), SMP/E accesses the ZONEINDEX subentries in the GLOBALZONE entry. Those subentries list the target zones and distribution zones that you have defined, including the zone type, target or DLIB, and the name of the CSI data set on which they reside.

After SMP/E has determined that the zone specified is valid, it uses the CSI data set name specified to dynamically allocate a DD statement to access the required data set (optionally, SMP/E looks for a DD statement equal to the zone name). SMP/E can now access that DD statement to obtain further processing information for the zone.

The first entry accessed is the TARGETZONE entry (for target zones) or the DLIBZONE entry (for distribution zones). Each of these, in turn, directs SMP/E to the correct OPTIONS entry to use, which in turn directs SMP/E to the correct UTILITY entries. Target and distribution zones also contain DDDEF entries.

- The OPTIONS and UTILITY entries serve the same purpose for the target zone and distribution zone as for the global zone. However, the OPTIONS and UTILITY entries used to process a target zone or distribution zone are defined not in the target or distribution zone, but in the global zone that points to the target or distribution zone.
- TARGETZONE and DLIBZONE entries contain a RELATED subentry, which identifies a related zone. For a target zone, the RELATED subentry identifies the
distribution zone from which this target zone was built. For a distribution zone, the RELATED subentry identifies the target zone that was built from these distribution libraries.

- **DDDEF entries** in the target zone and distribution zone are not connected to any other entries. They provide information used by SMP/E to dynamically allocate data sets.

Figure 8 on page 177 shows how the various target zone, distribution zone, and global zone processing entries are related. Subsequent sections of this chapter provide a more detailed explanation of each entry and the data it contains. Because neither the FMIDSET entries nor the DDDEF entries are connected to any other distribution zone or target zone entry, they are not shown in this figure.

### Entries that define status and structure

Once the processing control information for the specified zone has been determined, SMP/E uses the status and structure entries within the various zones to determine what should be done.

#### Processing the global zone

Processing the global zone is fairly simple, in that only one structure and status entry exists, SYSMOD. SYSMOD entries are used to determine whether a SYSMOD has been received already. Various indicators within the SYSMOD entry also indicate whether the SYSMOD has been applied or accepted. The SYSMOD entries, although not directly connected to the PTS MCS entries, are implicitly connected, in that SMP/E assumes that there is a one-to-one relationship between global zone SYSMOD entries and PTS MCS entries.

#### Processing the target zone

Processing the target zone is much more complex than processing the global zone, because the operations performed are much more complex and require more entry types to direct SMP/E. The primary purpose of the target zone entries is to enable SMP/E to apply new function and service to the target system libraries; thus, the APPLY command is used to describe the relationship between the various target zone entries.

The starting point for applying a SYSMOD is actually the global zone SYSMOD entries. From the global zone SYSMOD entry, SMP/E obtains information about the relationship between this SYSMOD and other SYSMODs. SMP/E then uses other global zone SYSMOD entries and the target zone SYSMOD entries to resolve these relationships.

Once the eligible SYSMODs have been determined, the actual MCS statements from the PTS are accessed and lead SMP/E to the other target zone entries:

- ++MAC and ++MACUPD statements lead to target zone MAC entries.
- ++SRC and ++SRCUPD statements lead to target zone SRC entries.
- ++MOD and ++ZAP statements lead to target zone MOD and LMOD entries.
- Data element statements lead to target zone data element entries.
- ++hfs_element statements lead to target zone hierarchical file system element entries.
If the SYSMOD replaces or modifies a macro (++MAC or ++MACUPD in SYSMOD), SMP/E uses the target zone MAC entry to determine the functional and service level of the macro and to determine whether any assemblies must be redone as a result of the macro update. This is done by checking the GENASM subentries in the target zone MAC entry. The GENASM subentries contain the names of either ASSEM entries or SRC entries in the target zone. In either case, SMP/E accesses the appropriate entry and uses the information stored there to perform the required assemblies.
If the SYSMOD replaces or modifies source (++SRC or ++SRCUPD in SYSMOD), SMP/E uses the target zone **SRC entry** to determine the functional and service level of the source and the library containing the source code, and then performs the required assembly.

After an assembly has been done (either using data from the ASSEM entry or SRC entry), SMP/E knows that the resulting object deck must be link-edited into the target libraries somewhere. Although there is no direct connection (that is, no subentry value present) between the ASSEM or SRC entries and a **MOD entry**, there is an implicit relationship. SMP/E assumes that for each ASSEM and SRC entry, there exists a MOD entry with the same name. Thus, after performing the assembly, SMP/E accesses the corresponding MOD entry to determine where to install the object deck. This leads us to the same point as if a MOD were supplied in the SYSMOD.

If the SYSMOD replaces or modifies a module (++MOD or ++ZAP in SYSMOD), or if an assembly was done for a SRC or ASSEM entry, SMP/E uses the target zone **MOD entry** to determine the functional and service level of the module and the load modules into which the module should be linked. Information about load modules is kept in the LMOD subentries in the MOD entry.

The LMOD subentries within the MOD entry lead SMP/E to the target zone **LMOD entries**. These represent load modules that exist in the target libraries. LMOD entries contain all the information necessary either to relink the load module or to superzap it.

If the SYSMOD replaces a data element (a data element MCS is in the SYSMOD), SMP/E uses the target zone **data element entry** to determine the functional and service level of the data element. It then gets the data element installed in the appropriate target library.

If the SYSMOD replaces a hierarchical file system element (a ++hfs_element MCS is in the SYSMOD), SMP/E uses the target zone **hierarchical file system element entry** to determine the functional and service level of the hierarchical file system element. It then gets the hierarchical file system element installed in the appropriate target library (which is actually in a UNIX file system). If a shell script has been defined for the element, SMP/E passes control to the shell script to allow it to perform any necessary post-installation processing.

After all the updating, assembling, linking, and so on, are done, you arrive back almost at the starting point, the SYSMOD entry. The difference is that this is the target zone SYSMOD entry rather than the global zone SYSMOD entry. The target zone SYSMOD entry contains all the information about what has been done as a result of installing this SYSMOD.

That leaves us with one target zone entry not discussed, the **DLIB entry**. The DLIB entries are not connected to any other entry. They are used by SMP/E to keep information about libraries that are totally copied during product installation, and are used during APPLY processing to create the appropriate element and LMOD entries for elements coming from the copied DLIB.

Figure 9 on page 179 shows how the various SMP/E status and structure entries are related. Subsequent sections of this chapter provide a more detailed explanation of each entry and the data it contains.
Note: Lines to the left of the figure are implicit connections based on the assumption that entries with equal names will be found.

Processing the distribution zone

Processing the distribution zone is similar to processing the target zone, but more complex than processing the global zone. The primary purpose of the distribution zone entries is to enable SMP/E to accept new function and service to the
distribution libraries; it is also used to remove a SYSMOD from the system. The ACCEPT command will be used to describe the relationship between the various distribution zone entries.

The starting point for accepting a SYSMOD is actually the global zone SYSMOD entry. From that entry, SMP/E obtains information about the relationship between this SYSMOD and other SYSMODs. SMP/E then uses other global zone SYSMOD entries and the distribution zone SYSMOD entries to resolve these relationships. In addition, SMP/E checks the target zone SYSMOD entries to ensure that the selected SYSMODs have been applied.

After the eligible SYSMODs have been determined, the various MCS statements lead SMP/E to the other distribution zone entries:
- ++MAC and ++MACUPD statements lead to distribution zone MAC entries.
- ++SRC and ++SRCUPD statements lead to distribution zone SRC entries.
- ++MOD and ++ZAP statements lead to distribution zone MOD entries.
- Data element statements lead to distribution zone data element entries.
- ++hfs_element statements lead to distribution zone hierarchical file system element entries.

If the SYSMOD replaces or modifies a macro (++MAC or ++MACUPD in SYSMOD), SMP/E uses the distribution zone MAC entry to determine the functional and service level of the macro and to determine whether any assemblies must be redone as a result of the macro update. This is done by checking the GENASM subentries in the target zone MAC entry. The names in the list can be names of either target zone ASSEM or SRC entries. In either case, SMP/E accesses the appropriate entry and uses the information stored there to perform the required assemblies.

If the SYSMOD replaces or modifies a source (++SRC or ++SRCUPD in SYSMOD), SMP/E uses the distribution zone SRC entry to determine the functional and service level of the source and the library containing the source code, and then performs the required assembly.

After an assembly has been done (with data from either the ASSEM entry or the SRC entry), SMP/E knows that the resulting object deck may be link-edited into the distribution libraries somewhere. Although there is no direct connection (that is, no subentry value present) between the ASSEM or SRC entries and a MOD entry, there is an implicit relationship. SMP/E assumes that for each ASSEM and SRC entry there exists a MOD entry with the same name. Thus, after performing the assembly, SMP/E accesses the corresponding MOD entry to determine where to install the object deck. This leads us to the same point as if a MOD were supplied in the SYSMOD.

If the SYSMOD replaces or modifies a module (++MOD or ++ZAP in SYSMOD), or if an assembly was done for a SRC or ASSEM entry, SMP/E uses the distribution zone MOD entry to determine the functional and service level of the module, the DLIB into which it must be linked, the load modules into which the module should be linked, and the link-edit attributes that should be used. Information about load modules is kept in the LMOD subentries in the MOD entry.

The LMOD subentries within the MOD entry lead SMP/E to the distribution zone LMOD entries. These represent load modules that exist in the target libraries. LMOD entries contain all the information necessary, either to relink the load module or to update it using IMASPZAP.
If the SYSMOD replaces a data element (a data element MCS is in the SYSMOD), SMP/E uses the distribution zone **data element entry** to determine the functional and service level of the data element. It then gets the data element installed in the appropriate distribution library.

If the SYSMOD replaces a hierarchical file system element (a ++hfs_element MCS is in the SYSMOD), SMP/E uses the distribution zone **hierarchical file system element entry** to determine the functional and service level of the hierarchical file system element. It then gets the hierarchical file system element installed in the appropriate distribution library.

After all the updating, assembling, linking, and so on, are done, you arrive back almost at the starting point, the SYSMOD entry. The difference is that this is the distribution zone SYSMOD entry, which contains all the information about what was done as a result of installing this SYSMOD.

**Figure 10 on page 182** shows how the various SMP/E status and structure entries are related. Subsequent sections of this chapter provide a more detailed explanation of each entry and the data it contains.
Note: Lines to the left of the figure are implicit connections based on the assumption that entries with equal names will be found.

The remaining sections of this chapter deal with each of the specific entries in the various SMP/E data sets.
The ASSEM entry contains assembler statements that can be assembled to create an object module. It is created during JCLIN processing when SMP/E encounters an assembler step with inline assembler input. When the module is reassembled using the statements in the ASSEM entry, SMP/E copies those statements into the SMPWRK2 data set, and then assembles the module.

If a macro is invoked in the assembly, the ASSEM entry is pointed to by the GENASM subentry in the MAC entry created for that macro. As a result, when that macro is updated, SMP/E can reassemble the affected module using the statements in the ASSEM entry. For additional information, see the “Processing” section in the JCLIN command chapter in *SMP/E Commands*.

### Subentries

These are the subentries for the ASSEM entry as they appear in the LIST output:

- **name**
  - is the name of the ASSEM entry. It corresponds to the name of the module that can be reassembled by use of that ASSEM entry.
  - The name can contain from 1 to 8 alphanumeric characters.

- **ASSEMBLER INPUT**
  - is the actual assembler statements that were saved for this module during JCLIN processing. These statements are passed to the assembler whenever this module must be reassembled.
  - The UCL operands are **++ASMIN** and **++ENDASMIN**.
  - An ASSEM entry must contain at least the **++ASMIN** and **++ENDASMIN** statements, plus the associated assembler statements.
  - The **++ASMIN** and **++ENDASMIN** statements must start in column 1.
  - No other operands can start on the same line as the **++ASMIN** statement.
  - If you specify the **++ASMIN** statement, you must also specify the **++ENDASMIN** statement.

- **LASTUPD**
  - identifies the cause of the last change to this ASSEM entry.
  - The UCL operand is **LASTUPD**(value). This subentry can contain one of the following values:
    - **JCLIN** indicates that the change was made during JCLIN command processing.
    - **UCLIN** indicates that the change was made as a result of UCLIN processing.
    - **sysmod_id** indicates that the change was made during the installation of the indicated SYSMOD.
      - The SYSMOD ID must contain 7 alphanumeric characters.

- **LASTUPD TYPE**
  - indicates how the entry was last changed.
  - The UCL operand is **LASTUPDTYPE**(value). This subentry can contain one of the following values:
    - **ADD** The entry was added.
    - **UPD** The entry was updated.
ASSEM entry (distribution and target zone)

LIST Examples

To list all the ASSEM entries in a particular zone, you can use the following commands:

```plaintext
SET BDY(TGT1) /* Set to requested zone. */.
LIST ASSEM /* List all ASSEM entries. */.
```

To list specific ASSEM entries, you can use these commands:

```plaintext
SET BDY(TGT1) /* Set to requested zone. */.
LIST ASSEM(ASSEM01 /* List only these two */
ASSEM02) /* entries. */.
```

The format of the LIST output for each ASSEM entry is the same for both of these commands. The only difference is the number of ASSEM entries listed. Figure 11 is an example of LIST output for ASSEM entries.

<table>
<thead>
<tr>
<th>NAME</th>
<th>LASTUPD</th>
<th>ASSEMBLER INPUT</th>
</tr>
</thead>
</table>
| ASSEM01| JXY1102  | ... assembler statements...
| ASSEM02| JXY1121  | ... assembler statements...

Figure 11. ASSEM entry: sample LIST output

You can use the LIST command to find the MAC entries for macros that are called by the assembler statements in this ASSEM entry. To include the names of these MAC entries in the LIST output, you can use the XREF operand, as shown in these commands:

```plaintext
SET BDY(TGT1) /* Set to requested zone. */.
LIST ASSEM /* List all ASSEM entries */
XREF /* and macros that use them. */.
```

Notes:

1. You can use XREF in either mass mode or select mode.
2. SMP/E obtains the data included for the XREF operand by checking the GENASM subentries in all the MAC entries. Because this data is not contained in the ASSEM entry itself, you cannot use UCLIN to change it in the ASSEM entry.

Figure 12 on page 185 is an example of the LIST output produced when the XREF operand is used.
UNLOAD Examples

To dump the ASSEM entries in UCL format, you can use the UNLOAD command. To unload all the ASSEM entries in a particular zone, you can use the following commands:

```
SET BDY(TGT1) /* Set to requested zone. */.
UNLOAD ASSEM /* Unload all ASSEM entries. */.
```

To unload specific ASSEM entries, you can use these commands:

```
SET BDY(TGT1) /* Set to requested zone. */.
UNLOAD ASSEM(ASSEM01) /* Unload only these two */
ASSEM02) /* entries. */.
```

The format of the UNLOAD output for each ASSEM entry is the same for both of these commands. The only difference is the number of ASSEM entries listed. Figure 13 on page 186 is an example of UNLOAD output for ASSEM entries.
ASSEM entry (distribution and target zone)

**UCLIN Examples**

You can use the ADD, REP, and DEL UCL statements to change subentries in the ASSEM entry. When you use UCLIN to update an ASSEM entry, keep these points in mind:

- After the UCLIN changes are done, the ASSEM entry must contain at least ++ASMIN and ++ENDASMIN statements, plus the associated assembler input. Otherwise, there is not enough information in the entry to assemble the associated module.

- The input following the ++ASMIN statement replaces the existing assembler input in the ASSEM entry.

- If a DEL statement deletes all the existing subentries in the entry, SMP/E deletes the entire entry.

- When SMP/E processes a DEL statement, it does not compare any assembler input after the ++ASMIN statement with the input that is currently in the ASSEM entry. It just deletes the existing assembler input. This causes an error, because there is now insufficient data in the ASSEM entry.
ASSEM entry (distribution and target zone)

The following examples are provided to help you use the ASSEM entry.

Example 1: Deleting an ASSEM entry
The main use of UCLIN for an ASSEM entry is to delete the entry. Here is an example:

```plaintext
SET   BDY(TGT1) /* Set to target zone. */.
UCLIN /* */.
DEL   ASSEM01 /* Delete the entry. */.
ENDUCL /* */.
```

Example 2: Adding a new ASSEM entry
To create an ASSEM entry, you should run the JCLIN command rather than use UCLIN. (For examples of how JCLIN creates ASSEM entries, see SMP/E Commands. However, you can also use UCLIN to create ASSEM entries. For example, you can use the following commands to create a new ASSEM entry and to update an existing MAC entry to show that the macro is used in the new assembly:

```plaintext
SET   BDY(TGT1) /* Set to target zone. */.
UCLIN /* */.
ADD   ASSEM099 /* New ASSEM entry. */.
++ASMIN /* Assembler data. */.
ASSEM099 CSECT /* */.
MAC99  0,0,1 /* */.
END ASSEM099 /* */.
++ENDASMIN /* End of assembler data. */.
ADD   MAC(MAC99) /* Modify macro entry used */.
      GENASM(ASSEM099) /* to indicate use in new */.
      /* assembly entry. */.
ENDUCL /* */.
```
BACKUP entries (SMPSCDS)

BACKUP entries are a collection of target zone entries that are copied to the SMPSCDS during APPLY processing before they are updated by inline JCLIN, a \texttt{++MOVE} MCS, or a \texttt{++RENAME} MCS, or before they are deleted by an element MCS with the DELETE operand. A BACKUP entry is also created for a MOD entry if a SYSMOD being installed provides a \texttt{++MOD} statement that either changes the distribution library for the module or adds the module to an existing load module.

- As SMP/E processes the inline JCLIN for a given SYSMOD, it determines which entries will be affected by that JCLIN. Before making the changes, it saves a copy of each of those entries on the SMPSCDS.
- Likewise, as SMP/E processes the \texttt{++MOVE}, \texttt{++RENAME}, and element MCSs in a given SYSMOD, it determines which entries will be updated or deleted. Before updating or deleting the entries, it saves a copy of each of those entries on the SMPSCDS.

Besides saving copies of the affected entries, SMP/E also saves a SYSMOD entry on the SMPSCDS to indicate which entries were added by JCLIN, updated by JCLIN or an MCS statement in the SYSMOD, or deleted by an MCS statement in the SYSMOD. Each entry is associated with only one SYSMOD. The entries associated with a SYSMOD are called the BACKUP entries for that SYSMOD.

BACKUP entries consist of:

- A SYSMOD entry indicating what entries were added, deleted, or updated
- ASSEM entries for updated target zone ASSEM entries
- JAR entries for deleted JAR entries
- LMOD entries for updated target zone LMOD entries
- MAC entries for updated or deleted target zone MAC entries
- MOD entries for updated or deleted target zone MOD entries
- SRC entries for updated or deleted target zone SRC entries
- Data element entries for deleted target zone data element entries
- Hierarchical file system entries for deleted target zone hierarchical file system entries
- DLIB entries for updated target zone DLIB entries

SMP/E provides access to the BACKUP entries as a group—for example, through the LIST command—but it does not provide access to the individual entries or subentries.

Subentries

The subentries in the BACKUP entries are the same as those in the various entry types that are copied. For more information, see the section in this chapter describing each entry.

LIST Examples

To list the BACKUP entries for all the SYSMODs in the SMPSCDS, you can use the following commands:

\begin{verbatim}
SET BDY(TGT1) /* Set to requested tgt zone. */.
LIST BACKUP /* List all BACKUP entries. */.
\end{verbatim}

To list the BACKUP entries for a specific SYSMOD, you can use these commands:

\begin{verbatim}
SET BDY(TGT1) /* Set to requested tgt zone. */.
LIST BACKUP(UZI2345, UZI2346) /* List only BACKUP entries */.
\end{verbatim}
The format of the LIST output for each group of BACKUP entries is the same for both of these commands. The only difference is the number of SYSMODs for which BACKUP entries are listed.

When you list the BACKUP entries for a SYSMOD, the first entry in the output is a summary entry for the SYSMOD, which indicates the date and time the SYSMOD was applied, as well as which entries were added or updated as a result of applying the SYSMOD. This is followed by a listing of all the existing target zone entries affected by this SYSMOD, before they were updated. Nothing is listed for added entries, because no entry existed before the SYSMOD was installed.

Figure 14 on page 190 is a partial example of LIST output for BACKUP entries. It shows the summary records, but not all the backup copies of the entries modified by the SYSMOD. This is because the format of those copies is the same as the format for the original target zone entries.

UCLIN Examples

You can use the DEL UCL statement to delete BACKUP entries from the SMPSCDS. This can be helpful if you plan to do an APPLY followed by ACCEPT when several target libraries have been created from the same distribution library.

When a SYSMOD is accepted into a distribution zone, the entries associated with the SYSMOD are automatically deleted from the SMPSCDS for the RELATED target zone. However, even if the SYSMOD was also applied to other target zones created from the same distribution zone, SMP/E does not clean up the SMPSCDS data sets for the other target zones.

To delete the entries from these data sets, you can accept the SYSMOD and name these other target zones as the RELATED zone. However, this updates the distribution library each time, which is time-consuming and can use up space in the distribution library data set.

Instead, you can use the DEL command to delete these entries without updating the distribution library. To determine which entries to specify, check the SMPLOG data set to see which ones SMP/E deleted during ACCEPT processing.

Note: You can also use the CLEANUP command to delete BACKUP entries without specifying them individually. For more information about the CLEANUP command, see SMP/E Commands.
BACKUP Entries (SMPSCDS)

Example: Deleting BACKUP entries

Assume BACKUP entries exist on the SCDS for SYSMODs UZ12345 and UZ12346. Those SYSMODs have been accepted, but the BACKUP entries have not been deleted from the SCDS. This can happen when multiple systems are supported off one set of DLIBs, where you accept from only one of the target systems. Here is an example of UCL you could use to delete the BACKUP entries from the SCDS:

```
SET BDY(TGT1) /* Set to TGT1 zone. */.
UCLIN /* */.
DEL BACKUP(UZ12345) /* Delete the BACKUP entries. */.
DEL BACKUP(UZ12346) /* Delete the BACKUP entries. */.
ENDUCL /* */.
```

Note: You must specify a separate UCL statement for each BACKUP entry to be deleted.

Figure 14. BACKUP entries: sample LIST output
The data element entry describes an element that is not a macro, module, or source (for example, a CLIST or a sample procedure). Data elements may exist in distribution or target libraries. A data element entry is created the first time you install a SYSMOD that contains an MCS for a data element that does not yet have an entry in the CSI data set.

SMP/E records the function and service level of the data element in the entry. Once a data element entry exists, it is updated as subsequent SYSMODs that affect the data element are installed.

Table 2 on page 10 shows the types of entries used for data elements. Some types of elements, such as panels, messages, or text, may have been translated into several languages. In these cases, the entry type contains \[xxx\], which represents the language used for the element. (If an element was not translated, the entry type does not contain any \[xxx\] value.) Table 3 on page 13 shows the \[xxx\] values and the languages they represent.

Subentries

These are the subentries for the data element entry as they appear in the LIST output:

**name**

is the name of the data element represented by the entry. It can contain from 1 to 8 alphanumeric characters and $, #, @, or hex C0.

**ALIAS**

specifies a list of alias names for the element. The UCL operand is ALIAS(name...).

Each alias name can contain from 1 to 8 alphanumeric characters.

**DISTLIB**

specifies the ddname of the distribution library for the data element. The UCL operand is DISTLIB(ddname).

- The ddname can contain from 1 to 8 alphanumeric characters.
- The DISTLIB subentry is required. Without it, SMP/E cannot process any changes for the data element.

**FMID**

specifies the functional owner of this data element. The functional owner is the last function SYSMOD that replaced this element. The UCL operand is FMID(sysmod_id).

The SYSMOD ID must contain 7 alphanumeric characters.

**LASTUPD**

identifies the cause of the last change to this data element entry. The UCL operand is LASTUPD(value). This subentry can contain one of the following values:

**UCLIN**

indicates that the change was made as a result of UCLIN processing.

**sysmod_id**

indicates that the change was made during the installation of the indicated SYSMOD.
Data element entry (distribution and target zone)

The SYSMOD ID must contain 7 alphanumeric characters.

LASTUPD TYPE
indicates how the entry was last changed.

The UCL operand is LASTUPDTYPE(value). This subentry may contain one of the following values:
- ADD The entry was added.
- UPD The entry was updated.

RMID
identifies the last SYSMOD that replaced this data element. Any subsequent SYSMOD that modifies this data element must have a defined relationship (such as PRE or SUP) with this SYSMOD.

The UCL operand is RMID(sysmod_id).
- The SYSMOD ID must contain 7 alphanumeric characters.
- If RMID is not specified but FMID is, SMP/E sets the RMID value to the specified FMID.

SYSLIB
specifies the ddname of the target library for the data element.

The UCL operand is SYSLIB(ddname).
- You can specify only one SYSLIB value.
- The ddname can contain from 1 to 8 alphanumeric characters.

LIST Examples
To list all the data element entries of a given type (such as CLIST) in a particular zone, you can use the following commands:

```
SET BDY(TGT1) /* Set to requested zone. */.
LIST CLIST /* List all CLIST entries. */.
```

To list specific CLIST entries, you can use these commands:

```
SET BDY(TGT1) /* Set to requested zone. */.
LIST CLIST(CLIST1 /* List only these two */
  CLIST2) /* entries. */.
```

The format of the LIST output for each data element entry is the same for both of these commands. The only difference is the number of data element entries listed.

Figure 15 is an example of LIST output for data element entries.
By specifying the FORFMID operand, you can reduce the number of data element entries listed. When FORFMID is specified, SMP/E lists a data element entry only if its FMID matches one of the FMIDs specified on the FORFMID operand. For example, to list CLIST entries whose FMIDs are defined in FMIDSET TP or else are MYCLST1, you can use these commands:

```
SET BDY(TGT1) /* Set to target zone. */
LIST CLIST /* List all CLIST entries */
   FORFMID(TP /* for the TP FMIDSET */
            MYCLST1 /* and FMID MYCLST1. */.
```

You can use the LIST command to find out the names of all SYSMODs that have modified data elements. To include the names of these SYSMODs in the LIST output, you can use the XREF operand, as shown in these commands:

```
SET BDY(TGT1) /* Set to requested zone. */.
LIST CLIST /* List all CLIST entries */
   XREF /* and related SYSMODs. */.
```

Notes:
1. You can use XREF in either mass mode or select mode.
2. SMP/E obtains the data included for the XREF operand by checking for entries for this data element in all the SYSMOD entries. Because this data is not contained in the data element entry itself, you cannot use UCLIN to change it in the data element entry.

Figure 16 on page 194 is an example of the LIST output produced when the XREF operand is used.
UNLOAD Examples

To dump the data element entries in UCL format, you can use the UNLOAD command. To unload all the CLIST entries in a particular zone, you can use the following commands:

```
SET BDY(TGT1) /* Set to requested zone. */.
UNLOAD CLIST /* Unload all CLIST entries. */.
```

To unload specific CLIST entries, you can use these commands:

```
SET BDY(TGT1) /* Set to requested zone. */.
UNLOAD CLIST(CLIST1 CLIST2) /* Unload only these two entries. */.
```

The format of the UNLOAD output for each data element entry is the same for both of these commands. The only difference is the number of data element entries listed. Figure 17 on page 195 is an example of UNLOAD output for CLIST entries.
By specifying the FORFMID operand, you can reduce the number of data element entries unloaded. When FORFMID is specified, SMP/E unloads a data element entry only if its FMID matches one of the FMIDs specified on the FORFMID operand. For example, to unload CLIST entries whose FMIDs are either defined in FMIDSET TP or are MYCLST1, you can use these commands:

```c
SET BDY(TGT1) /* Set to target zone. */.
UNLOAD CLIST /* Unload all CLIST entries */.
FORFMID(TP /* for the TP FMIDSET */
    MYCLST1) /* and FMID MYCLST1. */.
```

### UCLIN Examples

You can use the ADD, REP, and DEL UCL statements to change subentries in the data element entry. After the UCLIN changes are made, the data element entry must contain at least the following subentries:

- DISTLIB
- FMID
- RMID

Otherwise, there is not enough information in the entry to process the data element. If any of these subentries are missing, SMP/E does not make the requested UCL updates to the entry, and the entry remains as it was before the UCL command.

#### Example: Adding a new data element entry

Assume you have installed CLIST3 outside of SMP/E, but now you want to start using SMP/E to track changes to that CLIST. Here is an example of the UCL statements you would use to define entries for that CLIST in the appropriate target and distribution zones:

```c
SET BDY(TGT1) /* Set to target zone. */.
UCLIN /* */.
ADD CLIST(CLIST3) /* Define new CLIST entry. */
    DISTLIB(AMACLIB) /* Define DLIB. */
    SYSLIB(ISPCLIB) /* System library. */
    FMID(MYCLST1) /* Functional owner. */
    RMID(MYCLST1) /* */.
ENDUCL /* */.
SET BDY(DLB1) /* Now do same to DLIB. */.
UCLIN /* */.
ADD CLIST(CLIST3) /* Define new CLIST entry. */
    DISTLIB(AMACLIB) /* Define DLIB. */
    SYSLIB(ISPCLIB) /* System library. */
    FMID(MYCLST1) /* Functional owner. */
    RMID(MYCLST1) /* */.
ENDUCL /* */.
```

---

**Figure 17. Data element entry: sample UNLOAD output**

Data element entry (distribution and target zone)
Data element entry (distribution and target zone)

/* no SYSLIB info in DLIB. */
FMID(MYCLST1)  /* Functional owner. */.
ENDDCL  /* */.
DDDEF entry (distribution, target, and global zone)

The DDDEF entry contains the information SMP/E needs to dynamically allocate a specific data set. With DDDEF entries, you do not have to provide a DD statement for every data set SMP/E may need to process a particular command. When SMP/E determines that it needs a specific data set, it looks for a DD statement that it can use to allocate that data set. If there is no DD statement, SMP/E checks whether the current zone contains a DDDEF entry for that data set. If so, it uses the information in the DDDEF entry to dynamically allocate the data set.

For more information about dynamically allocating data sets, see the SMP/E User’s Guide.

Notes:
1. In a job with multiple SET commands, if you use DDDEF entries that specify SYSOUT for SMP/E output (such as SMPOUT or SMPRPT), SMP/E produces multiple SYSOUT data sets. This can cause undesirable results; for example, the output could appear to be out of sequence from one SET command to the next. Therefore, when you run such a job, you may prefer to use DD statements instead of DDDEF entries for SMP/E output data sets.
2. SMP/E does not strictly enforce rules for which subentries you can specify in DDDEF entries for specific data sets. To prevent possible allocation errors, refer to the z/OS MVS JCL User’s Guide.

Subentries

These are the subentries for the DDDEF entry as they appear in the LIST output: name

- is the ddname of the data set to be allocated.
  - The name can contain from 1 to 8 alphabetic (A through Z), national (@, #, or $), or numeric (0 through 9) characters. The first character must be alphabetic or national.
  - Other than checking for duplicate DDDEF names in a given zone, SMP/E does not check whether the specified name is associated with another data set. For example, SMP/E does not check whether a DDDEF name is the same as a zone name. You must do this checking yourself to avoid undesired results.

BLK(size), CYL, or TRK

- specifies the space units to be used in allocating the data set: blocks, cylinders, or tracks.
  - The UCL operand is BLOCK(size), CYLINDERS, or TRACKS.
  - size is the size, in decimal, of each block to be allocated. To specify the number of blocks to be allocated, use SPACE.
  - You can specify either the long form or the short form of these operands:
    - BLOCK or BLK
    - CYLINDERS or CYL
    - TRACKS or TRK
  - These operands are mutually exclusive with each other and with OLD, MOD, SHR, CONCAT, PATH, and SYSOUT.

CATALOG, DELETE, or KEEP

- specifies the final disposition (DISP) of the data set.
The UCL operand is **CATALOG, DELETE, or KEEP**.

- These operands are mutually exclusive with each other and with CONCAT and PATH.
- You cannot specify a final disposition for SMPTLIB data sets. SMP/E automatically specifies a final disposition based on the command being processed. For more information, see "SMPTLIB" on page 161.

**CONCAT** identifies one or more DDDEF entries, **existing in the same zone**, that should be concatenated during SMP/E processing.

The UCL operand is **CONCAT(name...)**.

- The DDDEF names can contain from 1 to 8 alphabetic (A through Z), national (@, #, or $), or numeric (0 through 9) characters. The first character must be alphabetic or national.
- There must be a DDDEF entry for each specified name. SMP/E does not check other sources (such as DD statements or GIMDDALC control statements in SMPPARM member GIMDDALC) to get the information needed to allocate the data sets.
- SMP/E allows you to specify up to 123 names. However, the actual number of partitioned data sets that can be concatenated depends on the operating system you are running under. To determine the maximum number of data sets you can concatenate, see "z/OS DFSMS Using Data Sets, SC26-7410".
- CONCAT is required for concatenated data sets.
- CONCAT cannot specify any DDDEF entries that contain the CONCAT operand. (You cannot nest concatenated DDDEF entries.)
- CONCAT is mutually exclusive with all other operands.

**Note:** When a DDDEF that specifies a disposition of OLD is allocated as part of a CONCAT, a disposition of SHR is used for the allocation.

**DATACLAS** specifies the name of a data class to be used for allocating a new data set managed by SMS.

The UCL operand is **DATACLAS(name)**.

- The data class name is defined by the storage administrator at your installation. This value can contain from 1 to 8 alphanumeric characters (A through Z and 0 through 9) or national characters (@, #, $) and must start with either an alphabetic or national character.
- You can specify this operand only if SMS manages your storage.
- Specify this operand only when you are allocating a new data set. SMS ignores this parameter if it is specified for an existing data set.
- This operand is mutually exclusive with PATH and CONCAT.

**DATASET** is the name of the data set to be allocated.

The UCL operand is **DATASET(dsname)**.

- You can specify either **DATASET** or **DA**.
- The data set name must conform to standard naming conventions for data sets. Each part of the name must contain from 1 to 8 characters, separated from the other parts by a period (.), with no intervening blanks. The maximum length of the entire name is 44 characters (including the periods).
- The data set name itself cannot contain parentheses.
To define a dummy data set, omit the DATASET operand and specify an initial DISP of NEW.

DATASET is required if OLD, SHR, or MOD is specified.

DATASET is mutually exclusive with CONCAT, PATH, and SYSOUT.

You cannot use DATASET to specify the data set name of an SMPTLIB data set. Instead, you can use the DSPREFIX operand in either the SMPTLIB DDDEF entry or in the OPTIONS entry used to process those data sets.

SMP/E does not check whether the specified data set name is unique within the zone. For example, SMP/E does not check whether the data set name was also defined in another DDDEF entry in the same zone, or whether the data set name defines the CSI data set containing the zone. You must do this checking yourself to avoid undesired results.

**DIR**

specifies the number of directory blocks to allocate.

The UCL operand is `DIR(nn)`. The number specified can contain from 1 to 4 decimal digits.

DIR is mutually exclusive with OLD, MOD, SHR, CONCAT, and PATH.

**DSNTYPE**

specifies the type of partitioned data set to be created.

The UCL operand is `DSNTYPE(LIBRARY)` or `DSNTYPE(PDS)`.

**LIBRARY**

specifies that a PDSE (which must be an SMS-managed data set) is to be created.

**PDS**

specifies that a PDS is to be created.

You can also specify DSNTYPE in either a data class or a member of SYS1.PARMLIB.

Notes:

1. This operand is mutually exclusive with PATH and CONCAT.
2. When SMP/E RECEIVE processing allocates a new SMPTLIB data set, it uses the original DSNTYPE of the corresponding RELFILE data set. If SMP/E cannot determine the original DSNTYPE of the corresponding RELFILE data set, SMP/E uses the DSNTYPE value specified in the SMPTLIB DDDEF entry.

**DSPREFIX**

specifies the data set prefix to be used to construct the full data set name for SMPTLIB data sets. For more information about names for SMPTLIB data sets, see [SMP/E Commands](#).

The UCL operand is `DSPREFIX(prefix)`.

- The prefix can contain from 1 to 26 alphanumeric characters.
- The prefix must follow standard conventions for naming data sets.

Instead of specifying DSPREFIX in the SMPTLIB DDDEF entry, you can specify it in the OPTIONS entry that is in effect when you receive RELFILEs into the SMPTLIB data sets.

If you do not specify a data set prefix in the SMPTLIB DDDEF entry or in the appropriate OPTIONS entry, no prefix is included when SMP/E assigns a name to the SMPTLIB data sets.

DSPREFIX is mutually exclusive with CONCAT and PATH.
DDDEF entry (distribution, target, and global zone)

Note: This subentry exists only in the global zone.

**MGMTCLAS**
specifies the name of a management class to be used for allocating a new data set managed by SMS.

The UCL operand is **MGMTCLAS**(name).
- The management class name is defined by the storage administrator at your installation. This value can contain from 1 to 8 alphanumeric characters (A through Z and 0 through 9) or national characters (@, #, $) and must start with either an alphabetic or national character.
- You can specify this operand only if SMS manages your storage.
- Specify this operand only when you are allocating a new data set. SMS ignores this parameter if it is specified for an existing data set.
- This operand is mutually exclusive with PATH and CONCAT.

**MOD, NEW, OLD, or SHR**
specifies the initial disposition (DISP) of the data set.

The UCL operand is **MOD, NEW, OLD, or SHR**.
- These operands are mutually exclusive with each other and with CONCAT and PATH.
- OLD, SHR, and MOD are also mutually exclusive with SYSOUT, PROTECT, BLOCK, CYLINDERS, TRACKS, DIR, and SPACE.
- You cannot specify an initial disposition for SMPTLIB data sets. SMP/E automatically specifies an initial disposition, which is based on the command being processed. For more information, see “SMPTLIB” on page 161.

**PATH**
identifies the name of the path to be allocated in a UNIX file system. The name is not a complete pathname; it is a directory. This value is concatenated with the appropriate element or load module name to create a complete pathname.

The UCL operand is **PATH**(pathname).
- PATH is mutually exclusive with all other DDDEF entry operands.
- The pathname can be from 1 to 255 characters.
- The pathname must begin and end with a slash (/).
- In addition to the required delimiters (/), a pathname must also be enclosed in single apostrophes (’) if any of the following is true:
  - The pathname contains lowercase alphabetic characters.
  - The pathname contains a character that is not uppercase alphabetic, numeric, national ($, #, or @), slash (/), plus (+), hyphen, period, or ampersand (&).
  - The pathname spans more than one line in the UCL control statement.
  - The apostrophes must be outside the required delimiters, as in ’/pathname/’, not ’/pathname’.
  - The single apostrophes used to enclose a pathname (the delimiters) do not count as part of the 255-character limit.
- Any apostrophes specified as part of a pathname (not the delimiters) must be doubled.
  - Double apostrophes count as two characters in the 255-character limit.
- The pathname can include characters X'40' through X'FE'.

161
DDDEF entry (distribution, target, and global zone)

- Because symbolic substitution is not detected by SMP/E, it is not supported by SMP/E.

**PROTECT**

specifies that the z/OS SecureWay™ Security Server PROTECT option should be used when a new data set is first allocated. If **PROTECT** is specified and z/OS SecureWay Security Server is installed, the data set allocated by SMP/E will be protected by z/OS SecureWay Security Server.

You can also use PROTECT to indicate that an existing data set is protected by z/OS SecureWay Security Server. In this case, because SMP/E does not allocate the data set, it does not check the PROTECT indicator. However, you can use it to keep a record of which data sets have been protected with z/OS SecureWay Security Server.

The UCL operand is **PROTECT**.

- **PROTECT** is mutually exclusive with CONCAT, PATH, OLD, MOD, and SHR.

**SPACE**

specifies the primary and secondary space allocation for new data sets.

The UCL operand is **SPACE(prime,second)**.

- Each value must contain from 1 to 4 decimal digits, and the two values must be separated by a comma or a blank.
- **SPACE** is mutually exclusive with CONCAT, PATH, OLD, MOD, and SHR.

**STORCLAS**

specifies the name of a storage class used for allocating a new data set managed by SMS.

The UCL operand is **STORCLAS(name)**.

- The storage class name is defined by the storage administrator at your installation. This value can contain from 1 to 8 alphanumeric characters (A through Z and 0 through 9) or national characters (@, #, $) and must start with either an alphabetic or national character.
- You can specify this operand only if SMS manages your storage.
- Specify this operand only when you are allocating a new data set. SMS ignores this parameter if it is specified for an existing data set.
- This operand is mutually exclusive with PATH and CONCAT.

**SYSOUT**

specifies the output class for SYSOUT data sets.

The UCL operand is **SYSOUT(value)**.

- If you specify a class, it must be 1 alphabetic or numeric character (A through Z or 0 through 9).
- If you specify *, the output class depends on references to an OUTPUT JCL statement.
  - If there is an implicit or explicit reference to an OUTPUT JCL statement, the output is written to the same class as the CLASS parameter on the OUTPUT statement.
  - If there is no reference to an OUTPUT JCL statement, the output is written to the same class as the one specified as MSGCLASS on the JOB card.
SYSOUT is mutually exclusive with CATALOG, DELETE, KEEP, DIR, 
DSNTYPE, UNIT, SPACE, VOLUME, BLOCK, CYLINDER, TRACK, 
CONCAT, PATH, and DATASET.

You cannot specify SYSOUT for SMPTLIB data sets.

UNIT
specifies the UNIT type the data set resides on if it is not cataloged.

The UCL operand is UNIT(type).

- If the data set is not cataloged, you must specify UNIT (unless it is not 
cataloged because of SMS). If you specify UNIT for a cataloged data set, it 
overrides the value in the catalog.
- The UNIT value can contain from 1 to 8 characters and should conform to 
standard UNIT naming conventions.
  SMP/E accepts any nonblank characters specified between the open and 
close parentheses, up to a maximum length of 8.
- UNIT is mutually exclusive with CONCAT and PATH.

VOLUME
specifies the volume serial number of the volume that the data set resides on if 
not cataloged.

The UCL operand is VOLUME(volid...).

- If the data set is not cataloged, you must specify VOLUME (unless it is not 
cataloged because of SMS). If you specify VOLUME for a cataloged data set, it 
overrides the value in the catalog.
- The volume identifier can contain from 1 to 6 alphanumeric characters.
- For SMPTLIB data sets, you can specify up to five volume serial numbers.
  All the volumes must have the same UNIT type. For other data sets, you can 
specify only one volume serial number.
- VOLUME is mutually exclusive with CONCAT and PATH.

WAIT=YES or WAIT= NO
indicates whether SMP/E should wait for the data set to be allocated if the 
volume is not mounted or if the data set is already in use. Not waiting causes 
allocation to fail for the data set.

The UCL operand is WAITFORDSN.

- You can specify either WAITFORDSN or WAIT.
- WAIT is mutually exclusive with CONCAT and PATH.
- If you do not specify a value, the default is not to wait.
- WAIT is not related to the PROCESS parameter specified on the EXEC 
statement. PROCESS affects how long a job should wait for a data set before 
being run. For more information, see Chapter 10, “JCL statements required to 
invoke SMP/E.”

LIST Examples
To list all the DDDEF entries in a particular zone, you can use the following 
commands:

```
SET BDY(TGT1) /* Set to requested target. */.
LIST DDDEF /* List all DDDEF entries. */.
```

To list specific DDDEF entries in a particular zone, you can use these commands:
### DDDEF entry (distribution, target, and global zone)

```plaintext
SET BDY(TGT1) /* Set to requested target. */.
LIST DDDEF(SMPMTS /* List only these three */
  MACLIB /* entries. */
  SYSLIB) /* */.
```

The format of the LIST output for each DDDEF entry is the same for both of these commands. The only difference is the number of DDDEF entries listed.

Figure 18 and Figure 19 on page 204 are examples of LIST output for DDDEF entries.

<table>
<thead>
<tr>
<th>NAME</th>
<th>DATASET</th>
<th>VOLUME</th>
<th>UNIT</th>
<th>SHR</th>
<th>PATH</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMACLIB</td>
<td>= SYS1.AMACLIB</td>
<td>DLIB01</td>
<td>3380</td>
<td></td>
<td>'/path_name1/'</td>
</tr>
<tr>
<td>BPXLIB1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMACLIB</td>
<td>= SYS1.MACLIB</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MACLIB</td>
<td>= SYS1.MACLIB</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SMPMTS</td>
<td>= SYS1.SMPMTS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SMPLOG</td>
<td>= SYS1.SMPLOG</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYSLIB</td>
<td></td>
<td>SMPMTS</td>
<td>CMACLIB</td>
<td>AMACLIB</td>
<td></td>
</tr>
<tr>
<td>SMPOUT</td>
<td>SYSOUT</td>
<td>A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SMPWRK1</td>
<td>UNIT</td>
<td>SYSDA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SPACE</td>
<td>(25,25)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DIR</td>
<td>25</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ALLOC</td>
<td>TRK</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>NEW</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DELETE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SMPWRK2</td>
<td>DATACLAS</td>
<td>FB80CLAS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MGMTCLAS</td>
<td>SMPEMCLS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>STORCLAS</td>
<td>SMPESCLS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SMPTLIB</td>
<td>VOLUME</td>
<td>DLIB01</td>
<td>DLIB02</td>
<td>DLIB03</td>
<td></td>
</tr>
<tr>
<td></td>
<td>UNIT</td>
<td>3380</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Figure 18. DDDEF entry: sample LIST output for a target zone
**UNLOAD Examples**

To dump the DDDEF entries in UCL format, you can use the UNLOAD command. To unload all the DDDEF entries in a particular zone, you can use the following commands:

```
SET BDY(TGT1) /* Set to requested target. */.
UNLOAD DDDEF /* Unload all DDDEF entries. */.
```

To unload specific DDDEF entries in a particular zone, you can use these commands:

```
SET BDY(TGT1) /* Set to requested target. */.
UNLOAD DDDEF(SMPMTS /* Unload only these three */.
MACLIB /* entries. */.
SYSLIB) /* */.
```

The format of the UNLOAD output for each DDDEF entry is the same for both of these commands. The only difference is the number of DDDEF entries unloaded.

**Note:** You can use the UNLOAD command only for target and distribution zones, not for the global zone.

*Figure 20 on page 205* is an example of UNLOAD output for DDDEF entries.
DDDEF entry (distribution, target, and global zone)

```plaintext
UCLIN.
REP DDDEF ( AMACLIB )
  DATASET ( SYS1.AMACLIB )
  VOLUME ( DLIB01 )
  UNIT ( 3380 )
  SHR
REP DDDEF ( BPXLIB1 )
  PATH ( '/path_name1/' )
REP DDDEF ( CMACLIB )
  DATASET ( SYS1.MACLIB )
  SHR
REP DDDEF ( MACLIB )
  DATASET ( SYS1.MACLIB )
  OLD
REP DDDEF ( SMPDATA1 )
  DATASET ( MVSTGT1.SMPDATA1 )
  MOD
REP DDDEF ( SMPDATA2 )
  DATASET ( MVSTGT1.SMPDATA2 )
  MOD
REP DDDEF ( SMPMTS )
  DATASET ( SYS1.SMPMTS )
  OLD
REP DDDEF ( SMPLOG )
  DATASET ( SYS1.SMPLOG )
  OLD
REP DDDEF ( SYSLIB )
  CONCAT ( SMPMTS CMACLIB AMACLIB )
  MOD
ENDUCL.
```

Figure 20. DDDEF entry: sample UNLOAD output

**UCLIN Examples**

You can use the ADD, REP, and DEL UCL statements to change subentries in a DDDEF entry. When you use UCLIN to update a DDDEF entry, keep these points in mind:

- If a DEL statement deletes all the existing subentries in the entry, SMP/E deletes the entire entry.
- When SMP/E dynamically allocates a concatenation list, the order of the concatenation is the same as that specified in the DDDEF entry. Therefore, you cannot use ADD statements to update a DDDEF entry that already contains a concatenation list. SMP/E would not know the correct position for the new data.

For example, suppose you have a DDDEF entry for the SYSLIB data set that concatenates SMPMTS and MACLIB. You cannot use the following commands to add XYZMACS at the end of the list:
**DDDEF entry (distribution, target, and global zone)**

```plaintext
SET  BDY(DLIB1) /* Set to DLIB1 zone. */.
UCLIN /* */.
ADD  DDDEF(SYSLIB) /* */.
     CONCAT(XYZMACS) /* */.
ENDUCL /* */.

Instead, you can use these commands to replace the entire concatenation list:

```plaintext
SET  BDY(DLIB1) /* Set to DLIB1 zone. */.
UCLIN /* */.
REP  DDDEF(SYSLIB) /* */.
     CONCAT(SMPMTS, /* Replace entire list. */
        MACLIB, /* */
        XYZMACS) /* */.
ENDUCL /* */.
```

The following examples are provided to help you use the DDDEF entry.

**Example 1: Defining global zone DDDEFs for cataloged data sets**

For this example, assume you want to define a DDDEF entry for the SMPPTS and
SMPLOG. The data set names are SYS1.SMPPTS and SYS1.GLOBAL.SMPLOG, and
both data sets are cataloged on the system that you are running on. Here is an
example of how to define the DDDEF entries:

```plaintext
SET  BDY(GLOBAL) /* Set to global zone. */.
UCLIN /* */.
ADD  DDDEF(SMPPTS) /* DDDEF for PTS. */.
     DA(SYS1.SMPPTS) /* Data set is cataloged. */.
     OLD /* DISP=OLD. */.
ADD  DDDEF(SMPLOG) /* DDDEF for LOG. */.
     DA(SYS1.SMPLOG) /* Data set is cataloged. */.
     MOD /* DISP=MOD. */.
ENDUCL /* */.
```

**Example 2: Defining DLIB zone DDDEFs for cataloged data sets**

Assume you want SMP/E to dynamically allocate distribution libraries AOS12 and
ASAMPLIB during ACCEPT. Assuming these libraries are cataloged on the system
you are running SMP/E on, you can use the following commands to define the
DDDEF entries:

```plaintext
SET  BDY(DLIB1) /* Set to DLIB zone. */.
UCLIN /* */.
ADD  DDDEF(AOS12) /* DLIB ddname. */.
     DA(SYS1.AOS12) /* Assume cataloged. */.
     OLD /* DISP=OLD. */.
ADD  DDDEF(ASAMPLIB) /* DLIB ddname. */.
     DA(SYS1.ASAMPLIB) /* Assume cataloged. */.
     OLD /* OLD for update. */.
ENDUCL /* */.
```

Note: Both data sets have a disposition of OLD. This is because these data sets are
used for output when SMP/E calls the system utilities, and you want only
one job to update the data sets at a time.

**Example 3: Defining target zone DDDEFs for cataloged data sets**

Assume you want SMP/E to dynamically allocate data sets MODGEN, MACLIB,
SMPMTS, and SMPLOG during APPLY. Assuming these data sets are cataloged on
the system that you are running SMP/E on, you can use the following commands to define the
DDDEF entries:

```plaintext
SET  BDY(TGT1) /* Set to target zone. */.
UCLIN /* */.
ADD  DDDEF(MODGEN ) /* For concatenation. */.
     DA(SYS1.MODGEN) /* Data set is cataloged. */.
     SHR /* SHR for read. */.
ADD  DDDEF(MACLIB ) /* In case updated. */.
```
**Example 4: Defining a DDDEF for a noncataloged data set**

During APPLY processing, SMP/E may need to refer to some distribution library data sets not cataloged on the system that SMP/E is running on. For example, suppose AMODGEN is a distribution macro library not cataloged on the running system, and you want to use AMODGEN in the SYSLIB concatenation for assemblies. You can use the following commands to define the DDDEF entry:

```
SET BDY(TGT1)  /* Set to target zone. */.
UCLIN /* */.
ADD DDDEF(AMODGEN ) /* For concatenation. */.
  DA(SYS1.AMODGEN) /* Data set not cataloged. */.
  VOLUME(DLIB01) /* */.
  UNIT(3330-1) /* */.
  SHR /* SHR for read. */.
ENDUCL /* */.
```

Note: This data set has a disposition of SHR. This is because it is not updated during APPLY, but rather is used just for input.

**Example 5: Defining a concatenated DDDEF entry**

After you have defined DDDEF entries for all the data sets to be included in the SYSLIB concatenation, you can define the DDDEF entry that concatenates the data sets. This entry must specify the concatenation ddname and the order of concatenation. The ddname is the name of the DDDEF entry, and the order of concatenation is the order in which the DDDEF names are specified on the CONCAT operand.

You can use the following commands to define the DDDEF entry:

```
SET BDY(TGT1)  /* Set to target zone. */.
UCLIN /* */.
ADD DDDEF(SYSLIB) /* ddname for concatenation. */.
  CONCAT( /* Concatenation order is */.
    SMPMTS /* SMPMTS first, */.
    MACLIB /* then MACLIB, */.
    MODGEN /* then MODGEN, */.
    AMODGEN /* then AMODGEN. */.
  ) /* End of list. */.
ENDUCL /* */.
```

The SYSLIB concatenation is the same as if the following JCL were used:

```
//SYSLIB DD DSN=SYS1.SMPMTS,DISP=OLD
// DD DSN=SYS1.MACLIB,DISP=OLD
// DD DSN=SYS1.MODGEN,DISP=OLD
// DD DSN=SYS1.AMODGEN,DISP=SHR,
// UNIT=3330-1,VOL=SER=DLIB01
```
Note: SYS1.MACLIB was allocated as DISP=OLD, even though it is not updated during APPLY processing. This is to limit access by any other job while SMP/E is running. If you wanted to use a disposition of SHR when MACLIB is part of the concatenation, you can use the following commands to define an additional DDDEF entry for MACLIB and change the SYSLIB DDDEF entry:

```
UCLIN /* */.
ADD DDDEF(CMACLIB) /* For read access. */.
  DA(SYS1.MACLIB) /* Data set is cataloged. */.
  SHR /* Change DISP to SHR. */.
ADD DDDEF(SYSLIB) /* dname for concatenation. */.
  CONCAT( /* concatenation order is */.
    SMPMTS /* SMPMTS first, */.
    CMACLIB /* ***** NOTE CHANGE ***** */.
    MODGEN /* then MODGEN, */.
    AMODGEN /* then AMODGEN, */.
  ) /* End of list. */.
ENDUCL /* */.
```

This corresponds to the following JCL:

```
//SYSLIB DD DSN=SYS1.SMPMTS,DISP=OLD
// DD DSN=SYS1.CMACLIB,DISP=SHR
// DD DSN=SYS1.MODGEN,DISP=OLD
// DD DSN=SYS1.AMODGEN,DISP=SHR,
// UNIT=3330-1,VOL=SER=DLIB01
```

SYS1.MACLIB is now allocated with DISP=SHR so other jobs can still access it.

Example 6: Defining DDDEFs for temporary data sets

Assume you want SMP/E to dynamically allocate the SMPOUT and SMPWRK1 data sets during ACCEPT. You can use the following commands to define the DDDEF entries:

```
SET BDY(DLIB1) /* Set to DLIB zone. */.
UCLIN /* */.
ADD DDDEF(SMPOUT) /* SMPOUT dname. */.
  SYSOUT(A) /* SYSOUT class. */.
ADD DDDEF(SMPWRK1) /* SMPWRK1 dname. */.
  NEW /* New data set. */.
  DELETE /* Delete when finished. */.
  TRACKS /* Allocate in tracks. */.
  SPACE(25,25) /* prime and secondary space. */.
  DIR(25) /* 25 directory blocks. */.
  UNIT(SYSDA) /* Allocate on SYSDA. */.
ADD DDDEF(SMPWRK2) /* SMPWRK2 dname. */.
  DATACLAS(FB80CLAS) /* Add DATACLAS name. */.
  MGMTCLAS(SMPEMCLS) /* Add MGMTCLAS name. */.
  STORCLAS(SMPESCLS) /* Add STORCLAS name. */.
ENDUCL /* */.
```

Example 7: Defining a global zone DDDEF for SMPTLIB data sets

Assume you want SMP/E to dynamically allocate the SMPTLIB data sets when it receives SYSMODs packaged in relative files. In addition, you want these data sets to be protected by z/OS SecureWay Security Server. You can use the following commands to define the DDDEF entry:

```
SET BDY(GLOBAL) /* Set to global zone. */.
UCLIN /* */.
ADD DDDEF(SMPTLIB) /* SMPTLIB dname. */.
  DATACLAS(FB80CLAS) /* Add DATACLAS name. */.
  DSPREFIX(C87MVSP) /* Data set prefix. */.
  MGMTCLAS(SMPEMCLS) /* Add MGMTCLAS name. */.
```
**DDDEF entry (distribution, target, and global zone)**

```plaintext
STORCLAS(SMPESCLS) /* Add STORCLAS name. */
PROTECT /* Request SecureWay Security */
/* Server protection. */
ENDUCL /* */
```

Note: No initial or final disposition is specified. This is because when SMP/E dynamically allocates SMPTLIB data sets, it automatically specifies the initial and final disposition based on the command it is processing.

**Example 8: Defining a DLIB zone DDDEF for SMPTLIB data sets**

Assume you want SMP/E to dynamically allocate the SMPTLIB data sets when it accepts SYSMODs packaged in relative files. You can use the following commands to define the DDDEF entry:

```plaintext
SET BDY(DLIB1) /* Set to DLIB zone. */
UCLIN /* */
ADD DDDEF(SMPTLIB) /* SMPTLIB ddname. */
VOLUME(DLIB01 /* Volumes used - DLIB01 */
DLIB02 /* DLIB02 */
DLIB03 /* DLIB03. */
UNIT(3380) /* Allocate on 3380. */
ENDUCL /* */
```

Note: No initial or final disposition is specified. This is because when SMP/E dynamically allocates SMPTLIB data sets, it automatically specifies the initial and final disposition based on the command it is processing.

**Example 9: Protecting data sets**

Assume you have previously defined your SMPMTS and SMPSTS data sets outside of SMP/E and protected them with z/OS SecureWay Security Server. You now want to have these dynamically allocated, and you also want to keep a record that they are protected by z/OS SecureWay Security Server. You can use the following commands to define the DDDEF entries:

```plaintext
SET BDY(TGT1) /* Set to target zone. */
UCLIN /* */
ADD DDDEF(SMPMTS) /* SMPMTS ddname. */
DA(SYS1.SMPMTS) /* Data set is cataloged. */
OLD /* OLD for update. */
PROTECT /* Protected by SecureWay Security Server. */
ADD DDDEF(SMPSTS) /* SMPSTS ddname. */
DA(SYS1.SMPSTS) /* Data set is cataloged. */
OLD /* OLD for update. */
PROTECT /* Protected by SecureWay Security Server. */
ENDUCL /* */
```

**Example 10: Defining pathnames in a UNIX file system**

Assume a product you plan to add to your system contains load modules and elements that will reside in a UNIX file system. You want to have the pathnames dynamically allocated instead of having DD statements for them in a cataloged procedure. You can use the following commands to define the DDDEF entry for each pathname:

```plaintext
-----+----1----+----2---- ... ----5----+----6----+----7----
SET BDY(TGT1) /* Set to target zone. */
UCLIN /* */
ADD DDDEF(BPXLIB1) /* Specify DDDEF name. */
PATH('/path_name1/') /* Define pathname. */
ADD DDDEF(BPXLIB2) /* Specify DDDEF name. */
PATH('/path_name2/This/pathname/is/an/example/of/a/very/long/pathname/It/shows/a/long/name/')
```

Chapter 5. SMP/E data set entries  209
DDDEF entry (distribution, target, and global zone)

/* define pathname. */
/* */
ENDUCL

/* */
/* */
The DLIB entry describes a distribution library that was totally copied to a target library. It is created during JCLIN processing when SMP/E encounters a COPY or COPYMOD statement without any SELECT statements. The INDD value on the COPY or COPYMOD statement becomes the name of the DLIB entry, and the OUTDD value on the COPY or COPYMOD statement becomes the SYSLIB value in the DLIB entry. For additional information, see the “Processing” section of the JCLIN command chapter in \textit{SMP/E Commands}.

SMP/E uses the DLIB entry to determine where elements and load modules should be applied, as follows:

- **Element entries other than MOD.** If there is no SYSLIB value either in the element entry or on the element MCS, SMP/E looks for a DLIB entry whose name matches the DISTLIB value for the element. SMP/E applies the element to the library indicated by the SYSLIB value in the DLIB entry, and adds that SYSLIB value to the element entry.

- **MOD entries.** For MOD entries, processing is slightly different, because MOD entries do not contain a SYSLIB subentry.
  - For a MOD entry that points to an LMOD entry, SMP/E checks whether the DISTLIB value in the MOD entry matches the name of a DLIB entry. If there is a match, and if the SYSLIB value in the DLIB entry is different from the ones in the LMOD entry, the SYSLIB value in the DLIB entry is used in addition to the LMOD's SYSLIB values.
  - For a MOD entry that does not point to an LMOD entry, SMP/E checks whether the DISTLIB value in the MOD entry matches the name of a DLIB entry. If there is a match, SMP/E assumes a load module with the same name as the MOD entry should exist in the library specified by the SYSLIB value in the DLIB entry.
    - If there is already an LMOD entry with the same name as the MOD entry, and that LMOD entry does not already contain two SYSLIB values, SMP/E adds the SYSLIB value from the DLIB entry to the LMOD entry.
    - If there is no LMOD entry with the same name as the MOD entry, SMP/E creates one and adds the SYSLIB value from the DLIB entry to the LMOD entry.

Lastly, SMP/E updates the MOD entry by adding an LMOD subentry for the LMOD entry that was updated or created.

**Subentries**

These are the subentries for the DLIB entry as they appear in the LIST output:

\textit{name} is the name of the distribution library represented by the DLIB entry.

The name can contain from 1 to 8 alphanumeric characters.

\textit{LASTUPD} identifies the cause of the last change to this DLIB entry.

The UCL operand is \textit{LASTUPD(value)}. This subentry can contain one of the following values:

- **JCLIN** indicates that the change was made during JCLIN command processing.
- **UCLIN** indicates that the change was made as a result of UCLIN processing.
DLIB entry (distribution and target zone)

sysmod_id
indicates that the change was made during the installation of the indicated SYSMOD.

The SYSMOD ID must contain 7 alphanumeric characters.

LASTUPD TYPE
indicates how the entry was last changed.

The UCL operand is LASTUPDTYPE(value). This subentry can contain one of the following values:
ADD The entry has been added.
UPD The entry has been updated.

SYSTEM LIBRARIES
identifies the ddnames of the target libraries into which the distribution library should be copied.

The UCL operand is SYSLIB(ddname...).

- A DLIB entry used for modules can contain one or two SYSLIB values.
- A DLIB entry used for macros, source, or data elements should contain only one SYSLIB value.
- The ddnames can contain from 1 to 8 alphanumeric characters.

LIST Examples
To list all the DLIB entries in a particular zone, you can use the following commands:

SET BDY(TGT1) /* Set to requested zone. */
LIST DLIB /* List all DLIB entries. */

To list specific DLIB entries, you can use these commands:

SET BDY(TGT1) /* Set to requested zone. */
LIST DLIB(AMACLIB /* List only these two */ MACDLB01 /* entries. */)

The format of the LIST output for each DLIB entry is the same for both of these commands. The only difference is the number of DLIB entries listed. Figure 21 is an example of LIST output for DLIB entries.

```
NAME
AMACLIB SYSTEM LIBRARY = MACLIB
MACDLB01 SYSTEM LIBRARY = MACLIB MACTGT01
MODDLB01 LASTUPD = JXY1102 TYPE=ADD SYSTEM LIBRARY = LPALIB MODTGT01
MODDLB02 LASTUPD = JXY1121 TYPE=UPD SYSTEM LIBRARY = LPALIB MODTGT02
```

Figure 21. DLIB entry: sample LIST output
UNLOAD Examples

To dump the DLIB entries in UCL format, you can use the UNLOAD command. To unload all the DLIB entries in a particular zone, you can use the following commands:

```
SET BDY(TGT1) /* Set to requested zone. */.
UNLOAD DLIB /* Unload all DLIB entries. */.
```

To unload specific DLIB entries, you can use these commands:

```
SET BDY(TGT1) /* Set to requested zone. */.
UNLOAD DLIB(AMACLIB MACDLB01) /* Unload only these two entries. */.
```

The format of the UNLOAD output for each DLIB entry is the same for both of these commands. The only difference is the number of DLIB entries listed. Figure 22 is an example of UNLOAD output for DLIB entries.

```
UCLIN .
REP  DLIB ( AMACLIB )
    SYSLIB ( MACLIB )
.
REP  DLIB ( MACDLB01 )
    SYSLIB ( MACLIB MACTGT01 )
.
REP  DLIB ( MODDLB01 )
    LASTUPD ( JXY1102 )
    LASTUPDTYPE ( ADD )
    SYSLIB ( LPALIB MDTGT01 )
.
REP  DLIB ( MODDLB02 )
    LASTUPD ( JXY1121 )
    LASTUPDTYPE ( UPD )
    SYSLIB ( LPALIB MDTGT02 )
.
ENDUCL.
```

Figure 22. DLIB entry: sample UNLOAD output

UCLIN Examples

You can use the ADD, REP, and DEL UCL statements to change subentries in the DLIB entry. When you use UCLIN to update a DLIB entry, keep these points in mind:

- After the UCLIN changes are made, the DLIB entry must contain at least a SYSLIB subentry. Otherwise, there is not enough information in the entry to indicate where elements should be installed.
- If a DEL statement deletes all the existing subentries in the entry, SMP/E deletes the entire entry.

Example: Changing the destination of a copied library

Assume you are doing a system generation and have completed the following steps:
1. Performed stage 1 system generation
2. Allocated new target zone
3. Copied distribution zone to new target zone
4. Ran SMP/E JCLIN command
You now want to change the target library into which a module distribution library has been totally copied. Assume that the DLIB was MODDLB01, that it was previously copied to LINKLIB, and that it is now copied to LPALIB. You can do this with the following UCL statements:

```
SET BDY(TGT1)  /* Set to target zone. */.
UCLIN  /* */.
REP DLIB(MODDLB01)  /* Module DLIB. */.
  SYSLIB(LPALIB)  /* Now copied to LPALIB. */.
  /* No longer in LINKLIB. */.
ENDUCL  /* */.
```

If the modules should also be copied into LPALIB, the following UCL can be used:

```
SET BDY(TGT1)  /* Set to target zone. */.
UCLIN  /* */.
ADD DLIB(MODDLB01)  /* Module DLIB. */.
  SYSLIB(LPALIB)  /* Now copied to LPALIB. */.
  /* SYSLIB added, not replaced. */.
ENDUCL  /* */.
```

In both cases, you must take care of copying the modules from the distribution library. SMP/E does not process elements when it records UCL changes.

Now that the UCL changes are made, when SMP/E installs SYSMODs that update MODDLB01, SMP/E checks the DLIB entry for MODDLB01, determines that the modules are copied to LPALIB, and creates or modifies the LMOD entries (with the same name as the MOD entries) to indicate a copy to LPALIB.

**Note:** This technique works before you actually install any service. If you want to change (not add) another library in the DLIB copy list, you must find all the MOD, MAC, SRC, and LMOD entries with a SYSLIB value matching the old target library, and then change that value using UCLIN. If you do not do this, SMP/E continues to copy entries with the old SYSLIB value to the old library (and also, in some cases, to the new library).
DLIBZONE entry (distribution zone)

The DLIBZONE entry contains information SMP/E uses to process a specific distribution zone and the associated distribution libraries. It is created by UCLIN and must be defined before you can do any other processing for that distribution zone.

Subentries

These are the subentries for the DLIBZONE entry as they appear in the LIST output:

name

is the name of the distribution zone. You assign the name when the zone is created.

The name can contain from one to seven alphanumeric characters (A through Z, 0 through 9) or national characters ($, #, @). The first character must be alphabetic.

ACCJCLIN

indicates that JCLIN is to be saved in the distribution zone whenever a SYSMOD containing inline JCLIN is accepted.

The UCL operand is ACCJCLIN.

• ACCJCLIN should be specified only for zones in which all the products have been installed with SMP/E Release 3, or later.
• To save inline JCLIN for a product at ACCEPT time, you must first accept that product using SMP/E Release 3, or later, and have ACCJCLIN set in the DLIBZONE entry. From then on, you must keep ACCJCLIN set in the DLIBZONE entry. This ensures that any time you accept service for that product, its JCLIN is updated in the distribution zone. For more information, see the section on inline JCLIN in the ACCEPT command chapter in SMP/E Commands.

OPTIONS

is the name of the OPTIONS entry in the global zone that should be used when processing this distribution zone. For more information, see “OPTIONS entry (global zone)” on page 292.

The UCL operand is OPTIONS(name).

• The name can contain from one to eight alphanumeric characters.
• This name can be overridden by using the OPTIONS parameter on the SET command. For more information about the SET command, see SMP/E Commands.

• If no OPTIONS entry name is specified, SMP/E uses a set of default utility values when processing this distribution zone. For more information, see “OPTIONS entry (global zone)” on page 292.

RELATED

is the name of the target zone to which this distribution zone is related. A distribution zone is related to the target zone that was built from these distribution libraries, such as during system generation.

The UCL operand is RELATED(zone).

• The zone name can contain from one to seven alphanumeric characters.
• Although the entry can be defined without this subentry, you must define the subentry before you can install any SYSMODs into the distribution libraries.
DLIBZONE entry (distribution zone)

**SREL**

lists the system releases to be supported within this distribution zone.

The UCL operand is **SREL(srel...)**.

- The SREL must contain four alphanumeric characters, usually one alphabetic character followed by three numeric characters. These are the SRELs defined by IBM:

<table>
<thead>
<tr>
<th>System</th>
<th>SREL</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB2</td>
<td>P115</td>
</tr>
<tr>
<td>CICS</td>
<td>C150</td>
</tr>
<tr>
<td>IMS</td>
<td>P115</td>
</tr>
<tr>
<td>MVS</td>
<td>Z038</td>
</tr>
<tr>
<td>NCP</td>
<td>P004</td>
</tr>
</tbody>
</table>

- Although the entry can be defined without this subentry, you **must** define the subentry before you can install any SYSMODs in the distribution libraries.

**Note:** Although you can support multiple products with different SREL values from one distribution zone, those products are still subject to all other restrictions related to combining products in one zone. The most common reason for not being able to combine products is common element names. For example, modules or macros with the same name are found in both products, but reside in different libraries.

**UPGLEVEL**

indicates the highest SMP/E release level that is allowed to make incompatible changes to the distribution zone. Before making an incompatible change to the distribution zone, SMP/E will check the UPGLEVEL value for that zone. If the release level of SMP/E is higher than the zone's UPGLEVEL value, SMP/E will not make the incompatible change.

The UPGLEVEL value is in the form `vr.pp`, where `vr` represents the version and release of SMP/E and `pp` represents the PTF level of SMP/E.

There is no UCL support for this subentry. When a zone is created by SMP/E using the UCLIN command or the Administration dialog, SMP/E sets the UPGLEVEL subentry value for that zone to the level of SMP/E used to create the zone. The UPGRADE command is used to change the UPGLEVEL subentry value for a zone.

**ZDESC**

is a user-written description for this zone.

The UCL operand is **ZONEDESCRIPTION(text)**.

- The zone description can be in single-byte characters (such as English alphanumeric characters) or in double-byte characters (such as Kanji).
- The zone description can contain up to 500 bytes of data, including blanks. (For double-byte data, the 500-byte maximum includes all shift-in and shift-out characters, as well as the double-byte characters.) Extra blanks are deleted. All data beyond column 72 is ignored, including blanks.
- The zone description cannot be only blanks.
- If parentheses are included in the text, they must be in matched pairs.

**LIST Examples**

To list the DLIBZONE entry for a particular distribution zone, you can use the following commands:
DLIBZONE entry (distribution zone)

```
SET BDY(DLIB1) /* Set to requested DLIB. */.
LIST DLIBZONE /* List DLIBZONE entry. */.
```

Figure 23 is an example of LIST output for a DLIBZONE entry.

```
NAME

DLIB1 DZONE ENTRY

NAME

DLIB1 DZONE = DLIB1
ZDESC = ZONE DESCRIPTION FOR DLIB1 ZONE
RELATED = TGT2
SREL = Z038
OPTIONS = OPTDLIB2
ACCJCLIN

Figure 23. DLIBZONE entry: sample LIST output
```

UCLIN Examples

You can use the ADD, REP, and DEL UCL statements to change subentries in the DLIBZONE entry. When you use UCLIN to update a DLIBZONE entry, remember that if a DEL statement deletes all the existing subentries in the entry, SMP/E deletes the entire entry.

The following examples are provided to help you use the DLIBZONE entry.

Example 1: Defining a DLIBZONE entry

Assume you are about to add a new function to your system and want to define a separate distribution zone for it named DLIB2. The OPTIONS entry you plan to use is OPTDLIB2, and the SREL for the SYSMOD is Z038. The distribution zone is in data set SMPE.SMPCSI.CSI. After installing the function in the distribution libraries, you plan to do a system generation to build a set of target libraries. The target zone for those libraries is TGT2. The following UCLIN can be used to define the new zone.

```
SET BDY(GLOBAL) /* Set to global zone. */.
UCLIN /* UCLIN for GZONE entry */.
ADD GZONE /* to set up */
ZONEINDEX( /* index for new zone. */
(DLIB2,SMPE.SMPCSI.CSI,DLIB) /* */
) /* */.
ENDUCL /* */.
SET BDY(DLIB2) /* Now define new zone. */.
UCLIN /* UCLIN to define it. */.
ADD DLIBZONE(DLIB2) /* Identify name. */
OPTIONS(OPTDLIB2) /* OPTIONS entry to use. */
SREL(Z038) /* SREL for MVS. */
RELATED(TGT2) /* For this tgt library. */
ZDESC(
THIS DISTRIBUTION ZONE IS FOR SREL Z038
) /* Zone description. */
) /* */.
ENDUCL /* */.
```

Note: Even if the OPTIONS entry or TARGETZONE entry has not yet been defined, you can still refer to it in the DLIBZONE entry. However, you must
create the entry before you process this distribution zone. For examples of defining OPTIONS entries, see “OPTIONS entry (global zone)” on page 292. For examples of defining TARGETZONE entries, see “TARGETZONE entry (target zone)” on page 346.

Example 2: Formatting a zone description
Assume you enter the following zone description with the first line ending in column 72 and the second line starting in column 1:

```plaintext
-----+----1----+----2---- ... ----5----+----6----+----7--
SET  BDY(DLIB1) /* Set to DLIB zone. */.
UCLIN /* UCLIN for DZONE entry */.
ADD  DZONE(DLIB1) /* to set up. */.
     ZDESC(THE DLIB1 ZONE)
     /* End of zone description. */.
ENDUCL /* End DLIB zone update. */.
```

Because there is no blank between the word ending in column 72 and the next word starting in column 1, SMP/E runs the two together.

The words in a zone description, even words that end in column 72, must be separated by blanks. To format the zone description in this example correctly, you can put a blank at the beginning of the second line:

```plaintext
-----+----1----+----2---- ... ----5----+----6----+----7--
SET  BDY(DLIB1) /* Set to DLIB zone. */.
UCLIN /* UCLIN for DZONE entry */.
ADD  DZONE(DLIB1) /* to set up. */.
     ZDESC(THE DLIB1 ZONE)
     /* End of zone description. */.
ENDUCL /* End DLIB zone update. */.
```

Because there is a blank explicitly coded between the word ending in column 72 and the word starting in column 1, SMP/E does not run the words together.
The FEATURE entry describes a collection of function SYSMODS called a feature. An SMP/E feature may be associated with an orderable software feature.

**Subentries**

- **name**
  - is the name of this feature.

- **DESCRIPTION**
  - describes the software feature.
  - The UCL operand is `DESCRIPTION(description)`.

- **FMID**
  - specifies the FMIDs that make up this software feature. There can be multiple FMID subentries associated with a single feature.
  - The UCL operand is `FMID(fmid,...)`.

- **PRODUCT**
  - identifies the product with which this feature is associated.
  - The UCL operand is `PRODUCT(prodid,vv,rr,mm)`.

- **RECDATE**
  - specifies the date on which the feature was received.
  - There is no UCL support for this subentry.

- **RECTIME**
  - specifies the time at which the feature was received.
  - There is no UCL support for this subentry.

- **UCLDATE**
  - specifies the date on which the feature was last processed using the UCLIN command.
  - There is no UCL support for this subentry.

- **UCLTIME**
  - specifies the time at which the feature was last processed using the UCLIN command.
  - There is no UCL support for this subentry.

- **REWORK**
  - identifies the level of this feature, which was received again for minor changes.
  - There is no UCL support for this subentry.

**LIST Examples**

To list all the FEATURE entries in a particular zone, you can use the following commands:

```
SET BDY(GLOBAL) /* Set to global zone. */.
LIST FEATURE /* List all FEATURE entries.*/.
```

To list specific FEATURE entries, you can use these commands:

```
SET BDY(GLOBAL) /* Set to global zone. */.
LIST FEATURE(OS3250BA /* List only these two */
  OS3250LD) /* entries. */.
```
FEATURE entry (global zone)

The format of the LIST output for each FEATURE entry is the same for both of these commands. The only difference is the number of FEATURE entries listed. Figure 24 shows an example of LIST output for FEATURE entries. FEATURE entries are listed alphanumerically by name.

By specifying the FORFMID operand, you can reduce the number of FEATURE entries listed. When FORFMID is specified, SMP/E lists a FEATURE entry only if its FMID matches one of the FMIDs specified on the FORFMID operand. For example, to list FEATURE entries whose FMIDs either are defined in FMIDSET ZOSSET or are HMP1E00, you can use these commands:

```
SET BDY(GLOBAL) /* Set to global zone. */.
LIST FEATURE /* List all FEATURE entries */.
FORFMID(ZOSSET /* for the ZOSSET FMIDSET */.
HMP1E00) /* and FMID HMP1E00. */.
```

UCLIN Examples

You can use the ADD, REP, and DEL UCL statements to change subentries in the FEATURE entry. When you use UCLIN to update a FEATURE entry, keep in mind that after the UCLIN changes are made, the FEATURE entry must contain at least the DESCRIPTION and PRODUCT subentries.

Example: Adding a FEATURE entry

Assume you have a feature, named Sample Feature, for which you wish to create a description and put in an entry named SAMPFEAT. You could set up a FEATURE entry as follows:

```
SET BDY(GLOBAL) /* Set to global zone. */.
UCLIN /* Start UCLIN processing */.
ADD FEATURE(SAMPFEAT) /* Identify the feature: */.
DESCRIPTION(Sample Feature) /* - Name */.
PRODUCT(1234-ABC,02.05.00) /* - Owning product */.
FMID(XYZ1234) /* - FMID */.
ENDUCL /* End UCLIN processing */.
```
The FMIDSET entry defines a group of FMIDs for use in limiting the SYSMODs processed by an SMP/E command. For example, you can specify an FMIDSET on the FORFMID operand of the APPLY command to process only SYSMODs applicable to one of the FMIDs in the FMIDSET.

**Subentries**

These are the subentries for the FMIDSET entry as they appear in the LIST output:

- **name** is the name of the FMIDSET.
  - The name can contain from 1 to 8 alphanumeric characters.

- **FMID** lists the function SYSMODs (that is, FMIDs) that are to be part of this FMIDSET.
  - The UCL operand is `FMID(sysmod_id...)`.
  - The SYSMOD ID must contain 7 alphanumeric characters.

**LIST Examples**

To list all the FMIDSET entries in a global zone, you can use the following commands:

```plaintext
SET BDY(GLOBAL) /* FMIDSET in global only. */.
LIST FMIDSET /* list all FMIDSET entries. */.
```

To list specific FMIDSET entries in a global zone, you can use these commands:

```plaintext
SET BDY(GLOBAL) /* FMIDSET in global only. */.
LIST FMIDSET(ALLSET /* List only these two */ XXSET) /* entries. */.
```

The format of the LIST output for each FMIDSET entry is the same for both of these commands. The only difference is the number of FMIDSET entries listed.

**Figure 25** is an example of LIST output for FMIDSET entries.

---

**Figure 25. FMIDSET entry: sample LIST output**
FMIDSET entry (global zone)

**UCLIN Examples**

You can use the ADD, REP, and DEL UCL statements to change subentries in an FMIDSET entry. When you use UCLIN to update an FMIDSET entry, keep these points in mind:

- After the UCLIN changes are made, the FMIDSET entry must contain at least an FMID subentry. Otherwise there is not enough information in the entry for SMP/E to use the entry.
- If a DEL statement deletes all the existing subentries in the entry, SMP/E deletes the entire entry.

The following examples are provided to help you use the FMIDSET entry.

**Example 1: Defining an FMIDSET entry**

Assume you have four function SYSMODs installed, JXX1102, JXX1121, JXX1122, and JXX1123, and that when installing service you would like to be able to install just the PTFs for these products. You could set up an FMIDSET entry as follows:

```
SET BDY(GLOBAL) /* FMIDSETs only in global. */.
UCLIN /* */.
ADD FMIDSET(XXSET) /* Define FMIDSET name. */.
  FMID(FXX1102 /* Define function SYSMODs */.
    JXX1121 /* to be part of set. */.
    JXX1122 /* */.
    JXX1123) /* */.
  /* */.
ENDUCL /* */.
```

You can use the FORFMID operand on the APPLY as follows:

```
SET BDY(TGT1) /* Set for APPLY to TGT1. */.
APPLY FORFMID(XXSET) /* Apply XX PTFs. */.
```

**Example 2: Modifying an FMIDSET entry**

Once an FMIDSET has been defined, it can be modified as new function SYSMODs are installed or as existing function SYSMODs are replaced. For example, starting with the FMIDSET from the previous example, assume you install a new feature, JXX1124, that deletes JXX1121, and that you want to modify the FMIDSET definition. You can do this as follows:

```
SET BDY(GLOBAL) /* FMIDSETs only in global. */.
UCLIN /* */.
REP FMIDSET(XXSET) /* Define FMIDSET name. */.
  FMID(FXX1102 /* Define function SYSMODs */.
    JXX1121 /* to be part of set. */.
    JXX1122 /* */.
    JXX1123) /* */.
  /* */.
ENDUCL /* */.
```

You can also do the following:

```
SET BDY(GLOBAL) /* FMIDSETs only in global. */.
UCLIN /* */.
DEL FMIDSET(XXSET) /* Define FMIDSET name. */.
  FMID(JXX1121) /* Delete JXX1121 from set. */.
  /* */.
ADD FMIDSET(XXSET) /* Define FMIDSET name. */.
  FMID(JXX1124) /* Add new FMID. */.
  /* */.
ENDUCL /* */.
```
The result in both cases is the same. The choice of which method to use most likely depends on the number of FMIDs defined in the FMIDSET.
GLOBALZONE entry (global zone)

The GLOBALZONE entry contains processing-related information for SMP/E. It is also used by SMP/E as an index to target and distribution zones, either in the same CSI or a different CSI data set. The GLOBALZONE entry is created by UCLIN and must be defined before you can do any other processing for that global zone.

Subentries

These are the subentries for the GLOBALZONE entry as they appear in the LIST output:

FMID

lists the function SYSMODs for which SMP/E is to receive service.

The UCL operand is FMID(sysmod_id...).

The SYSMOD ID must contain seven alphanumeric characters.

OPTIONS

is the name of the OPTIONS entry in the global zone that should be used in processing this global zone. For more information, see "OPTIONS entry (global zone)" on page 292.

The UCL operand is OPTIONS(name).

• The name can contain from one to eight alphanumeric characters.

• This name can be overridden by using the OPTIONS parameter on the SET command. For more information about the SET command, see SMP/E Commands.

• If no OPTIONS entry name is specified, SMP/E uses a set of default utility values when processing the global zone. For more information, see "OPTIONS entry (global zone)" on page 292.

SREL

lists the system releases to be supported in this global zone.

The UCL operand is SREL(srel...).

The SREL must contain four alphanumeric characters, usually one alphabetic character followed by three numerics. These are the SRELs defined by IBM:

<table>
<thead>
<tr>
<th>System</th>
<th>SREL</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB2</td>
<td>P115</td>
</tr>
<tr>
<td>CICS</td>
<td>C150</td>
</tr>
<tr>
<td>IMS</td>
<td>P115</td>
</tr>
<tr>
<td>MVS</td>
<td>Z038</td>
</tr>
<tr>
<td>NCP</td>
<td>P004</td>
</tr>
</tbody>
</table>

Note: Although you can support multiple products with different SREL values from one global zone, those products are still subject to all other restrictions related to combining products in one zone. The most common reason for not being able to combine products is common element names. For example, modules or macros with the same name are found in both products, but reside in different libraries.

UPGLEVEL

indicates the highest SMP/E release level that is allowed to make incompatible changes to the global zone. Before making an incompatible change to the
GLOBALZONE entry (global zone)

global zone, SMP/E will check the UPGLEVEL value for that zone. If the release level of SMP/E is higher than the zone's UPGLEVEL value, SMP/E will not make the incompatible change.

The UPGLEVEL value is in the form \textit{vr.pp}, where \textit{vr} represents the version and release of SMP/E and \textit{pp} represents the PTF level of SMP/E.

There is no UCL support for this subentry. When a zone is created by SMP/E using the UCLIN command or the Administration dialog, SMP/E sets the UPGLEVEL subentry value for that zone to the level of SMP/E used to create the zone. The UPGRADE command is used to change the UPGLEVEL subentry value for a zone.

\textbf{ZDESC}

is a user-written description for this zone.

The UCL operand is \textbf{ZONEDESCRIPTION(\textit{text})}.

- The zone description can be in single-byte characters (such as English alphanumeric characters) or in double-byte characters (such as Kanji).
- The zone description can contain up to 500 bytes of data, including blanks. (For double-byte data, the 500-byte maximum includes all shift-in and shift-out characters, as well as the double-byte characters.) Extra blanks are deleted. All data beyond column 72 is ignored, including blanks.
- The zone description cannot be only blanks.
- If parentheses are included in the text, they must be in matched pairs.

\textbf{ZONEINDEX}

identifies all the target zones and distribution zones associated with this global zone. This list is used by SMP/E to determine the data set in which the zone resides and the type of zone.

For ADD and REP statements, the UCL operand is \textbf{ZONEINDEX((\textit{name},\textit{dsn},\textit{type})...)}.  

\textbf{Note:} You cannot use REP to replace a single zone in the ZONEINDEX list. REP will cause the entire list of zones to be replaced. Instead, you must first delete the zone from the ZONEINDEX list, and then add the zone to the list with the new SMPCS\textit{I} data set name.

For DEL statements, the UCL operand is \textbf{ZONEINDEX((\textit{name})...}).

- \textbf{ZONEINDEX} can also be specified as \textbf{ZINDEX}.
- \textit{name} is the name of the zone. The name can contain from one to seven alphanumeric characters (A through Z, 0 through 9) or national characters ($, #, @). The first character must be alphabetic.
- \textit{dsn} is the fully qualified name of the data set in which the zone resides.
- \textit{type} is the zone type, either TARGET or DLIB.

The ZONEINDEX is only a pointer to a zone. Changes you make to a ZONEINDEX do not affect the associated zone. Therefore, you cannot add, rename, or delete a zone by simply adding, replacing, or deleting its ZONEINDEX. However, these are some other commands you can use:

- **Adding a zone:** To add a zone, you can use the following commands, depending on the particular situation:
  - UCLIN to add a ZONEINDEX, then UCLIN to define the DZONE or TZONE entry
  - ZONECOPY
  - ZONERENAME
GLOBALZONE entry (global zone)

- ZONEEXPORT, UCLIN for the ZONEINDEX, and ZONEIMPORT

**Renaming a zone:** To rename a zone, you must use the following command:
- ZONERENAME

**Deleting a zone:** To delete a zone, you can use the following commands, depending on the particular situation:
- ZONEDELETE
- ZONEEXPORT

For more information about any of these commands, see the chapter on that command.

**LIST Examples**

To list all the GLOBALZONE entry, you can use the following commands:

```
SET BDY(GLOBAL) /* Set to global zone. */.
LIST GLOBALZONE /* List GLOBALZONE entry. */.
```

Figure 26 is an example of LIST output for a GLOBALZONE entry.

```
GLOBAL GZONE ENTRY

NAME
GLOBAL ZONEINDEX = TGT1 TARGET SYS1.TGT.SMPCSI.CSI
TGT2 TARGET SYS1.TGT.SMPCSI.CSI
DLIB1 DLIB SYS1.DLIB.SMPCSI.CSI
DLIB2 DLIB SYS1.DLIB.SMPCSI.CSI

ZDESC = ZONE DESCRIPTION FOR GLOBAL ZONE
SREL = Z038 I115 P115 R020
FMID = JXY1102 FXY1121 FXY1122 FXY1123
OPTIONS = OPTGBL
```

Figure 26. GLOBALZONE entry: sample LIST output

**UCLIN Examples**

You can use the ADD, REP, and DEL UCL statements to change subentries in the GLOBALZONE entry. When you use UCLIN to update a GLOBALZONE entry, keep these points in mind:

- After the UCLIN changes are made, the GLOBALZONE entry must contain at least one of these subentries: FMID, OPTIONS, SREL, or ZONEINDEX. Otherwise, there is not enough information in the entry for SMP/E to use the entry.
- If a DEL statement deletes all the existing subentries in the entry, SMP/E deletes the entire entry.

The following examples are provided to help you use the GLOBALZONE entry.

**Example 1: Defining the GLOBALZONE entry**

To perform any SMP/E processing, you must first define the GLOBALZONE entry. Given the following assumptions:
- You are supporting functions JXY1102, JXY1121, JXY1122, and JXY1123 from this global zone.
- You are supporting system releases Z038, P115, and R020, and I121.
GLOBALZONE entry (global zone)

- You are eventually going to define two target zones, TGT1 and TGT2, to exist on SYS1.TGT.SMPCSI.CSI, and two distribution zones, DLIB1 and DLIB2, to exist on SYS1.DLIB.SMPCSI.CSI.
- The OPTIONS entry used is OPTGBL.

The following UCLIN can be used to define the GLOBALZONE entry:

```
SET BDY(GLOBAL) /* Set to global zone. */.
UCLIN /* */.
ADD GLOBALZONE /* Define global now. */.
  SREL(Z038,P115, /* Identify SRELs. */.
            R020,I115) /* */.
  FMID(JXY1102, /* Identify functions. */.
           JXY1121, /* */.
           JXY1122, /* */.
           JXY1123) /* */.
  OPTIONS(OPTGBL) /* OPTIONS entry to be used. */.
  ZONEINDEX( /* */.
            (TGT1,SYS1.TGT.SMPCSI.CSI,TARGET) /* */.
            (TGT2,SYS1.TGT.SMPCSI.CSI,TARGET) /* */.
            (DLIB1,SYS1.DLIB.SMPCSI.CSI,DLIB) /* */.
            (DLIB2,SYS1.DLIB.SMPCSI.CSI,DLIB) /* */.
      ) /* */.
ENDUCL /* */.
```

Note: Even though the OPTIONS entry has not been set up yet, you can still refer to it in the GLOBALZONE entry. The same is true for the target zone name. However, the OPTIONS entry must be created before the global zone is processed. For examples of setting up the OPTIONS entries, see “OPTIONS entry (global zone)” on page 292.

Example 2: Deleting two ZONEINDEX entries
Assume you decide not to create the TGT1 target zone and the DLIB1 distribution zone defined in the ZONEINDEX of example 1. You should delete the entries for those two zones by using UCL as follows:

```
SET BDY(GLOBAL) /* Set to global zone. */.
UCLIN /* */.
DEL GLOBALZONE /* Define global now. */.
  ZONEINDEX( /* */.
            (TGT1) /* Delete TGT1 entry. */.
            (DLIB1) /* Delete DLIB1 entry. */.
      ) /* */.
ENDUCL /* */.
```

Note: The CSI data set name and the zone type operands are not required for a delete, although the open and close parentheses around the zones are still required.

Example 3: Formatting a zone description
Assume you enter the following zone description with the first line ending in column 72 and the second line starting in column 1:

```
---+----1----+----2---- ... ----5----+----6----+----7--
SET BDY(GLOBAL) /* Set to global zone. */.
UCLIN /* */.
ADD GZONE /* to set up. */.
  ZDESC( THIS IS THE DESCRIPTION FOR
      THE GLOBAL ZONE) /* End of zone description. */.
ENDUCL /* End global zone update. */.
```
GLOBALZONE entry (global zone)

Because there is no blank between the word ending in column 72 and the next word starting in column 1, SMP/E runs the two together.

The words in a zone description, even words that end in column 72, must be separated by blanks. To format the zone description in this example correctly, you can put a blank at the beginning of the second line:

```
---+----+----1----+----2---- ... ----5----+----6----+----7--
SET BDY(GLOBAL) /* Set to global zone. */.
UCLIN /* UCLIN for GZONE entry */.
ADD GZONE /* to set up. */
   ZDESC( THIS IS THE DESCRIPTION FOR
   THE GLOBAL ZONE)
   /* End of zone description. */.
ENDUCL /* End global zone update. */.
```

Because there is a blank explicitly coded between the word ending in column 72 and the word starting in column 2, SMP/E does not run the words together.

Example 4: Renaming an SMPCSI data set and updating a ZONEINDEX

Assume you have a target zone named TGT1 that resides in a data set named SMPE.TEST.TGT1.CSI, and you decide to rename the data set to SYS1.PROD.TGT1.CSI. After renaming the data set, you must use UCLIN to update the ZONEINDEX for the target zone in the global zone, as follows:

```
SET BDY(GLOBAL) /* Set to global zone. */.
UCLIN /* UCLIN for GZONE entry */.
DEL GZONE ZINDEX( (TGT1) ) /* Delete old TGT1 zone. */.
ADD GZONE ZINDEX( (TGT1, SYS1.PROD.TGT1.CSI, TARGET) ) /* Supply new TGT1 zone. */.
ENDUCL /* End global zone update. */.
```

You cannot use REP to replace a single zone in the ZONEINDEX list. REP will cause the entire list of zones to be replaced. Instead, you must first delete the zone from the ZONEINDEX list, and then add the zone to the list with the new SMPCSI data set name.
Hierarchical file system element entry (distribution and target zone)

The hierarchical file system element entry describes an element that exists in a distribution library or a UNIX file system. A hierarchical file system entry is created the first time you install a SYSMOD containing an MCS for a hierarchical file system element that does not yet have an entry in the CSI data set.

SMP/E records the function and service level of the hierarchical file system element in the entry. Once a hierarchical file system element entry exists, it is updated as subsequent SYSMODs affecting the hierarchical file system element are installed.

Table 4 on page 26 shows the types of entries used for hierarchical file system elements. Some types of elements may be translated into several languages. In these cases, the entry type contains $XXX$, which represents the language used for the element. If an element was not translated, the entry type does not contain any $XXX$ value. Table 5 on page 27 shows the $XXX$ values and the languages they represent.

Subentries

These are the subentries for the hierarchical file system element entries as they appear in the LIST output:

name

is the name of the hierarchical file system element represented by the entry.

The name can contain 1–8 uppercase alphabetic, numeric, or national ($, #, @) characters.

BINARY or TEXT

indicates the installation mode to be used when the HFS copy utility is invoked to install the element into a UNIX file system.

The UCL operand is **BINARY** or **TEXT**.

- **Binary** mode means that the element is installed in its entirety as a data stream, with no breaks for logical records.
- **Text** mode means that the element is installed with breakpoints for logical records.
- **BINARY** and **TEXT** are mutually exclusive.
- If there is no mode indicator in the hierarchical file system element entry, the HFS copy utility determines how to install the element.

DISTLIB

specifies the ddname of the distribution library for the hierarchical file system element.

The UCL operand is **DISTLIB(ddname)**.

- The ddname can contain any uppercase alphabetic, numeric, or national ($, #, @) character, and can be 1–8 characters long.
- The DISTLIB subentry is required. Without it, SMP/E cannot process any changes for the hierarchical file system element.

FMID

identifies the functional owner of this hierarchical file system element. The functional owner is the last function SYSMOD that replaced this element.

The UCL operand is **FMID(sysmod_id)**.

The SYSMOD ID must contain 7 uppercase alphabetic, numeric, or national ($, #, @) characters.
Hierarchical file system element entry (distribution and target zone)

LASTUPD
identifies the cause of the last change to this hierarchical file system element entry.

The UCL operand is LASTUPD(value). This subentry can contain one of the following values:

UCLIN
indicates that the change was made as a result of UCLIN processing.

sysmod-id
indicates that the change was made during the installation of the indicated SYSMOD.

The SYSMOD ID must contain 7 uppercase alphabetic, numeric, or national ($, #, @) characters.

LASTUPD TYPE
indicates how the entry was last changed.

The UCL operand is LASTUPDTYPE(value). This subentry can contain one of the following values:

ADD The entry was added.

UPD The entry was updated.

LINK
specifies a list of alternative names by which this hierarchical file system element can be known in a UNIX file system. The full name is produced by concatenating the specified linkname with the UNIX file system directory identified by the SYSLIB subentry.

In LIST output, linknames are always enclosed in single apostrophes. If an apostrophe is part of a linkname, it is always shown as two consecutive apostrophes in LIST output.

The UCL operand is LINK(linkname...).

• The linkname can be from 1 to 1023 characters.
• A linkname can be enclosed in single apostrophes ('). A linkname must be enclosed in single apostrophes if any of the following is true:
  – The linkname contains lowercase alphabetic characters.
  – The linkname contains a character that is not uppercase alphabetic, numeric, national ($, #, or @), slash (/), plus (+), hyphen, period, or ampersand (&).
  – The linkname spans more than one line in the control statement.

The single apostrophes used to enclose a linkname (the delimiters) do not count as part of the 1023-character limit.

• Any apostrophes specified as part of a linkname (not the delimiters) must be doubled.
  Double apostrophes count as two characters in the 1023-character limit.

• The linkname can include characters X'40' through X'FE'.

PARM
specifies a character string that is to be passed to the HFS copy utility as an execution-time parameter. The maximum length of this character string is 300 bytes of nonblank data. If any blanks are specified in the PARM value, they are deleted by SMP/E during processing and do not count toward the 300-byte maximum.

The UCL operand is PARM(character_string).
Hierarchical file system element entry (distribution and target zone)

- PARM is an optional operand.
- The character string can be entered free-form, without regard to blanks (which are compressed out of the string), and can span multiple 80-byte records.
- If parentheses are specified in the PARM value, there must always be a pair (left and right); otherwise, the results are unpredictable.

RMID
identifies the last SYMSOD that replaced this hierarchical file system element. Any subsequent SYMSOD that modifies this hierarchical file system element must have defined a relationship (such as PRE or SUP) with this SYMSOD.

The UCL operand is RMID(sysmod_id).
- The SYMSOD ID must contain 7 uppercase alphabetic, numeric, or national ($, #, @) characters.
- If RMID is not specified but FMID is, SMP/E sets the RMID value to the specified FMID.

SHSCRIPT
specifies a UNIX shell script to run when this element is copied to (or deleted from) a directory in a UNIX file system.

The UCL operand is SHSCRIPT(scriptname). This subentry optionally contains one or both of the following values, which specify the point in SMP/E processing when the shell script is run:
- PRE  The shell script is run before the element is copied to the UNIX file system directory.
- POST The shell script is run after the element is copied to the UNIX file system directory.

POST is the default; the shell script is run after the element is copied to the directory.

SYMLINK
specifies a list of one or more symbolic links, which are file names that can be used as alternate names for referring to this element in a UNIX file system. Each symbolic link name listed here is associated with a path name listed in the SYMPATH entry. See the description of the SYMPATH entry for more information about how the symbolic link names and path names are associated.

The UCL operand is SYMLINK(symlinkname...).
- A hierarchical file system element entry that contains a SYMLINK entry must contain a matching SYMPATH entry. SMP/E will reject any UCLIN command that would violate this condition.
- A symbolic linkname can be from one to 1023 characters. Any characters in the range X'40' through X'FE' may be specified.
- A symbolic linkname can be enclosed in single apostrophes ('). A symbolic linkname must be enclosed in single apostrophes if any of the following is true:
  - The symbolic linkname contains lowercase alphabetic characters.
  - The symbolic linkname contains a character that is not uppercase alphabetic, numeric, national ($, #, or @), slash (/), plus (+), hyphen, period, or ampersand (&).
  - The symbolic linkname spans more than one line in the control statement.

The single apostrophes used to enclose a symbolic linkname (the delimiters) do not count as part of the 1023-character limit.
Hierarchical file system element entry (distribution and target zone)

- Any apostrophes specified as part of a symbolic linkname (not the delimiters) must be doubled. Double apostrophes count as two characters in the 1023-character limit.

**SYMPATH**

specifies a list of one or more pathnames that are associated with symbolic links identified by the SYMLINK operand. The first pathname in the SYMPATH operand is associated with the first symbolic link in the SYMLINK operand, the second pathname with the second symbolic link, and so on. If there are more symbolic links listed than there are pathnames, then the last listed pathname is used for the remaining symbolic links. If more pathnames are specified than symbolic linknames, then the excess pathnames (at the end of the list) are ignored. The UCL operand is **SYMPATH**(pathname...).

- A hierarchical file system element entry that contains a SYMPATH entry must contain a matching SYMLINK entry. is specified, otherwise it must be omitted.

- A symbolic pathname can be from one to 1023 characters. Any characters in the range X'40' through X'FE' may be specified.

- A symbolic pathname can be enclosed in single apostrophes ('). A symbolic pathname must be enclosed in single apostrophes if any of the following is true:
  - The symbolic pathname contains lowercase alphabetic characters.
  - The symbolic pathname contains a character that is not uppercase alphabetic, numeric, national ($, #, or @), slash (/), plus (+), hyphen, period, or ampersand (&).
  - The symbolic pathname spans more than one line in the control statement.

The single apostrophes used to enclose a symbolic pathname (the delimiters) do not count as part of the 1023-character limit.

- Any apostrophes specified as part of a symbolic pathname (not the delimiters) must be doubled. Double apostrophes count as two characters in the 1023-character limit.

**SYSLIB**

specifies the ddname of the “target library” within a UNIX file system for the hierarchical file system element.

The UCL operand is **SYSLIB**(ddname).

- Only one SYSLIB value can be specified.
- The ddname can contain any uppercase alphabetic, numeric, or national ($, #, @) character, and can be 1 to 8 characters long.
- The SYSLIB subentry is required. Without it, SMP/E cannot process any changes for the hierarchical file system element.

**LIST Examples**

To list all the generic hierarchical file system element entries in a particular zone that support the Spanish language, you can use the following commands:

```plaintext
SET   BDY(TGT1) /* Set to requested zone. */.
LIST  HFSESP /* List all generic HFS */
      /* element entries (Spanish)*./.
```

To list specific UNIX1 element entries, you can use these commands:
Hierarchical file system element entry (distribution and target zone)

```plaintext
SET BDY(TGT1) /* Set to requested zone. */.
LIST UNIX1(UNXEL1) /* List only these */ UNXEL2 /* three */ UNXEL3) /* entries. */.
```

The format of the LIST output for each hierarchical file system element entry is the same for both of these commands. The only difference is the number of hierarchical file system element entries listed. Figure 27 shows an example of LIST output for UNIX1 element entries.

By specifying the FORFMID operand, you can reduce the number of hierarchical file system element entries listed. When FORFMID is specified, SMP/E lists a hierarchical file system element entry only if its FMID matches one of the FMIDs specified on the FORFMID operand. For example, to list hierarchical file system element entries whose FMIDs either are defined in FMIDSET HFS or are HFSFUNC, you can use these commands:

```plaintext
SET BDY(TGT1) /* Set to target zone. */.
LIST HFS /* List all hierarchical */ HFSFUNC /* for the HFS FMIDSET */.
```

You can use the LIST command to find out the names of all SYSMODs that have modified a hierarchical file system element. To include the names of these SYSMODs in the LIST output you can use the XREF operand, as shown in these commands:
Hierarchical file system element entry (distribution and target zone)

```
SET BDY(TGT1) /* Set to requested zone. */.
LIST HFS /* List all hierarchical /*
    /* file system element */
    /* entries */
XREF /* and related SYSMODs. */.
```

Notes:
1. XREF can be used either in mass mode or in select mode.
2. SMP/E obtains the data included for the XREF operand by checking all the
   SYSMOD entries for subentries for this hierarchical file system element.
   Because this data is not contained in the hierarchical file system element entry
   itself, you cannot use UCLIN to change it in the hierarchical file system element entry.

Figure 28 is an example of the LIST output produced when the XREF operand is
used.

```
PAGE nnnn - NOW SET TO zzzzzz ZONE nnnnnnn DATE mm/dd/yy TIME hh:mm:ss SMP/E 36.nn SMPLIST OUTPUT
TGT1 UNIX1 ENTRIES

NAME
UNXEL1 LASTUPD = UNXFUNC TYPE=ADD
LIBRARIES = DISTLIB=APOSIXL1 SYSLIB=UNXTGTL1
  BINARY
  FMID = UNXFUNC
  RMID = UNXPTF1
  LINK = 'linfname_1'
        = 'linfname_2'
SHSCRIPT = INSTAL01,PRE
SYSMOD HISTORY = SYSMOD TYPE DATE MCS --STATUS--
                UNXFUNC FUNCTION 07.100 UNX APP ACC
                UNXPTF1 PTF 07.120 UNX APP ACC

UNXEL2 LASTUPD = UNXFUNC TYPE=ADD
LIBRARIES = DISTLIB=APOSIXL2 SYSLIB=UNXTGTL2
  TEXT
  FMID = UNXFUNC
  RMID = UNXFUNC
  LINK = 'linfname_3'
        = 'linfname_4'
SHSCRIPT = INSTAL02,POST
SYSMOD HISTORY = SYSMOD TYPE DATE MCS --STATUS--
                UNXFUNC FUNCTION 07.100 UNX APP ACC
```

Figure 28. Hierarchical file system element entry: sample LIST output when XREF is specified

To list the UNIX shell script (SHELLSCR) element entries for a particular zone, you
can use the following commands:

```
SET BDY(TGT1) /* Set to requested zone. */.
LIST SHELLSCR /* List all UNIX shell */
    /* script element entries */.
```

Figure 29 on page 235 shows an example of the LIST output for SHELLSCR
element entries.
UNLOAD Examples

To dump the hierarchical file system element entries in UCL format, you can use the UNLOAD command. For example, to unload all the OS21 element entries in a particular zone, you can use the following commands:

```
SET BDY(TGT1) /* Set to requested zone. */.
UNLOAD OS21 /* Unload all OS21 element entries. */.
```

To unload specific OS21 element entries, you can use these commands:

```
SET BDY(TGT1) /* Set to requested zone. */.
UNLOAD OS21(OS2EL1 /* Unload only these */
OS2EL2 /* three */
OS2EL3 /* entries. */.
```

The format of the UNLOAD output for each OS21 element entry is the same for both of these commands. The only difference is the number of element entries listed. [Figure 30 on page 236] is an example of UNLOAD output for OS21 element entries.
By specifying the FORFMID operand, you can reduce the number of OS21 element entries unloaded. When FORFMID is specified, SMP/E unloads an OS21 element entry only if its FMID matches one of the FMIDs specified on the FORFMID operand. For example, to unload OS21 element entries whose FMIDs either are defined in FMIDSET OS2 or are OS2FUNC, you can use these commands:

```
SET BDY(TGT1) /* Set to target zone. */
UNLOAD OS21 /* Unload all OS21 element entries */
   FORFMID(OS21 /* for the OS21 FMIDSET */
   OS2FUNC) /* and FMID OS2FUNC. */
```

**UCLIN Examples**

You can use the ADD, REP, and DEL UCL statements to change subentries in the hierarchical file system element entry. After the UCLIN changes are made, the hierarchical file system element entry must contain at least the following subentries:

- **DISTLIB**
Hierarchical file system element entry (distribution and target zone)

- FMID
- RMID
- SYSLIB

Otherwise, there is not enough information in the entry to process the element. If any of these subentries are missing, SMP/E does not make the requested UCL updates to the entry, and the entry remains as it was before the UCL command.

The following examples are provided to help you use the hierarchical file system element entries.

**Example 1: Adding a new hierarchical file system element entry**

Defining a new hierarchical file system element entry with UCL is very seldom required; generally, hierarchical file system element entries are created from the information specified on the hierarchical file system element MCSs contained in the SYMODYCs when the SYMODYCs are installed. If, however, you want to use UCL in defining a new hierarchical file system element entry, the following is an example of the minimum information you should provide:

```plaintext
SET BDY(TGT1) /* Set to target zone. */.
UCLIN /* */.
ADD HFS(HFS01) /* Define new entry. */.
DISTLIB(AHFSLIB) /* Define DLIB. */.
SYSLIB(HFSLIB) /* target library. */.
FMID(ZUSR001) /* Functional owner (in this example a user function). */.
RMID(ZUSR001) /* Same value as FMID. */.
/* */.
ENDUCL /* */.
SET BDY(DLB1) /* Now do same to DLIB. */.
UCLIN /* */.
ADD HFS(HFS01) /* Define new HFS entry. */.
DISTLIB(AHFSLIB) /* Define DLIB. */.
SYSLIB(HFSLIB) /* target library. */.
FMID(ZUSR001) /* Functional owner (in this example a user function). */.
RMID(ZUSR001) /* Same value as FMID. */.
/* */.
ENDUCL /* */.
```

Note: If the RMID value had not been specified, it would have defaulted to the FMID value.

**Example 2: Defining a linkname for an existing hierarchical file system element**

The following shows how to inform SMP/E of new linkname for an existing hierarchical file system element:

```plaintext
SET BDY(TGT1) /* Set to target zone. */.
UCLIN /* */.
ADD HFS(HFS01) /* Existing entry. */.
LINK('..\userlk') /* New linkname. */.
/* */.
ADD HFS(HFS02) /* Existing entry. */.
LINK('myname') /* New linkname. */.
/* */.
ENDUCL /* */.
```

SET BDY(DLB1) /* Now do same thing to appropriate DLIB. */.
UCLIN /* */.
ADD HFS(HFS01) /* Existing entry. */.
LINK('..\userlk') /* New linkname. */.
/* */.
Hierarchical file system element entry (distribution and target zone)

ADD    HFS(HFS02) /* Existing entry. */
       LINK('myname') /* New linkname. */
ENDUCL /* */.

Note: UCLIN does not create a linkname in the UNIX file system; that must be
done outside of SMP/E using standard utilities. The UCLIN changes ensure
that, when the hierarchical file system element is subsequently modified, the
hierarchical file system element is updated under both its primary name and
its alternative name in the UNIX file system.

Example 3: Adding a hierarchical file system element entry with
a PARM subentry
If you want to use UCL in adding a hierarchical file system element entry with a
PARM subentry, the following example shows you how this can be done.

SET    BDY(TGT1). /* Set to target zone. */
UCLIN  /* */
ADD    HFS(HFSEL3) LASTUPD(HFSFUNC) LASTUPDTYPE(ADD)
       DISTLIB(APOSIXL2) SYSLIB(HFSTGTL2) FMID(HFSFUNC) TEXT
       RMID(HFSPTF2) LINK('linkname_5' 'linkname_6')
       PARM(This_is_another_sample_character_string_specified_as_the_value_of_PARM._It_is_a_maximum_length_character_string_for_this_subentry._I.e._it_is_300_characters_long._The_string_has_no_blanks_in_it._If_you_see_something_that_looks_like_a_blank,_it's_not._The_previous_2_characters_are_X'41'...).
ENDUCL /* */.

Hierarchical file system element entry (distribution and target zone)
HOLDDATA entry (global zone)

The HOLDDATA entry contains ++HOLD statements that either were received from SMPHOLD (external HOLDDATA) or were within a SYSMOD that was received (internal HOLDDATA). It is a record of the ++HOLD statements and is not used by SMP/E to control exception SYSMOD processing.

The HOLDDATA entry is separate from the SYSMOD entry. However, when SMP/E saves ++HOLD statements to create HOLDDATA entries, it also creates HOLDDATA subentries in the associated global zone SYSMOD entry. (These subentries appear in the LIST output as HOLDERROR, HOLDFIXCAT, HOLDSYSTEM, or HOLDUSER plus the hold reason IDs.) SMP/E uses these HOLDDATA subentries to control exception SYSMOD processing.

LIST Examples

To list all the HOLDDATA entries in a global zone, you can use the following commands:

```
SET BDY(GLOBAL) /* Set to requested zone. */.
LIST HOLDDATA /* List all HOLDDATA entries. */.
```

To list HOLDDATA entries for a specific type of hold, code one or more of the following operands: HOLDERROR, HOLDSYSTEM, HOLDUSER. Here is an example:

```
SET BDY(GLOBAL) /* Set to requested zone. */.
LIST HOLDDATA /* List HOLDDATA entries */.
HOLDSYSTEM /* for system holds */.
```

To list specific SYSMOD entries in the global zone along with the associated ++MCSs (HOLDDATA), you can use these commands:

```
SET BDY(GLOBAL) /* Set to requested zone. */.
LIST SYSMOD(UZ12345 /* List only these two */.
    UZ56789) /* entries */.
    HOLDDATA /* plus HOLDDATA for them. */.
```

The output from the LIST requests differs in the amount of information presented. If you list just the HOLDDATA entries, only the HOLDDATA itself is presented, as in Figure 31 on page 240. If you request the HOLDDATA associated with SYSMOD entries, the HOLDDATA is included in the SYSMOD entry, as in Figure 32 on page 241. (For more information about listing HOLDDATA, see the description of the HOLDDATA operand of the LIST command in SMP/E Commands).

Notes:

1. The ++HOLD statements displayed in the LIST output are exactly as received by SMP/E. They are listed in alphanumeric order by reason ID.

2. If HOLDDATA for a particular SYSMOD has been received but the SYSMOD itself has not yet been received, only the hold information is shown when the SYSMOD entry is listed. For an example, see SYSMOD UZ56789 in Figure 32 on page 241.

3. HOLDDATA entries for system holds show either INT or EXT to indicate the source of the ++HOLD statement. INT means that the ++HOLD statement was contained in the held SYSMOD and can be resolved only with the BYPASS operand on the APPLY or ACCEPT command. EXT means that the ++HOLD statement was obtained from another source, such as SMPHOLD, and can be resolved by either the BYPASS operand or a ++RELEASE statement.
4. You cannot use UCLIN to add, update, or delete HOLDDATA entries or subentries. However, you can use the REJECT command to delete HOLDDATA entries. For more information about the REJECT command, see [SMP/E Commands].

<table>
<thead>
<tr>
<th>NAME</th>
<th>HOLDFIXCAT</th>
<th>HOLDERROR</th>
<th>HOLDSYSTEM</th>
<th>HOLDSYSTEM</th>
<th>HOLDUSER</th>
<th>HOLDERROR</th>
<th>HOLDERROR</th>
<th>HOLDERROR</th>
<th>HOLDERROR</th>
</tr>
</thead>
<tbody>
<tr>
<td>UZ12345</td>
<td>AA13644</td>
<td>AZ00001</td>
<td>DOC</td>
<td>UCLIN</td>
<td>INUSE</td>
<td>AZ00023</td>
<td>AZ00024</td>
<td>AZ00001</td>
<td>AZ00002</td>
</tr>
</tbody>
</table>
### GLOBAL SYSMOD ENTRIES

<table>
<thead>
<tr>
<th>NAME</th>
<th>TYPE</th>
<th>STATUS</th>
<th>DATE/TIME REC</th>
<th>SREL</th>
<th>DELETE VER(001)</th>
<th>SUPING VER(001)</th>
<th>MAC</th>
<th>MOD</th>
<th>SRC</th>
<th>HOLDERROR</th>
</tr>
</thead>
<tbody>
<tr>
<td>UZ12345</td>
<td>PTF</td>
<td>REC</td>
<td>07.100 08:00:00</td>
<td>2038</td>
<td>HBB2102</td>
<td>AZI1111 AZI1112 AZI1113</td>
<td>MAC01 MAC02 MAC03</td>
<td>MOD01 MOD02 MOD03 MOD04</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AZ00001</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AZ00002</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AZ00003</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AZ00004</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AZ00005</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AZ00006</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AZ00007</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AZ00008</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AZ00009</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AZ00010</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AZ00011</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AZ00012</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AZ00013</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AZ00014</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AZ00015</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AZ00016</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AZ00017</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AZ00018</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AZ00019</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AZ00020</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AZ00021</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AZ00022</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AZ00023</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>AZ00024</td>
</tr>
</tbody>
</table>

**Figure 32.** HOLDDATA listed for SYSMOD entry: sample LIST output when SYSMOD is specified
The JAR entry describes a Java Archive (JAR) file that resides in a UNIX file system and a distribution library. A JAR entry is created the first time a SYSMOD is installed that contains a ++JAR MCS.

Subentries

These are the subentries for the JAR entry as they appear in the LIST output:

name

is the name of the Java ARchive element represented by the JAR entry.

The name can contain 1–8 uppercase alphabetic, numeric, or national ($, #, @) characters.

DISTLIB

specifies the ddname of the distribution library for the JAR element.

The UCL operand is DISTLIB(ddname).

- The ddname can contain any uppercase alphabetic, numeric, or national ($, #, @) character, and can be 1–8 characters long.
- The DISTLIB subentry is required. Without it, SMP/E cannot process any changes for the JAR element.

FMID

identifies the functional owner of this JAR element. The functional owner is the last function SYSMOD that replaced this JAR element.

The UCL operand is FMID(sysmod_id).

The SYSMOD ID must contain 7 uppercase alphabetic, numeric, or national ($, #, @) characters.

JARPARM

specifies a character string that is to be passed to the jar command as an option string when updating the JAR file. The maximum length of this character string is 300 bytes of data.

Any jar command options specified here will be passed to the jar command in addition to the options supplied by SMP/E, which are uvf (u indicates that the JAR file is to be updated, v produces verbose jar command output, and f indicates that the JAR file to be updated is specified on the command line rather than through stdin).

Only those jar command options that require only the option indicator to be specified are supported in the JARPARM entry, such as 0 and M. Options requiring additional input over and above the option indicator are not supported. Examples of unsupported options are the m and -C options. If such options are specified, the jar command will likely fail, since SMP/E does not prohibit their use.

LASTUPD

identifies the cause of the last change to this JAR entry.

The UCL operand is LASTUPD(value). This subentry can contain one of the following values:

UCLIN

indicates that the change was made as a result of UCLIN processing.
JAR entry (target and distribution zone)

**sysmod-id**
indicates that the change was made during the installation of the indicated SYSMOD.

The SYSMOD ID must contain 7 uppercase alphabetic, numeric, or national ($, #, @) characters.

**LASTUPDTYPE**
indicates how the entry was last changed.

The UCL operand is `LASTUPDTYPE(value)`. This subentry can contain one of the following values:

ADD The entry was added.

UPD The entry was updated.

**LINK**
specifies a list of alternative names by which this JAR element can be known in the UNIX file system.

In LIST output, linknames are always enclosed in single apostrophes. If an apostrophe is part of a linkname, it is always shown as two consecutive apostrophes in LIST output.

The UCL operand is `LINK(linkname...)`.

- The linkname can be from 1 to 1023 characters.
- A linkname can be enclosed in single apostrophes ('). A linkname must be enclosed in single apostrophes if any of the following is true:
  - The linkname contains lowercase alphabetic characters.
  - The linkname contains a character that is not uppercase alphabetic, numeric, national ($, #, or @), slash (/), plus (+), hyphen, period, or ampersand (&).
  - The linkname spans more than one line in the control statement.

The single apostrophes used to enclose a linkname (the delimiters) do not count as part of the 1023-character limit.

- Any apostrophes specified as part of a linkname (not the delimiters) must be doubled.
  Double apostrophes count as two characters in the 1023-character limit.

- The linkname can include characters X'40' through X'FE'.

**PARM**
specifies a character string that is to be passed to the copy utility as an execution-time parameter. The maximum length of this character string is 300 bytes of nonblank data. If any blanks are specified in the PARM value, they are deleted by SMP/E during processing and do not count toward the 300-byte maximum.

The UCL operand is `PARM(character_string)`.

- PARM is an optional operand.
- The character string can be entered free-form, without regard to blanks (which are compressed out of the string), and can span multiple 80-byte records.
- If parentheses are specified in the PARM value, there must always be a pair (left and right); otherwise, the results are unpredictable.

**RMID**
identifies the last SYSMOD that **replaced** this JAR element. Any subsequent
JAR entry (target and distribution zone)

SYSMOD that modifies this JAR element must have defined a relationship (such as PRE or SUP) with this SYSMOD.

The UCL operand is RMSID(sysmod_id).

- The SYSMOD ID must contain 7 uppercase alphabetic, numeric, or national ($, #, @) characters.
- If RMSID is not specified but FMID is, SMP/E sets the RMSID value to the specified FMID.

SHSCRIPT

specifies a UNIX shell script to run when this JAR element is copied to (or deleted from) a directory in the UNIX file system.

The UCL operand is SHSCRIPT(scriptname). This subentry optionally contains one or both of the following values, which specify the point in SMP/E processing when the shell script is run:

- **PRE** The shell script is run before the JAR element is copied to a UNIX file system directory.
- **POST** The shell script is run after the JAR element is copied to a UNIX file system directory.

*POST* is the default; the shell script is run after the JAR element is copied to the directory.

SYMLINK

specifies a list of one or more symbolic links, which are file names that can be used as alternate names for referring to this JAR element in the UNIX file system. Each symbolic link name listed here is associated with a path name listed in the SYMPATH entry. See the description of the SYMPATH entry for more information about how the symbolic link names and path names are associated.

The UCL operand is SYMLINK(symlinkname...).

- A JAR entry that contains a SYMLINK entry must contain a matching SYMPATH entry. SMP/E will reject any UCLIN command that would violate this condition.
- A symbolic linkname can be from one to 1023 characters. Any characters in the range X'40' through X'FE' may be specified.
- A symbolic linkname can be enclosed in single apostrophes ('). A symbolic linkname must be enclosed in single apostrophes if any of the following is true:
  - The symbolic linkname contains lowercase alphabetic characters.
  - The symbolic linkname contains a character that is not uppercase alphabetic, numeric, national ($, #, or @), slash (/), plus (+), hyphen, period, or ampersand (&).
  - The symbolic linkname spans more than one line in the control statement.

  The single apostrophes used to enclose a symbolic linkname (the delimiters) do not count as part of the 1023-character limit.
- Any apostrophes specified as part of a symbolic linkname (not the delimiters) must be doubled. Double apostrophes count as two characters in the 1023-character limit.

SYMPATH

specifies a list of one or more pathnames that are associated with symbolic links identified by the SYMLINK operand. The first pathname in the SYMPATH operand is associated with the first symbolic link in the SYMLINK operand, the second pathname with the second symbolic link, and so on. If
there are more symbolic links listed than there are pathnames, then the last listed pathname is used for the remaining symbolic links. If more pathnames are specified than symbolic linknames, then the excess pathnames (at the end of the list) are ignored. The UCL operand is \texttt{SYMPATH\{sympathname...\}).

- A JAR entry that contains a SYMPATH entry must contain a matching SYMLINK entry. is specified, otherwise it must be omitted.
- A symbolic pathname can be from one to 1023 characters. Any characters in the range X'40' through X'FE' may be specified.
- A symbolic pathname can be enclosed in single apostrophes ('). A symbolic pathname must be enclosed in single apostrophes if any of the following is true:
  - The symbolic pathname contains lowercase alphabetic characters.
  - The symbolic pathname contains a character that is not uppercase alphabetic, numeric, national ($, #, or @), slash (/), plus (+), hyphen, period, or ampersand (&).
  - The symbolic pathname spans more than one line in the control statement.
- The single apostrophes used to enclose a symbolic pathname (the delimiters) do not count as part of the 1023-character limit.
- Any apostrophes specified as part of a symbolic pathname (not the delimiters) must be doubled. Double apostrophes count as two characters in the 1023-character limit.

\texttt{SYSLIB}

specifies the ddname of the “target library” within a UNIX file system for the JAR element.

The UCL operand is \texttt{SYSLIB\{ddname\}.

- Only one SYSLIB value can be specified.
- The ddname can contain any uppercase alphabetic, numeric, or national ($, #, @) character, and can be 1 to 8 characters long.
- The SYSLIB subentry is required. Without it, SMP/E cannot process any changes for the JAR element.

\texttt{UMID}

identifies all of the SYSMODs that have updated this JAR file since it was last replaced. Any subsequent SYSMOD that updates or replaces this JAR file must have a defined relationship (such as PRE or SUP) with all SYSMODs in the UMID subentry.

\textbf{LIST Examples}

To list all the JAR entries in a particular zone, you can use the following commands:

\begin{verbatim}
SET BDY(TGT1) /* Set to zone. */.
LIST JAR /* List all JAR entries. */.
\end{verbatim}

To list specific JAR entries, you can use these commands:

\begin{verbatim}
SET BDY(TGT1) /* Set to zone. */.
LIST JAR(JAREL1, /* List only these two */
JAREL2) /* entries. */.
\end{verbatim}

The format of the LIST output for each JAR entry is the same for both of these commands. The only difference is the number of JAR entries listed.
By specifying the FORFMID operand, you can reduce the number of JAR entries listed. When FORFMID is specified, SMP/E lists a JAR entry only if its FMID matches one of the FMIDs specified on the FORFMID operand. For example, to list JAR entries whose FMIDs either are defined in FMIDSET JAR or are JARFUNC, you can use these commands:

```plaintext
SET BDY(TGT1) /* Set to target zone. */.
LIST JAR /* List all JAR entries */.
FORFMID(JAR /* for the JAR FMIDSET */.
    JARFUNC) /* and FMID JARFUNC. */.
```

You can use the LIST command to find out the names of all SYSMODs that have modified a JAR element. To include the names of these SYSMODs in the LIST output you can use the XREF operand, as shown in these commands:

```plaintext
SET BDY(TGT1) /* Set to requested zone. */.
LIST JAR /* List all JAR entries */.
XREF /* and related SYSMODs. */.
```

Notes:
1. XREF can be used either in mass mode or in select mode.
2. SMP/E obtains the data included for the XREF operand by checking all the SYSMOD entries for subentries for this JAR element. Because this data is not contained in the JAR entry itself, you cannot use UCLIN to change it in the JAR entry.

Figure 34 on page 247 is an example of the LIST output produced when the XREF operand is used.
You can use the ADD, REP, and DEL UCL statements to change subentries in the JAR entry. After the UCLIN changes are made, the JAR entry must contain at least the following subentries:

- DISTLIB
- FMID
- RMID
- SYSLIB

Otherwise, there is not enough information in the entry to process the element. If any of these subentries are missing, SMP/E does not make the requested UCL updates to the entry, and the entry remains as it was before the UCL command.

The following examples are provided to help you use the JAR entry:

**Example 1**

```
SET BDY(TGT) /* Set to target zone */.
UCLIN /* Start UCLIN processing */.
ADD JAR(ABCTTT) JARPARM(0M) /* Add JARPARMs */.
ENDUCL /* End UCLIN processing */.
```

**Example 2**

```
SET BDY(TGT) /* Set to target zone */.
UCLIN /* Start UCLIN processing */.
REP JAR(ABCTTT) JARPARM(M) /* Replace with new JARPARMs */.
UMID(OW12345,OW54321) /* and new UMIDs */.
ENDUCL /* End UCLIN processing */.
```

**Example 3**

```
SET BDY(TGT) /* Set to target zone */.
UCLIN /* Start UCLIN processing */.
DEL JAR(ABCTTT) JARPARM() /* Delete JARPARMs */.
ENDUCL /* End UCLIN processing */.
```

In the first example, a string of options is added to the JARPARM subentry in the JAR entry ABCTTT. In the second example, the existing JARPARM subentry value for ABCTTT is replaced with a new value and the UMID subentry list is replaced.
JAR entry (target and distribution zone)

with a new list of values. In the third example, the JARPARM subentry is deleted from the JAR entry for entry name ABCTTT.

UNLOAD Examples

To dump all the JAR entries in UCL format, you can use the UNLOAD command:

```
SET BDY(TGT1) /* Set to zone. */.
UNLOAD JAR /* Unload all JAR entries. */.
```

To dump specific JAR entries, you can use these commands:

```
SET BDY(TGT1) /* Set to zone. */.
UNLOAD JAR(JAREL1, /* Unload only these two */
JAREL2) /* entries. */.
```

The format of the UNLOAD output for each JAR entry is the same for both of these commands. The only difference is the number of JAR entries unloaded.

Figure 35 is an example of the output created by the UNLOAD command for a JAR entry:

```
UCLIN .
REP JAR ( ABCTTT )
LASTUPD ( HABC100 )
LASTUPDTYPE ( ADD )
DISTLIB ( AABCBIN )
SYSLIB ( SABCBIN )
FMID ( HABC100 )
RMID ( HABC100 )
UMID ( OW12345 OW54321 OW34567 )
SHSCRIPT ( ABCSCRIPT,PRE,POST )
PARM ( PATHMODE(0,6,4,4) )
JARPARM ( 0M )
SYMLINK ( '..//usr/lib/TicTacToe.jar' )
SYMPATH ( '..//usr/lpp/ttt/bin/TicTacToe.jar' )
LINK ( '..//TicTacToe.jar' )
ENDUCL.
```

Figure 35. Example UNLOAD output for JAR entry

By specifying the FORFMID operand, you can reduce the number of JAR element entries unloaded. When FORFMID is specified, SMP/E unloads an JAR element entry only if its FMID matches one of the FMIDs specified on the FORFMID operand. For example, to unload JAR element entries whose FMIDs either are defined in FMIDSET JAR or are JARFUNC, you can use these commands:

```
SET BDY(TGT1) /* Set to target zone. */.
UNLOAD JAR /* Unload all JAR element entries */
FORFMID(JAR /* for the JAR FMIDSET */
JARFUNC) /* and FMID JARFUNC. */.
```
LMOD entry (distribution and target zone)

The LMOD entry contains all the information needed to replace or update a given load module. This includes information such as whether the load module is link-edited or copied during the system generation process, any link-edit statements required to relink the load module, the link-edit attributes of the load module, and the libraries in which it resides. An LMOD entry is generally created by one of the following methods:

- **Installing a SYSMOD that adds the load module.** LMOD entries can be created when a SYSMOD is installed. SMP/E builds an LMOD entry if it encounters a ++MOD statement for a load module that does not yet have an LMOD entry and if the distribution library for the module was totally copied during system generation. For more information about copied load modules, see “DLIB entry (distribution and target zone)” on page 211 and SMP/E Commands.

- **Processing JCLIN.** LMOD entries can be created during JCLIN processing when SMP/E scans the copy and link-edit steps. At the same time, SMP/E builds MOD entries for modules that are linked or copied to the load module. For more information, see “MOD entry (distribution and target zone)” on page 275 and the “Processing” section of the JCLIN command chapter in SMP/E Commands.

**Subentries**

These are the subentries for the LMOD entry as they appear in the LIST output:

- **name**
  
  The name of the load module described by the LMOD entry.
  
  The name can contain from 1 to 8 alphanumeric characters.

- **CALLLIBS**
  
  specifies one or more DDDEF entries, existing in the same zone, that compose the SYSLIB allocation to be used when the load module is link-edited.
  
  The UCL operand is CALLLIBS(name...).
  
  - The DDDEF names can contain from 1 to 8 alphabetic (A through Z), national (@, #, or $), or numeric (0 through 9) characters. The first character must be alphabetic or national.
  
  - SMP/E does not enforce any limit on the number of names that can be specified in a CALLLIBS subentry list. However, DFSMS/MVS sets its own limits on how many data sets can be concatenated. The actual limit depends on the kind of data sets to be concatenated (partitioned data sets or PDSEs) and the total number of extents for partitioned data sets. Refer to z/OS DFSMS Using Data Sets, SC26-7410 for information about calculating this limitation.
  
  - The order in which the libraries are specified is important because it indicates the order in which the SYSLIB concatenation is built.

- **COPY**
  
  is a special SMP/E indicator meaning that the load module was copied during system generation, and that there is a one-to-one correspondence between the distribution library module (the MOD entry) and the target system load module (the LMOD entry). This information is used during APPLY processing to determine whether the LEpARM values from a ++MOD statement are applicable to the load module. It is also used during delete and compress processing to determine whether a load module can be deleted before the new modules are installed.
  
  The UCL operand is COPY.
**LMOD entry (distribution and target zone)**

**LASTUPD**
identifies the cause of the last change to this LMOD entry.

**Note:** If a given UCLIN command specifies only cross-zone subentries, this field is not changed.

The UCL operand is `LASTUPD(value)`. This subentry can contain one of the following values:

- **JCLIN** indicates that the change was made during JCLIN command processing.
- **UCLIN** indicates that the change was made as a result of UCLIN processing.

**sysmod_id** indicates that the change was made during the installation of the indicated SYSMOD. The SYSMOD did one of the following:
- Contained inline JCLIN that affected the LMOD entry.
- Renamed the load module using the `++RENAME` statement.
- Moved the load module to a new target system library using the `++MOVE` statement.

A SYSMOD that does not do any of the previously listed actions will not cause an update to the LASTUPD subentry. For example, a SYSMOD that supplies one or more modules (`++MOD`) that cause the load module to be link edited does not necessarily cause the LMOD entry's LASTUPD subentry to be updated. The LASTUPD subentry only indicates when the LMOD entry was updated, not when the module content of a load module is updated.

The SYSMOD ID must contain 7 alphanumeric characters.

**LASTUPD TYPE**
indicates how the entry was last changed.

**Note:** If a given UCLIN command specifies only cross-zone subentries, this field is not changed.

The UCL operand is `LASTUPDTYPE(value)`. This subentry can contain one of the following values:

- **ADD** The entry was added.
- **DEL** A subentry in the entry was deleted.
- **MOV** The load module was moved.
- **REN** The load module was renamed.
- **UPD** The entry was updated.

**LKED ATTRIBUTES** identifies the link-edit attributes that must be used when this load module is link-edited. SMP/E supports the following link-edit attributes. For more information, see z/OS MVS Program Management: User's Guide and Reference, SA22-7643.

**AC=1** specifies that the AC=1 parameter (which is the authorization code) is to be passed to the link-edit utility when the load module is link-edited.

The UCL operand is `AC=1`. 
ALIASES
specifies that the ALIASES parameter is to be passed to the link-edit utility when the load module is link-edited.
The UCL operand is ALIASES(ALL).

ALIGN2
specifies that the ALIGN2 parameter (alignment on a 2KB boundary) is to be passed to the link-edit utility when the load module is link-edited.
The UCL operand is ALIGN2 or ALN2.

AMODE=24
specifies that the AMODE=24 parameter is to be passed to the link-edit utility when the load module is link-edited.
The UCL operand is AMODE=24 or AMOD=24.

AMODE=31
specifies that the AMODE=31 parameter is to be passed to the link-edit utility when the load module is link-edited.
The UCL operand is AMODE=31 or AMOD=31.

AMODE=64
specifies that the AMODE=64 parameter is to be passed to the link-edit utility when the load module is link-edited.
The UCL operand is AMODE=64 or AMOD=64.

AMODE=ANY
specifies that the AMODE=ANY parameter is to be passed to the link-edit utility when the load module is link-edited.
The UCL operand is AMODE=ANY or AMOD=ANY.

AMODE=MIN
specifies that the AMODE=MIN parameter is to be passed to the link-edit utility when the load module is link-edited.
The UCL operand is AMODE=MIN or AMOD=MIN.

CALL
indicates that the CALL parameter has been specified for a load module in a JCLIN link-edit step.

Note: If the load module being link-edited contains a CALLLIBS subentry list, SMP/E uses NCAL for the link to the SMPLTS library, and ignores the CALL subentry in the LMOD entry.

For more information about link-edit parameters, see the discussion of “Link-edit utility” under the PARM subentry of UTILITY entry (global zone). Also see the APPLY command chapter in SMP/E Commands.
The UCL operand is CALL.

CASE
specifies that the CASE parameter, which controls case sensitivity and folding, is to be passed to the link-edit utility when the load module is link-edited.
The UCL operand is CASE(UPPER|MIXED).
• CASE(UPPER) and CASE(MIXED) are mutually exclusive.
COMPAT
specifies that the COMPAT parameter is to be passed to the link-edit
utility when the load module is link-edited.

The UCL operand is COMPAT=LKED | PM1 | PM2 | PM3 | PM4.

DC
specifies that the DC parameter, which is the downward compatible
attribute, is to be passed to the link-edit utility when the load module
is link-edited.

The UCL operand is DC.

DYNAM
specifies that the DYNAM parameter is to be passed to the link-edit
utility when the load module is link-edited.

The UCL operand is DYNAM(DLL).

FETCHOPT
specifies that the FETCHOPT parameter is to be passed to the link-edit
utility when the load module is link-edited.

The UCL operand is FETCHOPT(PACK | NOPACK, PRIME | NOPRIME).

FILL
specifies that the FILL parameter is to be passed to the link-edit utility
when the load module is link-edited.

The UCL operand is FILL(nn), where nn is the 2-character
representation of a single hex byte (00 - FF).

HOBSET
specifies that the HOBSET parameter is to be passed to the link-edit
utility when the load module is link-edited.

The UCL operand is HOBSET.

MAXBLK
specifies that the MAXBLK parameter is to be passed to the link-edit
utility when the load module is link-edited.

The UCL operand is MAXBLK(nnnn), where nnnn is a number
between 256 and 32760.

NE
specifies that the NE parameter, which is the noneditable attribute, is
to be passed to the link-edit utility when the load module is
link-edited.

The UCL operand is NE.

NOCALL
specifies that the NCAL parameter is to be passed to the link-edit
utility when the load module is link-edited.

Note: If the load module being link-edited contains a CALLLIBS
subentry list, SMP/E uses CALL for the link to the actual target
library, and ignores the NOCALL subentry in the LMOD entry.

For more information about link-edit parameters, see the discussion
of “Link-edit utility” under the PARM subentry of the UTILITY entry
(global zone). Also see the APPLY command chapter in SMP/E
Commands.

The UCL operand is NOCALL or NCAL.
**LMOD entry (distribution and target zone)**

**OL** specifies that the OL parameter is to be passed to the link-edit utility when the load module is link-edited.

The UCL operand is **OL**.

**OVLY** specifies that the OVLY parameter, which specifies that the load module is in overlay structure, is to be passed to the link-edit utility when the load module is link-edited.

The UCL operand is **OVLY**.

**REFR** specifies that the REFR parameter, which is the refreshable attribute, is to be passed to the link-edit utility when the load module is link-edited.

The UCL operand is **REFR**.

**RENT** specifies that the RENT parameter, which indicates that the load module is re-entrant, is to be passed to the link-edit utility when the load module is link-edited.

The UCL operand is **RENT**.

**REUS** specifies that the REUS parameter, which indicates that the load module is reusable, is to be passed to the link-edit utility when the load module is link-edited.

The UCL operand is **REUS**.

**REUS(NONE)** specifies that the REUS(NONE) parameter, which indicates that the load module cannot be reused, is to be passed to the link-edit utility when the load module is link-edited.

The UCL operand is **REUS(NONE)**.

**RMODE=24** specifies that the RMODE=24 parameter is to be passed to the link-edit utility when the load module is link-edited.

The UCL operand is **RMODE=24** or **RMOD=24**.

**RMODE=31** specifies that the RMODE=ANY parameter is to be passed to the link-edit utility when the load module is link-edited. (RMODE=31 is a synonym for RMODE=ANY.)

The UCL operand is **RMODE=31** or **RMOD=31**.

**RMODE=ANY** specifies that the RMODE=ANY parameter is to be passed to the link-edit utility when the load module is link-edited.

The UCL operand is **RMODE=ANY** or **RMOD=ANY**.

**RMODE=SPLIT** specifies that the RMODE=SPLIT parameter is to be passed to the link-edit utility when the load module is link-edited.

The UCL operand is **RMODE=SPLIT** or **RMOD=SPLIT**.

**SCTR** specifies that the SCTR parameter, which indicates that the load module can be scatter-loaded, is to be passed to the link-edit utility when the load module is link-edited.

The UCL operand is **SCTR**.
LMOD entry (distribution and target zone)

**STD**  
is a special SMP/E indication that the load module should be link-edited with none of the previously listed attributes.

STD appears in LIST and UNLOAD output in these cases:
- STD was specified on a UCL statement for this entry.
- No link-edit attributes were specified for this entry. SMP/E uses the standard system link-edit parameters.
- The LMOD entry was created by a JCLIN copy step.

If this attribute is specified when a load module is to be link-edited and no UTILITY entry is being used, SMP/E passes the default link-edit parameters to the link-edit utility. For more information about link-edit parameters, see the APPLY command chapter in [SMP/E Commands](#).

The UCL operand is **STD**.

**UPCASE**  
specifies that the UPCASE parameter, which indicates how the Binder should process symbol names, is to be passed to the link-edit utility when the load module is link-edited.

The UCL operand is **UPCASE(YES|NO)**.
- UPCASE(YES) and UPCASE(NO) are mutually exclusive.

**LKED CONTROL**  
contains all the link-edit control cards necessary to relink-edit this load module.

**Note:** Relink-editing a load module is triggered by a replacement for one of the modules within that load module. SMP/E performs only CSECT replaces when relink-editing the load module; it does not completely rebuild the load module from the distribution libraries. Therefore, it is not necessary to save all the link-edit control statements in the LMOD entry; for example, the INCLUDE statements are not saved.

The UCL operands are **++LMODIN** and **++ENDLMODIN**.
- The ++LMODIN and ++ENDLMODIN statements must start in column 1.
- No other operands can start on the same line as the ++LMODIN statement.
- The ++ENDLMODIN statement must be specified if the ++LMODIN statement is specified.
- The link-edit control statements must follow SMP/E’s coding conventions. For more information, see the “General JCLIN Coding Conventions” section of the JCLIN command chapter in [SMP/E Commands](#).
- Comment statements are ignored and are not saved in the LMOD entry. (Any link-edit control statement with an asterisk in column 1 is considered a comment statement.)

**MODDEL**  
specifies the modules that were once part of this load module but have been deleted. If the modules are reinstalled, they are linked back into this load module.

The UCL operand is **MODDEL(module...)**.

The module names can contain from 1 to 8 alphanumeric characters.

**RETURN CODE**  
specifies the maximum acceptable return code from the link edit utility for this
specific load module. If the return code from the link edit utility is higher than this value, SMP/E normally assumes the link edit operation failed.

The UCL operand is **RC(rc)**, where the rc value can be any decimal number from 0 to 16.

**SIDE DECK LIBRARY**

specifies the library to contain the definition side deck.

The definition side deck may be created as a member of a partitioned data set (PDS or PDSE), or as a file within a directory in a UNIX file system. The definition side deck is created by the link edit utility when the DYNAM(DLL) binder option is specified for DLL load modules that export symbols. If a load module does not export symbols, or if the DYNAM(DLL) option is not specified, the definition side deck is not created. The member or file name of the definition side deck is the same as the load module name. In addition, the definition side deck will have the same aliases as those specified for the load module.

The UCL operand is **SIDEDECKLIB(ddname)**, where *ddname* is the ddname of the library where the load module's definition side deck will reside.

**Notes:**

1. This subentry name appears in the LIST output as SIDE DECK LIB.
2. When this subentry value is SMPDUMMY, the SIDE DECK LIBRARY will be allocated as a DUMMY data set, and the definition side deck will not be created by the link edit utility.

**SYSTEM LIBRARIES**

specifies the target system libraries in which this load module resides.

The UCL operand is **SYSLIB(ddname...)**.

- There can be up to two SYSLIB values in an LMOD entry.
- The ddnames can contain from 1 to 8 alphanumeric characters.

**Note:** This subentry name appears in the LIST output as SYSTEM LIBRARY.

**UTILITY INPUT**

specifies one or more files to be included when this load module is link-edited. The ddname of the library where the file resides is also indicated.

The UCL operand is **UTIN((name,ddname)...)**, where *name* is a data set member name or relative UNIX filename, and *ddname* is the ddname of the library where the file resides.

- The *name* can be from 1 to 1023 characters in length.
- The *name* can include any nonblank characters X'41' through X'FE'.
- The *name* cannot begin or end with a slash ('/') because any UNIX filename specification must be relative to the directory associated with the *ddname*.
- A *name* can be enclosed in single quotation marks ('). A *name* value must be enclosed in single quotation marks if any of the following are true:
  - The *name* contains a character that is not uppercase alphabetic, numeric, national ($,#, or @), slash (/), plus (+), hyphen (-), period (.) or ampersand (&).
  - The *name* spans more than one line in the UCLIN statement.

The single quotation marks used to enclose a *name* (the delimiters) do not count as part of the 1023-character limit.
LMOD entry (distribution and target zone)

- Any quotation marks specified as part of a name (not the delimiters) must be doubled. Double quotation marks count as two characters in the 1023 character limit.
- If the name value spans lines, it must extend through column 72 of the first line and begin in column 1 of the subsequent line.

The combination of name and ddname determines the uniqueness of a subentry in the UTILITY INPUT subentry list. Only one subentry value for a given name and ddname combination is saved in the UTILITY INPUT subentry list.

XZMOD

specifies one or more modules that were added to the load module by the LINK MODULE command. The name of the zone supplying each module is also indicated.

The UCL operand is XZMOD((module,zone)...).
- The zone name specified for an XZMOD subentry cannot match the name of the set-to zone.
- The XZMOD subentry is added to an LMOD entry automatically during LINK MODULE command processing. However, it is almost never removed automatically, except during JCLIN processing. For more information, see the “Cross-Zone Relationships” section of the JCLIN command chapter in SMP/E Commands.
- An LMOD entry can contain XZMOD and XZMODP subentries without any other subentries. Such an LMOD entry represents a stub load module, which is retained after a load module with cross-zone relationships is deleted. If the deleted load module is later reinstated, SMP/E uses the information in the stub load module to issue messages reminding the user to use the LINK MODULE command to restore those previous cross-zone relationships, if they are still appropriate.
- If UCLIN is used to update an existing LMOD entry and only cross-zone subentries (XZMOD and XZMODP) are changed, SMP/E does not update the LASTUPD and LASTUPDTYPE subentries.

XZMODP

indicates that the load module contains one or more modules from another zone and that XZMOD subentries exist in this LMOD entry.

The UCL operand is XZMODP.
- You never need to specify the XZMODP subentry on a UCL statement. SMP/E automatically determines the setting of XZMODP, according to whether the LMOD entry contains XZMOD subentries.
- You cannot add the XZMODP subentry to an LMOD entry that does not contain XZMOD subentries.
- You cannot delete the XZMODP subentry from an LMOD entry containing XZMOD subentries.
- An LMOD entry can contain XZMOD and XZMODP subentries without any other subentries.
- If UCLIN is used to update an existing LMOD entry and only cross-zone subentries are changed (XZMOD and XZMODP), SMP/E does not update the LASTUPD and LASTUPDTYPE subentries.

LIST Examples

To list all the LMOD entries in a particular zone, you can use the following commands:
LMOD entry (distribution and target zone)

```
SET BDY(TGT1) /* Set to requested zone. */.
LIST LMOD /* List all LMOD entries. */.
```

To list specific LMOD entries, you can use these commands:
```
SET BDY(TGT1) /* Set to requested zone. */.
LIST LMOD(LMOD01 /* List only these two */
  LMOD02) /* entries. */.
```

The format of the LIST output for each LMOD entry is the same for both of these commands. The only difference is the number of LMOD entries listed. Figure 36 is an example of LIST output for LMOD entries.

Figure 36. LMOD entry: sample LIST output (Part 1 of 2)
You can use the LIST command to find out the names of all the distribution library modules that are link-edited into this load module. To include the names of these modules in the LIST output you can use the XREF operand, as shown in these commands:

```
SET BDY(TGT1) /* Set to requested zone. */.
LIST LMOD /* List all LMOD entries */.
XREF /* and MODs in them. */.
```

**Notes:**
1. XREF can be used either in mass mode or in select mode.
2. SMP/E obtains the data included for the XREF operand by checking the LMOD subentries in all the MOD entries. Because this data is not contained in the LMOD entry itself, you cannot use UCLIN to change it in the LMOD entry.

Figure 37 on page 259 is an example of the LIST output produced when the XREF operand is used.
### LMOD entry (distribution and target zone)

<table>
<thead>
<tr>
<th>LMOD entry</th>
<th>System Library</th>
<th>LKED Attributes</th>
<th>LKED Control</th>
<th>Modules</th>
</tr>
</thead>
<tbody>
<tr>
<td>BPXLMOD01</td>
<td>BPXLIB1</td>
<td>RENT, CASE(MIXED)</td>
<td>ENTRY BPXMOD01</td>
<td>NAME FMID</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ALIAS '.../FRIENDLY/NAME/THAT/DOES/NOT/NEED/QUOTES'</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ALIAS '.../friendly/name/with a comma/which/requires/quotes'</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ALIAS '.../friendly/name/with/a''/which/requires/quotes'</td>
<td></td>
</tr>
<tr>
<td>LMOD01</td>
<td>SLIB01</td>
<td>COPY, RENT, REUS, SCTR, OVLY, EFR, DC, NE, AC=1, AMODE=24, RMODE=24</td>
<td>ENTRY LMOD01</td>
<td>NAME FMID</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MODULE2 HXZ1234</td>
<td></td>
</tr>
<tr>
<td>LMOD02</td>
<td>SLIB02</td>
<td>COPY, RENT, REUS, SCTR, OVLY, EFR, DC, NE, AC=1, AMODE=24, RMODE=24</td>
<td>ENTRY LMOD02</td>
<td>NAME FMID</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MODULE2 HXZ1234</td>
<td></td>
</tr>
<tr>
<td>LMOD03</td>
<td>LINKLIB</td>
<td>RENT, REUS, AC=1</td>
<td>ALIAS MOD3</td>
<td>NAME FMID</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>*** CHANGE/REPLACE STMTS FOR MOD03 FROM DLIB AOS12</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CHANGE MOD03C1(NEW03C1)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CHANGE MOD03C2(NEW03C2)</td>
<td></td>
</tr>
<tr>
<td>LMOD04</td>
<td>LINKLIB</td>
<td>STD</td>
<td>ENTRY LMOD04</td>
<td>NAME FMID</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ALIAS LMOD04A1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MODULE4 HXZ1234</td>
<td></td>
</tr>
<tr>
<td>LMOD05</td>
<td>LINKLIB</td>
<td></td>
<td></td>
<td>NAME FMID</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>MODULE4 HXZ1234</td>
</tr>
</tbody>
</table>

---

Figure 37. LMOD entry: sample LIST output when XREF is specified (Part 1 of 2)
**UNLOAD Examples**

To dump the LMOD entries in UCL format, you can use the UNLOAD command. To unload all the LMOD entries in a particular zone, you can use the following commands:

```plaintext
SET BDY(TGT1) /* Set to requested zone. */.
UNLOAD LMOD /* Unload all LMOD entries. */.
```

To unload specific LMOD entries, you can use these commands:

```plaintext
SET BDY(TGT1) /* Set to requested zone. */.
UNLOAD LMOD(LMOD01 /* Unload only these two */
LMOD02) /* entries. */.
```

The format of the UNLOAD output for each LMOD entry is the same for both of these commands. The only difference is the number of LMOD entries listed. Figure 38 on page 261 is an example of UNLOAD output for LMOD entries.
LMOD entry (distribution and target zone)

UCLIN.
REP LMOD ( DL1LMOD )
LASTUPD ( DL1F001 )
LASTUPDTYPE ( ADD )
SYSLIB ( LINKLIB )
SIDEDECKLIB ( SGOSSD )
/* LEPARM */ RENT REUS RMOD=ANY DYNAM(DLL)
CALLLIBS ( SCEELOAD SEDCBASE )
/* UTILITY INPUT */
UTIN ( { GOS1MOD1 SGOSSD } { GOS1MOD2 SGOSSD } )
++LMODIN
ENTRY DL1MOD1
++ENDLMODIN.

REP LMOD ( LMOD01 )
LASTUPD ( JXY1102 )
LASTUPDTYPE ( ADD )
SYSLIB ( LINKLIB )
RC ( 4 )
/* LEPARM */ STD
MODEL ( MOD05 MOD06 )
++LMODIN
ENTRY MOD01
ALIAS LMOD01A1
++ENDLMODIN.

REP LMOD ( LMOD02 )
LASTUPD ( JXY1102 )
LASTUPDTYPE ( ADD )
SYSLIB ( LINKLIB PPLIB01 )
/* LEPARM */ RENT REUS AC=1
++LMODIN
ENTRY MOD03
ALIAS MOD03
++ENDLMODIN.

REP LMOD ( LMOD03 )
LASTUPD ( JXY1102 )
LASTUPDTYPE ( ADD )
SYSLIB ( LINKLIB )
/* LEPARM */ RENT REUS AC=1
++LMODIN
ENTRY MOD03
ALIAS MOD03
CHANGE MOD03C1(NEW03C1)
CHANGE MOD03C2(NEW03C2)
INCLUDE AOS12 (MOD03 )
++ENDLMODIN.

Figure 38. LMOD entry: sample UNLOAD output (Part 1 of 3)
LMOD entry (distribution and target zone)

```
REP LMOD ( LMOD04 )
LASTUPD ( LINK )
LASTUPDTYPE ( UPD )
SYSLIB ( LINKLIB )
RC ( 4 )
/* LEPARM */ STD
/* CROSS-ZONE*/ XZMODP
XZMOD ( ( CICSMOD1 CICS1 ) ( CICSMOD2 CICS1 ) ( IMSMOD1 IMS1 )
++LMODIN
ENTRY MOD01
ALIAS LMOD04A1
++ENDLMODIN

REP LMOD ( LMOD05 )
/* CROSS-ZONE*/ XZMODP
XZMOD ( ( CICSMOD1 CICS1 ) ( CICSMOD2 CICS1 ) ( IMSMOD1 IMS1 )
REP LMOD ( LMOD06 )
LASTUPD ( JCLIN )
LASTUPDTYPE ( ADD )
SYSLIB ( LINKLIB )
/* LEPARM */ SCTR MAXBLK(6160) AC=1 AMOD=MIN
REP LMOD ( LMOD07 )
LASTUPD ( JCLIN )
LASTUPDTYPE ( ADD )
SYSLIB ( LINKLIB )
MODELL ( ISPLINK MOD01 )
/* LEPARM */ FETCHOPT(PACK,PRIME) NOCALL
REP LMOD ( LMOD08 )
LASTUPD ( HXY0001 )
LASTUPDTYPE ( ADD )
SYSLIB ( APPLLIB )
/* LEPARM */ RENT REUS
CALLLIBS ( PLIBASE CSSLIB )
```

Figure 38. LMOD entry: sample UNLOAD output (Part 2 of 3)
UCLIN Examples

You can use the ADD, REP, and DEL UCL statements to change subentries in the LMOD entry. When you use UCLIN to update an LMOD entry, keep these points in mind:

- After the UCLIN changes are made, the LMOD entry must contain at least a SYSLIB subentry. Otherwise, there is not enough information in the entry to indicate where the load module should be installed.
- The input following the ++LMODIN statement replaces all existing link-edit control cards in the LMOD entry. This is different from JCLIN processing, where all control cards are replaced except CHANGE and REPLACE, which are merged with the existing CHANGE and REPLACE control cards.
- If a DEL statement deletes all the existing subentries in the entry, SMP/E deletes the entire entry.
- When SMP/E processes a DEL statement, it does not compare any control cards after the ++LMODIN statement with the control cards that are currently in the LMOD entry. It deletes all the existing control cards.

The following examples are provided to help you use the LMOD entry.

Example 1: Adding a new LMOD entry

The correct method of adding a new LMOD entry is through JCLIN, so that in addition to the LMOD entry being created, SMP/E also ensures that all related MOD entries are updated. If, however, you want to define a new load module entry using UCL, the following is an example of the minimum information you should provide.

```plaintext
SET BDY(TGT1) /* Set to target zone. */
UCLIN /* */
ADD LMOD(LMOD001) /* Define new LMOD entry. */
SYSLIB(LPALIB) /* System library 1. */
PPLIB01 /* System library 2. */
RENT REUS AC=1 /* All link attributes. */
++LMODIN /* All link-edit stmts. */
INCLUDE AOS12(MOD01)
```
LMOD entry (distribution and target zone)

INCLUDE A0S12(MOD02)
ENTRY MOD01
ALIAS LMOD01A1
NAME LMOD01(R)
++ENDLMODIN

/* */.
REP MOD(MOD01) /* Now fix MOD entries. */
DISTLIB(AOS12) /* MOD DLIB. */
FMID(FXY1102) /* Functional owner. */
LMOD(LMOD01) /* Connect to LMOD. */
/* */.
REP MOD(MOD02) /* Now fix MOD entries. */
DISTLIB(AOS12) /* MOD DLIB. */
FMID(FXY1102) /* Functional owner. */
LMOD(LMOD01) /* Connect to LMOD. */
/* */.
ENDUCL /* */.

Note: In this example, the entire set of control cards needed to link the load module was specified. MOD entries for modules within the load module were also updated.

Example 2: Changing the link-edit attributes of an LMOD
As in the previous example, the correct way to change any part of a LMOD entry is through the use of JCLIN. However, at times, assuming a knowledge of SMP/E and UCL processing, you may decide to use UCL to do it. For this example, assume you have a load module, LMOD99, with link-edit attributes of RENT REUS, and that you have made changes so that it is no longer reentrant but is now authorized and will fill uninitialized areas. This change can be made as follows:

SET BDY(TGT1) /* Set to target zone. */
UCLIN /* */.
DEL LMOD(LMOD99) /* Existing LMOD entry. */
RENT. /* Delete RENT attribute. */
ADD LMOD(LMOD99) /* Existing LMOD entry. */
AC=1 FILL(00) /* Add authorization, */
/* and fill uninitialized */
/* areas to hex byte 00. */
ENDUCL /* */.

Example 3: Deleting a MODDEL subentry
Suppose load module LMOD03 for product A included module MOD01 from product B, and both products are installed in zone ZOSZN. Product B deleted MOD01 without replacing it, and as a result, SMP/E created a MODDEL subentry for MOD01 in the LMOD entry for LMOD03. You have decided to reintroduce your own version of MOD01, but do not want it relinked into LMOD03. To prevent this relinking, delete the MODDEL subentry from the LMOD entry, as shown below:

SET BDY(ZOSZN) /* Set to zone. */
UCLIN /* */.
DEL LMOD(LMOD03) /* Identify LMOD entry. */
MODDEL(MOD01) /* Delete MODDEL subentry. */
ENDUCL /* */.

Example 4: Completing cross-zone updates
If SMP/E could not complete cross-zone updates for APPLY or RESTORE processing, you may need to use UCLIN to make the remaining changes. (SMP/E issues messages and reports to tell you what cross-zone work you may need to finish.) Here are some examples of using UCLIN to change XZMOD subentries in
LMOD entries. Make sure you check the messages and reports to determine whether any additional changes are needed for cross-zone subentries in MOD entries or TARGETZONE entries.

- **Adding XZMOD subentries:** This example adds module IGCXXX from zone ZXXX and module IGCABC from zone ZABC to the IEANUC01 LMOD entry. If this LMOD entry previously had no XZMOD subentries, SMP/E automatically sets XZMODP to indicate that the LMOD entry now contains XZMOD subentries.

  ```
  SET BDY(ZOSZN) /* Set to z/OS zone */.
  UCLIN /* start UCLIN processing */.
  ADD LMOD(IEANUC01) /* identify LMOD entry */
  XZMOD((IGCXXX,ZXXX), /* add module from ZXXX zone */
  (IGCABC,ZABC)) /* add module from ZABC zone */.
  ENDUCL /* end UCLIN processing */.
  ```

- **Replacing XZMOD subentries:** This example replaces all the XZMOD subentries in LMOD entry CICSDIAG with a single subentry for module ISPLINK from zone ZOSZN. If this LMOD entry previously had no XZMOD subentries, SMP/E automatically sets XZMODP to indicate that the LMOD entry now contains XZMOD subentries.

  ```
  SET BDY(CICS1) /* set to CICS zone */.
  UCLIN /* start UCLIN processing */.
  REP LMOD(CICSDIAG) /* identify LMOD entry */
  XZMOD((ISPLINK,ZOSZN)) /* replace list with one value */.
  ENDUCL /* end UCLIN processing */.
  ```

- **Deleting XZMOD subentries:** This example shows how to delete a single XZMOD subentry or all XZMOD subentries.
  - For LMOD entry IMSDIAG1, one XZMOD subentry is deleted. If no XZMOD values are left, SMP/E turns the XZMODP indicator off to indicate that the LMOD entry no longer has XZMOD subentries.
  - For LMOD entry IMSDIAG2, all XZMOD subentries are deleted. SMP/E turns the XZMODP indicator off to indicate that the LMOD entry no longer contains XZMOD subentries.

  ```
  SET BDY(IMS1) /* set to IMS zone */.
  UCLIN /* start UCLIN processing */.
  DEL LMOD(IMSDIAG1) /* identify LMOD entry */
  XZMOD((ISPLINK,ZOSZN)) /* delete ISPLINK reference */.
  DEL LMOD(IMSDIAG2) /* identify LMOD entry */
  XZMOD() /* delete entire list */.
  ENDUCL /* end UCLIN processing */.
  ```

**Example 5: Adding a CALLLIBS subentry list to an LMOD entry**

You can use either the ADD or the REP statement to add a CALLLIBS subentry list to an LMOD entry, depending on whether the CALLLIBS already exists or not. The order in which the libraries are specified is important because it indicates the order in which the SYSLIB concatenation is built. This is shown in the examples that follow.

- **Adding a CALLLIBS subentry:** This example adds a CALLLIBS subentry list containing PLIBASE and APPLIB to LMOD entry LMOD04.

  ```
  SET BDY(ZOSZN) /* set to z/OS zone */.
  UCLIN /* start UCLIN processing */.
  ADD LMOD(LMOD04) /* identify LMOD entry */
  CALLLIBS(PLIBASE,APPLIB) /* add CALLLIBS subentry */.
  ENDUCL /* end UCLIN processing */.
  ```

- **Adding to an existing CALLLIBS subentry:** Suppose LMOD entry LMOD05 has a CALLLIBS subentry list containing PLIBASE and APPLIB, and CSSLIB is to be added to this list. The entire CALLLIBS subentry must be replaced as shown below:

  ```
  SET BDY(ZOSZN) /* set to z/OS zone */.
  UCLIN /* start UCLIN processing */.
  ADD LMOD(LMOD05) /* identify LMOD entry */
  CALLLIBS(PLIBASE,APPLIB,CSSLIB) /* add CALLLIBS subentry */.
  ENDUCL /* end UCLIN processing */.
  ```
Example 6: Deleting link-edit control statements
Suppose you have installed a SYSMOD that was packaged incorrectly and added unwanted link-edit control statements to the LMOD entry for an existing load module. As a result, you now need to delete those statements from the LMOD entry. Here is an example of the UCLIN statements you can use to make this change:

```plaintext
SET BDY(ZOSZN) /* set to z/OS zone */.
UCLIN /* start UCLIN processing */.
DEL LMOD(LMODX2) /* identify LMOD entry */.
++LMODIN /* delete all link-edit */.
++ENDLMODIN /* control statements. */.
ENDUCL /* end UCLIN processing */.
```

Using the REP command instead of the DEL command gives you the same result. There is no need to specify the control statements to be deleted. In fact, you cannot selectively delete link-edit control statements from an LMOD entry—you can only delete them all.

Example 7: Adding a UTIN and SIDEDECKLIB subentry to an LMOD
In this example, a utility input subentry list, containing DSOMCMP1 from library SDSOMSD and DSOMCMP2 from library SDSOMSD, is added to the LMOD entry for load module LMOD1. In addition, a side deck library subentry of SGOSSD is added to the LMOD entry for load module LMOD5.

```plaintext
SET BDY(TGT) /* Set to target zone. */.
UCLIN /* Start UCLIN processing. */.
ADD LMOD(LMOD1) /* Identify LMOD entry. */.
UTIN((DSOMCMP1,SDSOMSD)) /* Add utility input... */.
(DSOMCMP2,SDSOMSD)) /* ...subentry list. */.
ADD LMOD(LMOD5) /* Identify LMOD entry. */.
SIDEDECKLIB(SGOSSD) /* Add side deck library. */.
ENDUCL /* End UCLIN processing. */.
```

Example 8: Deleting a UTIN subentry from an LMOD
In this example, DSOMCMP1 is deleted from the utility input subentry for LMOD entry LMOD2, and the entire utility input subentry list is removed from LMOD entry LMOD3.

```plaintext
SET BDY(TGT) /* Set to target zone. */.
UCLIN /* Start UCLIN processing. */.
DEL LMOD(LMOD2) /* Identify LMOD entry. */.
DTIN((DSOMCMP1,SDSOMSD)) /* Remove DSOMCMP1 from */.
(ENDLMODIN) /* utility input subentry. */.
DEL LMOD(LMOD3) /* Identify LMOD entry. */.
UTIN() /* Delete all utility input. */.
ENDUCL /* End UCLIN processing. */.
```
MAC entry (distribution and target zone)

The MAC entry describes a macro that exists in the distribution or target macro libraries. A MAC entry is generally created by one of the following methods:

- **Installing a SYMMOD that contains the macro.** MAC entries are created the first time you install a SYMMOD containing a ++MAC statement for a macro that does not yet have a MAC entry.

- **Processing JCLIN.** MAC entries can be built during JCLIN processing when SMP/E scans the assembler statements for inline assemblies to determine which macros are used in that assembly. The name of the assembly is kept in the MAC entry, so when the macro is updated, SMP/E can reassemble the required modules.

MAC entries can also be built when SMP/E scans copy steps and finds a SELECT statement that specifies TYPE=MAC.

For additional information, see the “Processing” section of the JCLIN command chapter in [SMP/E Commands](#).

SMP/E records the function and service level of each macro in the MAC entry, as well as information about how that macro affects the structure of the distribution or target libraries and modules.

Once a MAC entry exists for a macro, it is updated as subsequent SYMMODs are installed.

**Subentries**

These are the subentries for the MAC entry as they appear in the LIST output:

- **name**
  - is the name of the macro represented by the MAC entry.
  - The name can contain from 1 to 8 alphanumeric characters and $, #, @, or hex C0.

- **DISTLIB**
  - is the ddname of the macro distribution library.
  - The UCL operand is DISTLIB(ddname).
  - The ddname can contain from 1 to 8 alphanumeric characters.
  - The DISTLIB subentry is not required when the MAC entry is first defined.
    - For example, during JCLIN processing SMP/E builds MAC entries without a DISTLIB value. However, SMP/E can add a DISTLIB value later when it processes the ++MAC statement. A DISTLIB value is needed at some time to process any changes for the macro.

- **FMID**
  - identifies the functional owner of this macro. The functional owner is the last function SYMMOD that replaced this macro.
  - The UCL operand is FMID(sysmod_id).
  - The SYSMOD ID must contain 7 alphanumeric characters.

- **GENASM**
  - identifies those assemblies that have to be done during APPLY each time this macro is modified. These assemblies must exist as either ASSEM or SRC entries in the target zone.
  - The UCL operand is GENASM(name...).
MAC entry (distribution and target zone)

The names can contain from 1 to 8 alphanumeric characters.

**LASTUPD**
identifies the cause of the last change to this MAC entry.

The UCL operand is **LASTUPD(value)**. This subentry can contain one of the following values:

**JCLIN**
indicates that the change was made during JCLIN command processing.

**UCLIN**
indicates that the change was made as a result of UCLIN processing.

**sysmod-id**
indicates that the change was made during the installation of the indicated SYSMOD.

The SYSMOD ID must contain 7 alphanumeric characters.

**LASTUPD TYPE**
indicates how the entry was last changed.

The UCL operand is **LASTUPDTYPE(value)**. This subentry can contain one of the following values:

**ADD**  The entry was added.
**MOV**  The entry was moved.
**UPD**  The entry was updated.

**MALIAS**
identifies any alias names for this macro.

The UCL operand is **MALIAS(name...)**.

The alias names can contain from 1 to 8 alphanumeric characters.

**RMID**
identifies the last SYSMOD that replaced this macro. Any subsequent SYSMOD that modifies this macro must have a defined relationship (such as PRE or SUP) with this SYSMOD.

The UCL operand is **RMID(sysmod_id)**.

- The SYSMOD ID must contain 7 alphanumeric characters.
- If RMID is not specified, but **FMID** is, SMP/E sets the RMID value to the specified FMID.

**SYSLIB**
is the ddname of the target system macro library.

The UCL operand is **SYSLIB(ddname)**.

- Only one SYSLIB value may be specified.
- The ddname can contain from 1 to 8 alphanumeric characters.

**UMID**
identifies all those SYSMODs that have updated this macro since it was last replaced. Any subsequent SYSMOD that modifies this macro must have a defined relationship (such as PRE or SUP) with all these SYSMODs.

The UCL operand is **UMID(sysmod_id...)**.

The SYSMOD ID must contain 7 alphanumeric characters.
LIST Examples

To list all the MAC entries in a particular zone, you can use the following commands:

```
SET BDY(TGT1) /* Set to requested zone. */.
LIST MAC /* List all MAC entries. */.
```

To list specific MAC entries, you can use these commands:

```
SET BDY(TGT1) /* Set to requested zone. */.
LIST MAC(MAC01 /* List only these two */
    MAC02) /* entries. */.
```

The format of the LIST output for each MAC entry is the same for both of these commands. The only difference is the number of MAC entries listed. Figure 39 shows an example of LIST output for MAC entries.

```
PAGE nnnn - NOW SET TO zzzzz ZONE nnnnnnn DATE mm/dd/yy TIME hh:mm:ss SMP/E 36.nn SMPLIST OUTPUT

TGT1 MACRO ENTRIES

NAME

MAC01 LASTUPD = JXY1002 TYPE=ADD
  LIBRARIES = DISTLIB=DLIBMAC1 SYSLIB=MACLIB01
  FMID = JXY1102
  RMID = JXY1102
  MALIAS = TERMINAL TERM T
  GENASM = ASSEM01 ASSEM02 SRC01 SRC02

MAC02 LASTUPD = JXY1000 TYPE=UPD
  LIBRARIES = DISTLIB=DLIBMAC1 SYSLIB=MACLIB01
  FMID = JXY1121
  RMID = UZ00010
  UMID = UZ00014 UZ00015
  GENASM = ASSEM01 SRC02
```

Figure 39. MAC entry: sample LIST output

By specifying the FORFMID operand, you can reduce the number of MAC entries listed. When FORFMID is specified, SMP/E lists a MAC entry only if its FMID matches one of the FMIDs specified on the FORFMID operand. For example, to list MAC entries whose FMIDs either are defined in FMIDSET TP or are JXY1102, you can use these commands:

```
SET BDY(TGT1) /* Set to target zone. */.
LIST MAC /* List all macro entries */
    FORFMID(TP /* for the TP FMIDSET */
      JXY1102) /* and FMID JXY1102. */.
```

You can use the LIST command to find out the names of all SYSMODs that have modified a macro. To include the names of these SYSMODs in the LIST output, you can use the XREF operand, as shown in these commands:

```
SET BDY(TGT1) /* Set to requested zone. */.
LIST MAC /* List all macro entries */
    XREF /* and related SYSMODs. */.
```

Notes:

1. XREF can be used either in mass mode or in select mode.
SMP/E obtains the data included for the XREF operand by checking for MAC and MACUPD entries for this macro in all the SYSMOD entries. Because this data is not contained in the MAC entry itself, you cannot use UCLIN to change it in the MAC entry.

Figure 40 is an example of the LIST output produced when the XREF operand is used.

Figure 40. MAC entry: sample LIST output when XREF is specified

### UNLOAD Examples

To dump the MAC entries in UCL format, you can use the UNLOAD command. To unload all the MAC entries in a particular zone, you can use the following commands:

```sh
SET BDY(TGT1) /* Set to requested zone. */.
UNLOAD MAC /* Unload all MAC entries. */.
```

To unload specific MAC entries, you can use these commands:

```sh
SET BDY(TGT1) /* Set to requested zone. */.
UNLOAD MAC(MAC01) /* Unload only these two */.
UNLOAD MAC(MAC02). /* entries. */.
```

The format of the UNLOAD output for each MAC entry is the same for both of these commands. The only difference is the number of MAC entries listed. Figure 41 on page 271 is an example of UNLOAD output for MAC entries.
By specifying the FORFMID operand, you can reduce the number of MAC entries unloaded. When FORFMID is specified, SMP/E unloads a MAC entry only if its FMID matches one of the FMIDs specified on the FORFMID operand. For example, to unload MAC entries whose FMIDs either are defined in FMIDSET TP or are JXY1102, you can use these commands:

```
SET BDY(TGT1) /* Set to target zone. */.
UNLOAD MAC /* Unload all macro entries */.
FORFMID(TP /* for the TP FMIDSET */.
JXY1102) /* and FMID JXY1102. */.
```

**UCLIN Examples**

You can use the ADD, REP, and DEL UCL statements to change subentries in the MAC entry. After the UCLIN changes are made, the MAC entry must contain at least the following subentries:

- FMID
- RMID

Otherwise, there is not enough information in the entry to process the macro. If any of these subentries are missing, SMP/E does not make the requested UCL updates to the entry, and the entry remains as it was before the UCL command.

The following examples are provided to help you use the MAC entry.

**Example 1: Adding a new MAC entry**

Defining a new macro entry with UCL is very seldom required; generally, MAC entries are created from the information specified on the ++MAC statements contained in the SYSMODs when SYSMODs are installed. If, however, you want to use UCL in defining a new macro entry, the following is an example of the minimum information you should provide:

```
SET BDY(TGT1) /* Set to target zone. */.
UCLIN /* */.
ADD MAC(MAC01) /* Define new macro entry. */.
DISTLIB(AAMACLIB) /* Define DLIB, */.
SYSLIB(MACLIB) /* system library */.
```

**MAC entry (distribution and target zone)**

```
UCLIN.
REP MAC ( MAC01 )
LASTUPD ( JXY1102 )
LASTUPDTYPE ( ADD )
DISTLIB ( DLIBMAC1 )
SYSLIB ( MACLIB01 )
FMID ( JXY1102 )
RMID ( JXY1102 )
MALIAS ( TERMINAL TERM T )
GENASM ( ASSEMB1 ASSEMB2 SRC01 SRC02 )
.
REP MAC ( MAC02 )
LASTUPD ( JXY1121 )
LASTUPDTYPE ( UPD )
DISTLIB ( DLIBMAC1 )
SYSLIB ( MACLIB01 )
FMID ( JXY1121 )
RMID ( UZ00010 )
UMID ( UZ00014 UZ00015 )
GENASM ( ASSEMB1 SRC02 )
.
ENDUCL.
```

Figure 41. MAC entry: sample UNLOAD output

By specifying the FORFMID operand, you can reduce the number of MAC entries unloaded. When FORFMID is specified, SMP/E unloads a MAC entry only if its FMID matches one of the FMIDs specified on the FORFMID operand. For example, to unload MAC entries whose FMIDs either are defined in FMIDSET TP or are JXY1102, you can use these commands:
MAC entry (distribution and target zone)

```
FMID(ZUSR001) /* Functional owner (in this example a user function). */
/* */.
ENDUCL /* */.
SET BDY(DLB1). /* Now do same to DLIB. */
UCLIN /* */.
ADD MAC(MAC01) /* Define new macro entry. */
DISTLIB(AMACLIB) /* Define DLIB. */
/* No SYSLIB info in DLIB. */
FMID(ZUSR001) /* Functional owner (in this example a user function). */
/* */.
ENDUCL /* */.
```

Example 2: Defining an alias for an existing macro

The following defines the method of adding an alias to an existing macro:

```
SET BDY(TGT1) /* Set to target zone. */
UCLIN /* */.
ADD MAC(MAC01) /* Existing macro entry. */
MALIAS(MAC01AL1) /* New alias name. */
/* End of adding macro. */
ENDUCL /* */.
SET BDY(DLB1) /* Now do same thing to appropriate DLIB. */
UCLIN /* */.
ADD MAC(MAC01) /* Existing macro entry. */
MALIAS(MAC01AL1) /* New alias name. */
/* End of adding macro. */
ENDUCL /* */.
```

Note: This UCL does not create an alias entry in the target or distribution libraries; that must be done outside of SMP/E using standard utilities. This ensures that, when the macro is subsequently modified, both the major entry and the alias entry are updated.
MCS entry (SMPPTS)

The MCS entry is a copy of a SYSMOD exactly as it was received from the SMPPTFIN data set. The MCS entry is in the SMPPTS data set, which is used as a warehouse for SYSMODs. When SMP/E receives a SYSMOD, it stores the SYSMOD as a separate member in the SMPPTS. The member name matches the SYSMOD ID, and each member is an MCS entry. SMP/E also creates a SYSMOD entry in the global zone to describe the SYSMOD that was received. Thus, the MCS entry and the global zone SYSMOD entry are closely related.

When SMP/E accepts or applies SYSMODs, it gets them from the MCS entries in the SMPPTS. An MCS entry is generally kept in the SMPPTS until the associated SYSMOD is accepted; then the entry is deleted.

- You may want SMP/E to save the MCS entries after ACCEPT processing, for example, if you plan to do a system generation. To do this, specify **NOPURGE** in the OPTIONS entry that is in effect during ACCEPT processing.
- Likewise, you may want to save the MCS entries after RESTORE processing. To do this, specify **NOREJECT** in the OPTIONS entry that is in effect during RESTORE processing.

Subentries

The MCS entry contains no SMP/E data and appears to the system as a member of a normal partitioned data set.

LIST Examples

To list all the MCS entries in the SMPPTS, you can use the following commands:

```
SET BDY(GLOBAL) /* Set to global. */.
LIST MCS /* List all MCS entries. */.
```

To list specific MCS entries, you can use these commands:

```
SET BDY(GLOBAL) /* Set to global. */.
LIST MCS(UZ12345, /* List only these two */
       UZ12346) /* entries. */.
```

The format of the LIST output for each MCS entry is the same for both of these commands. The only difference is the number of MCS entries listed.

**Figure 42 on page 274** is an example of LIST output for MCS entries.
As the example for UZ12346 shows, SMP/E includes inline JCLIN when listing the MCS entries.

You can use various SYSMOD-related LIST operands to limit which MCS entries are listed. For more information, see “SYSMOD entry (distribution and target zone)” on page 322 and “SYSMOD entry (global zone)” on page 336.

In addition to the LIST command, you can use standard system utility programs (such as IEBGENER, IEBPTPCH, IEHLIST, and so on) or products such as ISPF to display these entries or information about the data set.

UCLIN Examples

You cannot use UCLIN to add, update, or delete MCS entries. However, you can use the REJECT command to delete MCS entries. For more information about the REJECT command, see SMP/E Commands.

Do not use system utility programs to update MCS entries. The global zone SYSMOD entry is coordinated with the MCS entry. Any changes to the MCS entry made outside of SMP/E may get these entries out of synchronization and can result in unpredictable results when the associated SYSMOD is processed.
MOD entry (distribution and target zone)

The MOD entry describes a particular module, its function and service level, and how it relates to a load module in the target library. A MOD entry is generally created by one of the following methods:

- **Installing a SYSMOD that contains the module.** MOD entries are created the first time you install a SYSMOD that contains a ++MOD statement for a module that does not yet have a MOD entry. If the module comes from a copied distribution library, SMP/E also builds an LMOD entry with the same name. For additional information about copied libraries and the creation of LMOD entries, see “DLIB entry (distribution and target zone)” on page 211 and [SMP/E Commands](#).

A MOD entry can also be created when you install a SYSMOD that causes the assembled source to be linked to the distribution library. This can happen when a SYSMOD contains both a ++MOD and a ++SRC statement for the same module, or when the DISTMOD operand is specified on the ++SRC statement.

- **Processing JCLIN.** MOD entries can be built during JCLIN processing when SMP/E scans the link-edit and copy steps. A MOD entry built during JCLIN processing is always associated with an LMOD entry. Thus, when the module is changed, SMP/E can determine which load modules are affected. For additional information, see the “Processing” section of the JCLIN command chapter in [SMP/E Commands](#).

Subentries

These are the subentries for the MOD entry as they appear in the LIST output:

- **name**
  
  is the name of the module represented by the MOD entry.
  
  The name can contain from 1 to 8 alphanumeric characters and $, #, @, or hex C0.

- **ASSEMBLE**
  
  indicates that the source for this module must always be assembled, even if the object module is supplied in the SYSMOD.
  
  The UCL operand is **ASSEMBLE**.

- **CSECT**
  
  specifies the CSECTs present in this module.
  
  The UCL operand is **CSECT(name...)**.
  
  - The CSECT subentry is not required. However, if CSECT is missing, SMP/E assumes that the module contains only one CSECT, whose name matches the module name.
  
  - A CSECT name can contain from 1 to 8 characters.
    
    The name can contain any characters except the following:
    
    Comma ,
    
    Left parenthesis (  
    
    Right parenthesis )
    
    Blank
    
    - Comments are not allowed within a CSECT name. For example, the following is not allowed:
      
      CSECT ( /* this is a csect name */ CSECT01)
      
      The comment is interpreted as part of the CSECT name, instead of a comment.
MOD entry (distribution and target zone)

- The list of CSECT names must include all of the CSECTs in the module, even if one of the CSECTs matches the module name.
- Once a ++MOD with the CSECT operand is processed, SMP/E saves the CSECT information in the MOD entry. From then on, SMP/E uses that saved information if it is not supplied on subsequent SYSMODs.
- If a module is changed from multiple CSECTs to a single CSECT that matches the module name, the CSECT operand must be specified with one name in order to get SMP/E to store that information in the MOD entry.

DALIAS
specifies the alias name for the module, where the alias exists only in the distribution library.

The UCL operand is **DALIAS(name)**.

The DALIAS name can contain from 1 to 8 alphanumeric characters.

DISTLIB
is the ddname of the module distribution library.

The UCL operand is **DISTLIB(ddname)**.

The ddname can contain from 1 to 8 alphanumeric characters.

FMID
identifies the functional owner of this module. The functional owner is the last function SYSMOD that replaced this module.

The UCL operand is **FMID(sysmod_id)**.

- The SYMMOD ID must contain 7 alphanumeric characters.
- Some MOD entries, specifically those associated with a system generation assembly, may have no functional owner. The DISTLIB for these modules is SYSPUNCH. They may have either no FMID, or one of the following values:
  - If the module has no functional owner but was assembled during system generation, *SYSGEN* appears as the FMID.
  - If the module has no functional owner and was not assembled during system generation, ??????? appears as the FMID.

LASTUPD
identifies the cause of the last change to this MOD entry.

**Note:** If a given UCLIN command specifies only cross-zone subentries, this field is not changed.

The UCL operand is **LASTUPD(value)**. This subentry can contain one of the following values:

- **JCLIN** indicates that the change was made during JCLIN command processing.
- **UCLIN** indicates that the change was made as a result of UCLIN processing.
- **sysmod-id** indicates that the change was made during installation of the specified SYSMOD. The SYSMOD did one of the following:
  - Contained inline JCLIN that affected the module
  - Changed the distribution library for the module through the DISTLIB operand on the ++MOD statement
MOD entry (distribution and target zone)

- Added the module to an existing load module through the LMOD operand on the ++MOD statement

The SYSMOD ID must contain 7 alphanumeric characters.

LASTUPD TYPE
indicates how the entry was last changed.

Note: If a given UCLIN command specifies only cross-zone subentries, this field is not changed.

The UCL operand is LASTUPDTYPE(value). This subentry can contain one of the following values:
ADD The entry was added.
DEL A subentry in the entry was deleted.
MOV The module was moved.
UPD The entry was updated.

LKED ATTRIBUTES
identifies the link-edit attributes that must be used when this module is link-edited. SMP/E supports the following link-edit attributes. For more information, see z/OS MVS Program Management: User’s Guide and Reference, SA22-7643.

AC=1 specifies that the AC=1 parameter, which is the authorization code, is to be passed to the link-edit utility when the module is link-edited.

The UCL operand is AC=1.

ALIGN2 specifies that the ALIGN2 parameter (alignment on a 2KB boundary) is to be passed to the link-edit utility when the module is link-edited.

The UCL operand is ALIGN2 or ALN2.

AMODE=24 specifies that the AMODE=24 parameter is to be passed to the link-edit utility when the module is link-edited.

The UCL operand is AMODE=24 or AMOD=24.

AMODE=31 specifies that the AMODE=31 parameter is to be passed to the link-edit utility when the module is link-edited.

The UCL operand is AMODE=31 or AMOD=31.

AMODE=64 specifies that the AMODE=64 parameter is to be passed to the link-edit utility when the module is link-edited.

The UCL operand is AMODE=64 or AMOD=64.

AMODE=ANY specifies that the AMODE=ANY parameter is to be passed to the link-edit utility when the module is link-edited.

The UCL operand is AMODE=ANY or AMOD=ANY.

AMODE=MIN specifies that the AMODE=MIN parameter is to be passed to the link-edit utility when the module is link-edited.

The UCL operand is AMODE=MIN or AMOD=MIN.
MOD entry (distribution and target zone)

**COMPAT**
specifies that the COMPAT parameter is to be passed to the link-edit utility when the module is link-edited.

The UCL operand is **COMPAT=LKED | PM1 | PM2 | PM3 | PM4**.

**DC**
specifies that the DC parameter, which is the downward compatible attribute, is to be passed to the link-edit utility when the module is link-edited.

The UCL operand is **DC**.

**FETCHOPT**
specifies that the FETCHOPT parameter is to be passed to the link-edit utility when the module is link-edited.

The UCL operand is **FETCHOPT(PACK | NOPACK, PRIME | NOPRIME)**.

**FILL**
specifies that the FILL parameter is to be passed to the link-edit utility when the module is link-edited.

The UCL operand is **FILL(nn)**, where nn is the 2-character representation of a single hex byte (00 - FF).

**HOBSET**
specifies that the HOBSET parameter is to be passed to the link-edit utility when the module is link-edited.

The UCL operand is **HOBSET**.

**MAXBLK**
specifies that the MAXBLK parameter is to be passed to the link-edit utility when the module is link-edited.

The UCL operand is **MAXBLK(nn)**, where nn is a number between 256 and 32760.

**NE**
specifies that the NE parameter, which is the noneditable attribute, is to be passed to the link-edit utility when the module is link-edited.

The UCL operand is **NE**.

**NOCALL**
specifies that the NCAL parameter is to be passed to the link-edit utility when the module is link-edited.

The UCL operand is **NOCALL or NCAL**.

**OL**
specifies that the OL parameter is to be passed to the link-edit utility when the module is link-edited.

The UCL operand is **OL**.

**OVLY**
specifies that the OVLY parameter, which specifies that the module is in overlay structure, is to be passed to the link-edit utility when the module is link-edited.

The UCL operand is **OVLY**.

**REFR**
specifies that the REFR parameter, which is the refreshable attribute, is to be passed to the link-edit utility when the module is link-edited.

The UCL operand is **REFR**.
MOD entry (distribution and target zone)

RENT specifies that the RENT parameter, which indicates that the module is reentrant, is to be passed to the link-edit utility when the module is link-edited.

The UCL operand is RENT.

REUS specifies that the REUS parameter, which indicates that the module is reusable, is to be passed to the link-edit utility when the module is link-edited.

The UCL operand is REUS.

REUS(NONE) specifies that the REUS(NONE) parameter, which indicates that the module cannot be reused, is to be passed to the link-edit utility when the module is link-edited.

The UCL operand is REUS(NONE).

RMODE=24 specifies that the RMODE=24 parameter is to be passed to the link-edit utility when the module is link-edited.

The UCL operand is RMODE=24 or RMOD=24.

RMODE=31 specifies that the RMODE=ANY parameter is to be passed to the link-edit utility when the module is link-edited. (RMODE=31 is a synonym for RMODE=ANY.)

The UCL operand is RMODE=31 or RMOD=31.

RMODE=ANY specifies that the RMODE=ANY parameter is to be passed to the link-edit utility when the module is link-edited.

The UCL operand is RMODE=ANY or RMOD=ANY.

RMODE=SPLIT specifies that the RMODE=SPLIT parameter is to be passed to the link-edit utility when the module is link-edited.

The UCL operand is RMODE=SPLIT or RMOD=SPLIT.

SCTR specifies that the SCTR parameter, which indicates that the module can be scatter-loaded, is to be passed to the link-edit utility when the module is link-edited.

The UCL operand is SCTR.

STD is a special SMP/E indication that the module should be link-edited with none of the previously listed attributes.

When this indicator is present and a link-edit is to be done, SMP/E will pass the link-edit utility only those parameters specified in the appropriate link-edit UTILITY entry.

The UCL operand is STD.

UPCASE specifies that the UPCASE parameter, which indicates how the Binder should process symbol names, is to be passed to the link-edit utility when the module is link-edited.

The UCL operand is UPCASE(YES | NO).
• UPCASE(YES) and UPCASE(NO) are mutually exclusive.
MOD entry (distribution and target zone)

**LMOD**
specifies the names of the load modules into which this module was copied or link-edited on the target system.

The UCL operand is **LMOD(name...)**.
- The load module names can contain from 1 to 8 alphanumeric characters.
- If there are no LMOD names in a MOD entry, SMP/E assumes that the module was not selected when the SYSMOD containing the module was initially installed. Therefore, during APPLY processing, SMP/E will not link or copy the module into any target library.

**RMID**
identifies the last SYSMOD that replaced this module. Any subsequent SYSMOD that modifies this module must have a defined relationship (such as PRE or SUP) with this SYSMOD.

The UCL operand is **RMID(sysmod_id)**.
- The SYSMOD ID must contain 7 alphanumeric characters.
- If RMID is not specified, but FMID is, SMP/E sets the RMID subentry to the FMID value.
- RMID is not required for a module assembled during system generation. The DISTLIB for these modules is SYSPUNCH.

**RMIDASM**
specifies that the last replacement (RMID) to the module was done by a SYSMOD that caused an assembly of the module as a result of a source or macro modification.

The UCL operand is **RMIDASM**.

**TALIAS**
specifies one or more alias names for the module, where the alias exists in the distribution library and, for copied modules, also in the target library.

The UCL operand is **TALIAS(name...)**.

The alias names can contain from 1 to 8 alphanumeric characters.

**UMID**
identifies all those SYSMODs that have updated this module since it was last replaced. Any subsequent SYSMOD that modifies this module must have a defined relation (such as PRE or SUP) with all these SYSMODs.

The UCL operand is **UMID(sysmod_id...)**.

The SYSMOD ID must contain 7 alphanumeric characters.

**XZLMOD**
specifies one or more load modules in other zones into which this module was added by the LINK MODULE command. The name of the zone supplying each module is also indicated.

The UCL operand is **XZLMOD((load module,zone)...)**.
- The zone name specified for an XZLMOD subentry cannot match the name of the set-to zone.
- An entry can contain XZLMOD and XZLMODP subentries without any other subentries.
- If UCLIN is used to update an existing MOD entry and only cross-zone subentries are changed (XZLMOD and XZLMODP), SMP/E does not update the LASTUPD and LASTUPDTYPE subentries.
The XZLMOD subentry is added to a MOD entry automatically during LINK MODULE command processing. However, it is never automatically removed.

XZLMODP indicates that this module has been linked into one or more load modules controlled by a different target zone, and that XZLMOD subentries exist in this MOD entry.

The UCL operand is XZLMODP.

- It is never necessary to specify the XZLMODP subentry on a UCL statement. SMP/E automatically determines the setting of XZLMODP, according to whether the MOD entry contains XZLMOD subentries.
- You cannot add the XZLMODP subentry to a MOD entry that does not contain XZLMOD subentries.
- You cannot delete the XZLMODP subentry from a MOD entry containing XZLMOD subentries.
- An entry can contain XZLMOD and XZLMODP subentries without any other subentries.
- If UCLIN is used to update an existing MOD entry and only cross-zone subentries are changed (XZLMOD and XZLMODP), SMP/E does not update the LASTUPD and LASTUPDTYPE subentries.

**LIST Examples**

To list all the MOD entries in a particular zone, you can use the following commands:

```
SET BDY(TGT1) /* Set to requested zone. */.
LIST MOD /* List all MOD entries. */.
```

To list specific MOD entries, you can use these commands:

```
SET BDY(TGT1) /* Set to requested zone. */.
LIST MOD(MOD01) /* List only these two */.
MOD02 /* entries. */.
```

The format of the LIST output for each MOD entry is the same for both of these commands. The only difference is the number of MOD entries listed. The following examples show LIST output for MOD entries. Figure 43 on page 282 does not have cross-zone subentries. Figure 44 on page 283 does.
### MOD entry (distribution and target zone)

<table>
<thead>
<tr>
<th>NAME</th>
<th>LASTUPD</th>
<th>LIBRARIES</th>
<th>ASSEMBLY</th>
<th>FMID</th>
<th>RMID</th>
<th>CSECT</th>
<th>LMOD</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASSEM01</td>
<td>JCLIN</td>
<td>DISTLIB=SYSPUNCH</td>
<td>ASSEMBLY</td>
<td>ASSEM01</td>
<td>ASSEM01</td>
<td>ASSEM01</td>
<td>LMOD99A LMOD99B</td>
</tr>
<tr>
<td>ASSEM02</td>
<td>JCLIN</td>
<td>DISTLIB=SYSPUNCH</td>
<td>ASSEMBLY</td>
<td>ASSEM02</td>
<td>ASSEM02</td>
<td>ASSEM02</td>
<td></td>
</tr>
<tr>
<td>DLIMOD1</td>
<td>DLIF001</td>
<td>DISTLIB=DLIB3</td>
<td>ASSEMBLY</td>
<td>DLIF001</td>
<td>DLIF001</td>
<td>DLIF001</td>
<td>DL1LMOD</td>
</tr>
<tr>
<td>MOD01</td>
<td>JXY1102</td>
<td>DISTLIB=AOS12</td>
<td>ASSEMBLY</td>
<td>JXY1102</td>
<td>JXY1102</td>
<td>MOD01C1 MOD01C2 MOD01C3</td>
<td>LMOD001</td>
</tr>
<tr>
<td>MOD02</td>
<td>JXY1102</td>
<td>DISTLIB=AOS12</td>
<td>ASSEMBLY</td>
<td>JXY1102</td>
<td>JXY1102</td>
<td>MOD02C1</td>
<td>LMOD001 LMOD02</td>
</tr>
<tr>
<td>MOD03</td>
<td>JXY1121</td>
<td>DISTLIB=AOS12</td>
<td>ASSEMBLY</td>
<td>JXY1121</td>
<td>JXY1121</td>
<td>MOD03</td>
<td>UZ00010 UZ00014</td>
</tr>
<tr>
<td>MOD04</td>
<td>JXY1121</td>
<td>DISTLIB=AOS12</td>
<td>ASSEMBLY</td>
<td>JXY1121</td>
<td>JXY1121</td>
<td>MOD04</td>
<td>MOD04</td>
</tr>
<tr>
<td>MOD05</td>
<td>JXY0001</td>
<td>DISTLIB=DLIB1</td>
<td>ASSEMBLY</td>
<td>JXY0001</td>
<td>JXY0001</td>
<td>MOD05</td>
<td>JXY0001</td>
</tr>
<tr>
<td>MOD06</td>
<td>JXY0001</td>
<td>DISTLIB=DLIB1</td>
<td>ASSEMBLY</td>
<td>JXY0001</td>
<td>JXY0001</td>
<td>MOD06</td>
<td>JXY0001</td>
</tr>
</tbody>
</table>

Figure 43. MOD entry: sample LIST output (no cross-zone subentries)
By specifying the FORFMID operand, you can reduce the number of MOD entries listed. When FORFMID is specified, SMP/E lists a MOD entry only if its FMID matches one of the FMIDs specified on the FORFMID operand. For example, to list MOD entries whose FMIDs are either defined in FMIDSET TP or are JXY1102, you can use these commands:

```
SET BDY(TGT1) /* Set to target zone. */.
LIST MOD /* List all MOD entries */.
FOREMID(TP /* for the TP FMIDSET */.
JXY1102) /* and FMID JXY1102. */.
```

You can also use the LIST command to find out the names of all SYSMODs that have updated a module. To include the names of these SYSMODs in the LIST output, you can use the XREF operand, as shown in these commands:

```
SET BDY(TGT1) /* Set to requested zone. */.
LIST MOD /* List all module entries */.
XREF /* and related SYSMODs. */.
```

Notes:
1. XREF can be used either in mass mode or in select mode.
2. SMP/E obtains the data included for the XREF operand by checking all the SYSMOD entries to find which SYSMODs contained ++MOD or ++ZAP.
MOD entry (distribution and target zone)

statements for this module. Because this data is not contained in the MOD
entry itself, you cannot use UCLIN to change it in the MOD entry.

Figure 45 is an example of the LIST output produced when the XREF operand is
used.

<table>
<thead>
<tr>
<th>PAGE nnnn - NOW SET TO zzzzzz ZONE nnnnnn DATE mm/dd/yy TIME hh:mm:ss SMP/E 36.nn SMPLIST OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>TGT1 MODULE ENTRIES</td>
</tr>
<tr>
<td><strong>NAME</strong></td>
</tr>
<tr>
<td><strong>ASSEM01</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>ASSEM02</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>MOD01</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>MOD02</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>MOD03</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>MOD04</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

Figure 45. MOD entry: sample LIST output when XREF is specified
MOD entry (distribution and target zone)

UNLOAD Examples

To dump the MOD entries in UCL format, you can use the UNLOAD command. To unload all the MOD entries in a particular zone, you can use the following commands:

```
SET BDY(TGT1) /* Set to requested zone. */.
UNLOAD MOD /* Unload all MOD entries. */.
```

To unload specific MOD entries, you can use these commands:

```
SET BDY(TGT1) /* Set to requested zone. */.
UNLOAD MOD(MOD01 MOD02) /* Unload only these two entries. */.
```

The format of the UNLOAD output for each MOD entry is the same for both of these commands. The only difference is the number of MOD entries unloaded. The following examples show UNLOAD output for MOD entries. Figure 46 does not have cross-zone subentries. Figure 47 on page 287 does.

---

```
UCLIN.
REP MOD ( ASSEM01 )
  LASTUPD ( JCLIN )
  LASTUPDTYPE ( ADD )
  DISTLIB ( SYSPUNCH )
  LMOD ( LMOD99A LMOD99B )

REP MOD ( ASSEM02 )
  LASTUPD ( JCLIN )
  LASTUPDTYPE ( ADD )
  DISTLIB ( SYSPUNCH )
  LMOD ( ASSEM02 )

REP MOD ( DL1MOD1 )
  LASTUPD ( DL1F001 )
  LASTUPDTYPE ( ADD )
  DISTLIB ( DLIB3 )
  FMID ( DL1F001 )
  RMID ( DL1F001 )
  LMOD ( DL1LMOD )

REP MOD ( MOD01 )
  LASTUPD ( JXY1102 )
  LASTUPDTYPE ( ADD )
  DISTLIB ( AOS12 )
  FMID ( JXY1102 )
  RMID ( JXY1102 )
  CSECT ( MOD01C1 MOD01C2 MOD01C3 )
  LMOD ( LMOD01 )

REP MOD ( MOD02 )
  LASTUPD ( JXY1102 )
  LASTUPDTYPE ( ADD )
  DISTLIB ( AOS12 )
  FMID ( JXY1102 )
  RMID ( JXY1102 )
  CSECT ( MOD02C1 )
  LMOD ( LMOD01 LMOD02 )
```

Figure 46. MOD entry: sample UNLOAD output (no cross-zone subentries) (Part 1 of 2)
MOD entry (distribution and target zone)

```plaintext
REP  MOD  ( MOD03  )
    LASTUPD ( JXY1121  )
    LASTUPDTYPE ( UPD  )
    DISTLIB ( AOS12  )
    FMID ( JXY1121  )
    RMID ( JXY1121  )
    UMID ( UZ00010 UZ00014  )
    LMOD ( LMOD03  )

REP  MOD  ( MOD04  )
    LASTUPD ( JXY1121  )
    LASTUPDTYPE ( UPD  )
    DISTLIB ( AOS12  )
    ASSEMBLE
    FMID ( JXY1121  )
    RMID ( JXY1121  )
    RMIDASM
    LMOD ( MOD04  )

REP  MOD  ( MOD05  )
    LASTUPD ( JXY0001  )
    LASTUPDTYPE ( ADD  )
    DISTLIB ( DLIB1  )
    FMID ( JXY0001  )
    RMID ( JXY0001  )

ENDUCL.
```

Figure 46. MOD entry: sample UNLOAD output (no cross-zone subentries) (Part 2 of 2)
By specifying the FORFMID operand, you can reduce the number of MOD entries unloaded. When FORFMID is specified, SMP/E unloads a MOD entry only if its FMID matches one of the FMIDs specified on the FORFMID operand. For example, to unload MOD entries whose FMIDs are either defined in FMIDSET TP or are JXY1102, you can use these commands:

```
SET BDY(TGT1) /* Set to target zone. */.
UNLOAD MOD /* Unload all MOD entries */.
FORFMID(TP /* for the TP FMIDSET */.
        JXY1102) /* and FMID JXY1102. */.
```

**UCLIN Examples**

You can use the ADD, REP, and DEL UCL statements to change subentries in the MOD entry. After the UCLIN changes are done, the MOD entry must contain at least the following subentries:

- **DISTLIB**
MOD entry (distribution and target zone)

- FMID
- RMID

Otherwise, there is not enough information in the entry to process the module. If any of the required subentries are missing, SMP/E does not make the requested UCL updates to the entry, and the entry remains as it was before the UCL command.

The following examples are provided to help you use the MOD entry.

**Example 1: Adding a new MOD entry**

To create a MOD entry, you should use one of the following methods rather than UCLIN:

- JCLIN, if a new module is being added to a new load module. For examples of how JCLIN creates MOD entries, see [SMP/E Commands](#).
- The LMOD operand on the ++MOD statement, if a new module is being added to an existing load module and no changes are needed in the link-edit control statements other than the INCLUDE for the new module. For an example of how to do this, see "++MOD MCS" on page 76.

You could make the same changes with the UCLIN command. However, you should make sure the changes are correct and complete. For example, assume you want to define two new modules, MOD99A and MOD99B. These are linked together to form load module LMOD99AB, which has an entry point of MOD99A and exists in LINKLIB. The FMID of both new modules is ZUSR001, a user-created function. You could use these commands to create the MOD and LMOD entries:

```
SET BDY(TGT1) /* Set to target zone. */.
UCLIN /* */.
ADD MOD(MOD99A) /* Define new module entry. */
    DISTLIB(USRDLIB1) /* Define DLIB. */
    LMOD(LMOD99AB) /* Load module. */
    FMID(ZUSR001) /* Functional owner. */
    /* */.
ADD MOD(MOD99B) /* Define new module entry. */
    DISTLIB(USRDLIB1) /* Define DLIB. */
    LMOD(LMOD99AB) /* Load module. */
    FMID(ZUSR001) /* Functional owner. */
    /* */.
ADD LMOD(LMOD99AB) /* Now define LMOD. */
    SYSLIB(LINKLIB) /* */
    RENT REUS /* Attributes. */
    ++LMODIN /* Link statements. */
    INCLUDE USRDLIB1(MOD99A,MOD99B)
    ENTRY MOD99A
    ++ENDLMODIN /* */
    /* */
    ENDUCL /* */.
```

**Example 2: Forcing assembly of a module**

Assume you have a macro that is used by a source, and you want to make sure the source is always assembled when the macro is changed. You can do this by setting the ASSEMBLE indicator as follows:

```
SET BDY(DLIB1) /* Set to DLIB zone. */.
UCLIN /* */.
ADD MOD(SRC01) /* MOD entry for source. */
    ASSEMBLE /* Always assemble it. */
    /* */.
ENDUCL /* */.
```
Example 3: Completing cross-zone updates

If SMP/E could not complete cross-zone updates for APPLY or RESTORE processing, you may need to use UCLIN to make the remaining changes. (SMP/E issues messages and reports to tell you what cross-zone work you may need to finish.) Here are some examples of using UCLIN to change XZLMOD subentries in MOD entries. Make sure to check the messages and reports to determine whether any additional changes are needed for cross-zone subentries in LMOD entries or TARGETZONE entries.

- **Adding XZLMOD subentries:** This example adds LMOD IEANUC01 to the IGCXXX MOD entry in zone IMS1. If this MOD entry previously had no XZLMOD subentries, SMP/E automatically sets XZLMODP to indicate that the MOD entry now contains XZLMOD subentries.

```plaintext
SET BDY(IMS1) /* set to IMS zone */.
UCLIN /* start UCLIN processing */.
ADD MOD(IGCXXX) /* identify MOD entry */
  XZLMOD((IEANUC01,ZOSZNA)) /* add load module from zone ZOSZNA */.
ENDUCL /* end UCLIN processing */.
```

- **Replacing XZLMOD subentries:** This example replaces all the XZLMOD subentries in MOD entry ISPLINK with two XZLMOD subentries. If this MOD entry previously had no XZLMOD subentries, SMP/E automatically sets XZLMODP to indicate that the MOD entry now contains XZLMOD subentries.

```plaintext
SET BDY(ZOSZNA) /* set to first zone. */.
UCLIN /* start UCLIN processing */.
REP MOD(ISPLINK) /* identify MOD entry */
  XZLMOD((CICSDIAG,CICS1) /* replace list with two values */
    (IMSDIAG1,IMS1)) /* */.
ENDUCL /* end UCLIN processing */.
```

- **Deleting XZLMOD subentries:** This example deletes an XZLMOD subentry from the ISPLINK subentry. If this causes the XZLMOD subentry list to become empty, SMP/E automatically turns XZLMODP off to indicate that the MOD entry no longer contains XZLMOD subentries.

```plaintext
SET BDY(ZOSZNB) /* set to second zone. */.
UCLIN /* start UCLIN processing */.
DEL MOD(ISPLINK) /* delete IMSDIAG2 reference */
  XZLMOD((IMSDIAG2,IMS1)) /* */.
ENDUCL /* end UCLIN processing */.
```
MTSMAC entry (SMPMTS)

The MTSMAC entry is a copy of a macro that resides only in a distribution library but is needed temporarily during APPLY processing. The MTSMAC entry is in the SMPMTS data set, which serves as a target macro library for such macros.

When SMP/E applies the SYSMODs that affect these macros, it calls utility programs to store the macros on the SMPMTS. This way, the most current service level of each macro is available for use in assemblies. After SMP/E has accepted all the SYSMODs that affect these macros, it deletes the associated MTSMAC entries from the SMPMTS.

Note: If you specify SAVEMTS in the OPTIONS entry that is in effect during ACCEPT processing, SMP/E will not delete MTSMAC entries from the SMPMTS after the SYSMODs that affect those macros have been successfully accepted.

Subentries
The MTSMAC entry contains no SMP/E data and appears to the system as a member of a normal macro library.

LIST Examples
You cannot use SMP/E to list the MTSMAC entries. However, you can use standard system utility programs (such as IEBGENER, IEBPTPCH, and IEHLIST) or products such as ISPF to display these entries or information about the data set.

UCLIN Examples
You can use the DEL UCL statement to delete an MTSMAC entry from the SMPMTS. This can be helpful if you plan to do an APPLY followed by ACCEPT when several target libraries have been created from the same distribution library.

When a SYSMOD is accepted into a distribution zone, the entries associated with it are automatically deleted from the SMPMTS for the RELATED target zone. However, even if the SYSMOD was also applied to other target zones created from the same distribution zone, SMP/E does not clean up the SMPMTS data sets for the other target zones.

To delete the entries from these data sets, you can accept the SYSMOD and name these other target zones as the RELATED zone. However, this would update the distribution library each time; this is time-consuming and could use up space in the distribution library data set.

Instead, you can use the DEL command to delete these entries without updating the distribution library. To determine which entries to specify, check the SMPLOG data set to see which ones SMP/E deleted during ACCEPT processing.

Note: You can also use the CLEANUP command to delete MTSMAC entries without specifying them individually. For more information about the CLEANUP command, see SMP/E Commands.

Example: Deleting an MTSMAC entry
Assume that you have two target zones, TGT1 and TGT2, generated off the same distribution zone, DLB1. During ACCEPT processing of a SYSMOD, SMP/E has deleted MTSMAC MAC01 and MAC02 from the SMPMTS data set associated with target zone TGT2. After performing the ACCEPT, you want to delete the same
macro from the SMPMTS associated with target zone TGT1. Assume either that
you have a cataloged procedure for TGT1 with the correct SMPMTS specified, or
that you have set up the correct DDDEF entries. The following UCLIN can be used
to delete the MTSMAC entry:

```
SET BDY(TGT1) /* Set to TGT zone. */.
UCLIN /* */.
DEL MTSMAC(MAC01) /* Delete the macro. */.
DEL MTSMAC(MAC02) /* Delete the macro. */.
ENDUCL /* */.
```

**Note:** One UCL statement is required for each MTSMAC entry to be deleted.

You can make the same changes by use of system utilities. However, the SMPLOG
will not reflect the processing done.
OPTIONS entry (global zone)

The OPTIONS entry defines processing options that are to be used for an SMP/E command or set of commands. Although OPTIONS entries exist only in the global zone, they are also used to process commands for target and distribution zones. There are two ways you can specify the OPTIONS entry that should be in effect when you are processing a zone:

- In the GLOBALZONE, TARGETZONE, and DLIBZONE entries. The OPTIONS entry specified here is the default OPTIONS entry for that zone.
- On the SET command. The name specified on the SET command overrides the default OPTIONS name.

When SMP/E processes a command, it checks these sources to determine which OPTIONS entry should be in effect for that command. If SMP/E cannot find a reference to an OPTIONS entry, it will use defaults for some of the subentries. The following section describes these defaults, as well as the other values you can specify for each subentry.

Subentries

These are the subentries for the OPTIONS entry as they appear in the LIST output:

- **name**
  - is the name of the OPTIONS entry.
  - The name can contain from 1 to 8 alphanumeric characters.

- **AMS**
  - is the name of the UTILITY entry that SMP/E is to use to obtain information when calling the access method services (AMS) utility.
  - The UCL operand is `AMS(name)`.
  - The name can contain from 1 to 8 alphanumeric characters.
  - If no entry name is specified, SMP/E uses a default UTILITY entry. For details, see Table 6 on page 350.

- **ASM**
  - is the name of the UTILITY entry that SMP/E is to use to obtain information when calling the assembler utility.
  - The UCL operand is `ASM(name)`.
  - The name can contain from 1 to 8 alphanumeric characters.
  - If no entry name is specified, SMP/E uses a default UTILITY entry. For details, see Table 6 on page 350.

- **CHANGEFILE**
  - specifies whether library change file records should be generated and written to the SMPDATA1 and SMPDATA2 data sets during APPLY and RESTORE command processing.
  - The UCL operand is `CHANGEFILE(YES | NO)`, where:
    - **YES** indicates that library change file records should be generated during APPLY or RESTORE processing.
    - **NO** indicates that library change file records should not be generated during APPLY or RESTORE processing. This is the default for APPLY and RESTORE processing.
OPTIONS entry (global zone)

**COMP**

is the name of the UTILITY entry that SMP/E is to use to obtain information when calling the utility program to compress data sets.

The UCL operand is **COMP(name)**.

- The name can contain from 1 to 8 alphanumeric characters.
- If no entry name is specified, SMP/E uses a default UTILITY entry. For details, see [Table 6 on page 350](#).

**COMPACT**

specifies whether inline element data within SYMMS in the SMPPTS data set should be compacted. The element data is normally compacted during the GZONEMERGE and RECEIVE commands.

The UCL operand is **COMPACT(YES|NO)**, where:

**YES**

indicates inline element data within SYMMS in the SMPPTS should be compacted to reduce the space requirements of the SMPPTS during the GZONEMERGE and RECEIVE commands. The element data is expanded as needed during ACCEPT and APPLY command processing. YES is the default.

**NO**

indicates inline element data within SYMMS in the SMPPTS should not be compacted during the GZONEMERGE and RECEIVE commands. The element data will reside in the SMPPTS data set in its original form.

**COPY**

is the name of the UTILITY entry that SMP/E is to use to obtain information when calling the copy utility.

The UCL operand is **COPY(name)**.

- The name can contain from 1 to 8 alphanumeric characters.
- If no entry name is specified, SMP/E uses a default UTILITY entry. For details, see [Table 6 on page 350](#).

**DSPREFIX**

specifies the data set prefix to be used to construct the full data set name when SMPTLIB data sets are being allocated for RELFILEs. For more information about names for SMPTLIB data sets, see [SMP/E Commands](#).

The UCL operand is **DSPREFIX(prefix)**.

- The prefix can contain from 1 to 26 alphanumeric characters.
- The prefix must follow standard naming conventions for data sets.
- If a DDDEF entry is in effect for the SMPTLIB data sets, the DSPREFIX value in that DDDEF entry overrides the DSPREFIX value specified in the OPTIONS entry in effect.
- If no DSPREFIX value is specified in either the OPTIONS entry or the DDDEF entry, no high-level qualifier is assigned to the SMPTLIB data sets.
- If the DSPREFIX value is the same as the RFDSPREFIX value for the RELFILE data sets that are being processed, the SMPTLIB data sets cannot be allocated in either of these cases:
  - The RELFILE data sets are on DASD.
  - The RELFILE data sets are on tape and are cataloged.

**DSSPACE**

specifies the primary and secondary space allocation (in tracks) and the number of directory blocks to be allocated for each SMPTLIB data set. After
OPTIONS entry (global zone)

the data set is loaded, unused space is freed. For more information about SMPTLIB data sets, see [SMP/E Commands](#). The UCL operand is **DSSPACE**(prime, secondary, directory).

- Each value can contain from 1 to 4 numeric characters.
- If a DDDEF entry is in effect for the SMPTLIB data sets, the SPACE and DIR values in that DDDEF entry override the DSSPACE values specified in the OPTIONS entry in effect.
- These values should be specified in the appropriate OPTIONS or DDDEF entries before you receive a relative file. Otherwise, SMP/E cannot allocate any new SMPTLIB data sets.

**EXRTYDD**

specifies the list of ddnames that are not eligible for retry processing after an x37 abend. EXRTYDD is used with RETRYDDN in order to exclude a subset of libraries from retry processing.

The UCL operand is **EXRTYDD**(ddname...).

- The ddnames can contain from 1 to 8 alphanumeric characters.
- If a ddname is specified in both the EXRTYDD list and the RETRYDDN list, the ddname is excluded from retry processing.
- If no ddnames are specified for RETRYDDN, no retry processing will be done for any libraries. The ddnames specified on EXRTYDD are ignored.
- If ALL is specified on EXRTYDD, it is treated as just another ddname; it does not exclude all ddnames from retry processing. To exclude all libraries from retry processing, do one of the following instead:
  - Specify **RETRY(NO)** on the SMP/E command being processed.
  - Do not specify a RETRYDDN list in the OPTIONS entry that is in effect for the SMP/E command being processed.
  - Do not have an OPTIONS entry in effect for the SMP/E command being processed.

**FIXCAT**

specifies the list of fix categories. During APPLY, ACCEPT, and REPORT MISSINGFIX processing, this subentry, or the FIXCAT operand on the command, identifies the fix categories of interest and is used to determine which FIXCAT HOLDDATA entries will affect command processing.

The UCL operand is **FIXCAT**(category...).

- A Fix Category value is case sensitive, can be 1- to 64-characters in length, can contain any nonblank character in the range 'X'41' - 'X'FE' except single quotation mark (‘), comma (,), left parenthesis (()), and right parenthesis ()), and can be specified in two ways:
  - Explicitly, by fully specifying a particular fix category value. For example, IBM.Device.ZIIP. In this case, all HOLDDATA associated with this Fix Category is applicable to command processing.
  - Implicitly, by partially specifying a fix category value using any number of asterisks (*) as global characters and percent signs (%) as placeholders.
  - A single asterisk indicates that zero or more characters can occupy that position. Here are some examples:

**IBM.Device***

In this example, all HOLDDATA associated with a fix category that begins with the character string IBM.Device is applicable.
OPTIONS entry (global zone)

*z/OS In this example, all HOLDDATA associated with a fix category that ends with the character string z/OS is applicable.

IBM+z/OS
In this example, all HOLDDATA associated with a fix category that begins with the character string IBM and ends with the character string z/OS is applicable.

A single percent sign indicates that any one single character can occupy that position. For example, IBM.Device.20%4 indicates that HOLDDATA associated with any of the following fix categories is applicable: IBM.Device.2084, IBM.Device.2094, and IBM.Device.20t4. HOLDDATA associated with a fix category of IBM.Device.20914, however, is not applicable.

The following examples are all acceptable fix categories:
- IBM.Device.zIIP
- IBM.Function*
- IBM.Device.20%4.*
- *.HealthChecker

HFSCOPY
is the name of the UTILITY entry that SMP/E is to use to obtain information when calling the HFS copy utility.

This is the utility used to install hierarchical file system elements and must meet the same program-to-program interface as BPXCOPY.

The UCL operand is HFSCOPY(name).
- The name can contain from 1 to 8 uppercase alphanumeric characters.
- If no HFSCOPY subentry is specified in the OPTIONS entry, SMP/E uses a default UTILITY entry. For details, see Table 6 on page 350.

IOSUP
is the name of the UTILITY entry that SMP/E is to use to obtain information when calling the IEHIOSUP utility program to process maintenance for an OS/VS1 system.

The UCL operand is IOSUP(name).
- The name can contain from 1 to 8 alphanumeric characters.

LKED
is the name of the UTILITY entry that SMP/E is to use to obtain information when calling the link-edit utility.

The UCL operand is LKED(name).
- The name can contain from 1 to 8 alphanumeric characters.
- If no entry name is specified, SMP/E uses a default UTILITY entry. For details, see Table 6 on page 350.

MSGWIDTH
specifies whether the message output (SMPOUT only) should be formatted in 80 or 120 character widths.

The UCL operand is MSGWIDTH(value). The subentry can contain one of the following values:
- 80 indicates that the message text should be formatted to an 80 character width.
OPTIONS entry (global zone)

120 indicates that the message text should be formatted to a 120 character width. This is the default.

MSGFILTER specifies whether messages issued to SMPOUT during APPLY, ACCEPT, and RESTORE processing should be filtered to contain only the following:
- Messages with a severity of Warning or higher.
- Messages with a severity of Informational that add clarity or additional information to a previously issued higher severity message.
- Messages GIM20501I and GIM20502I.

The UCL operand is MSGFILTER(YES | NO), where:
- YES indicates that the number of messages issued to SMPOUT should be in a filtered format for APPLY, ACCEPT, and RESTORE processing.
- NO indicates that the number of messages issued to SMPOUT should be in an unfiltered format for APPLY, ACCEPT, and RESTORE processing. This is the default.

NOPURGE indicates that after SMP/E accepts SYSMODs, it should not delete the associated global zone SYSMOD entries, SMPPTS MCS entries, or SMPTLIB data sets.

The UCL operand is NOPURGE.
- The default is for NOPURGE not to be specified. In this case, SMP/E deletes the entries after the SYSMOD has been successfully accepted. (This is true only if the SYSMOD has been applied and BYPASS(APPLYCHECK) was not specified on the ACCEPT command.) The associated HOLDDATA, including internal SYSTEM HOLDDATA, is not deleted when the SYSMOD is deleted.
- Although this operand can be specified in any OPTIONS entry, it is effective only when the OPTIONS entry is used during ACCEPT processing.

NOREJECT specifies that the global zone SYSMOD entry and the associated MCS entry should not be deleted after the SYSMOD is restored.

The UCL operand is NOREJECT.
- The default is for NOREJECT not to be specified. In this case, SMP/E deletes the entries after the SYSMOD has been successfully restored. The associated HOLDDATA, including internal SYSTEM HOLDDATA, is not deleted when the SYSMOD is deleted.
- Although this operand can be specified in any OPTIONS entry, it is effective only when the OPTIONS entry is used during RESTORE processing.

ORDERRET
Indicates the retention period, in days, that ORDER entries are kept in the global zone before being deleted. During RECEIVE ORDER processing, an ORDER entry will be deleted from the global zone if either of the following conditions occurs:
- The ORDER entry has a status of DOWNLOADED, and the difference between the current date and the ORDER entry's DOWNLDATE subentry value is greater than the OPTIONS entry ORDER RETENTION subentry value.
The ORDER entry has a status of ERROR or PENDING and the difference between the current date and the ORDER entry’s ORDERDATE subentry value is greater than the OPTIONS entry ORDER RETENTION subentry value.

When an ORDER entry is deleted from the global zone, SMP/E also deletes the order package stored in the SMPNTS.

The UCL operand is ORDERRET(nnmn).
- The value can contain from 1 to 4 numeric characters from 0 to 9999.
- If no value is specified, the default is 180 days.

**PAGELEN**

specifies the page length for the SMPLIST, SMPHRPT, SMPOUT, and SMPRPT data sets.

The UCL operand is PAGELEN(nnmn).
- The value can contain from 1 to 4 numeric characters.
- If no value is specified, the default is 60.

**PEMAX**

specifies the maximum number of subentries that can be present in any CSI entry. Most often, the largest entry is a SYSMOD entry.

The UCL operand is PEMAX(nnmn).
- The value can contain from 1 to 4 numeric characters, with a value range of 1 to 9999.
- If no value is specified, the default is 32767. Therefore, if you want a value higher than 9999, you must use the default.

**RECEXZGRP**

specifies a list of zones or zonesets to be excluded during APPLYCHECK and ACCEPTCHECK processing during the SYSMOD selection phase of RECEIVE processing.

The UCL operand is RECEXZGRP(value...).
- Each value is a zone or zoneset name. Each name can contain from 1 to 8 alphanumeric characters.
- The zone or zonesets can contain both target and distribution zones.

**Note:** The value specified cannot be GLOBAL nor ALLZONES.
- If a zone is specified in both the RECEXZGRP list and the RECZGRP list, the zone is excluded from processing.
- If no zones are specified for RECZGRP, the zones specified on RECEXZGRP are ignored.

**RECZGRP**

specifies a list of zones and zonesets eligible for APPLYCHECK and ACCEPTCHECK processing during the SYSMOD selection phase of RECEIVE processing. Any SYSMOD that has been applied or accepted into any of the zones specified by this list will not be received by the RECEIVE command, unless the SYSMOD is specified on the SELECT operand of the RECEIVE command or the list is overridden by the BYPASS(APPLYCHECK) or BYPASS(ACCEPTCHECK) operands, or a new list is specified by the ZONEGROUP operand.

The UCL operand is RECZGRP(value...).
- Each value is a zone or zoneset name. Each name can contain from 1 to 8 alphanumeric characters.
OPTIONS entry (global zone)

- The zone or zonesets can contain both target and distribution zones.

  **Note:** The value specified cannot be GLOBAL.

- ALLZONES can be specified to indicate that all zones defined by a ZONEINDEX subentry in the GLOBALZONE entry are eligible. When ALLZONES is specified, all other values specified are ignored.

- If no zones or zonesets are specified, no APPLYCHECK or ACCEPTCHECK processing is performed unless the ZONEGROUP operand is specified on the RECEIVE command.

**RETRY**

is the name of the UTILITY entry that SMP/E is to use to obtain information when calling the utility program to compress a data set after an x37 abend.

The UCL operand is **RETRY(name)**.

- The name can contain from 1 to 8 alphanumeric characters.

- If no entry name is specified, SMP/E uses a default UTILITY entry. For details, see [Table 6 on page 350](#).

**RETRYDDN**

specifies the list of ddnames eligible for retry processing after an x37 abend.

The UCL operand is **RETRYDDN(ddname...)**.

- The ddnames can contain from 1 to 8 alphanumeric characters.

- **ALL** can be specified to indicate that all ddnames are eligible.

- If no ddnames are specified, no retry processing will be done, even if RETRY was specified on the command being processed.

- Retry processing is attempted for partitioned data sets (PDSs) and partitioned data sets extended (PDSEs) that experience an x37 abend, indicating that they have run out of space.

- For pointers on how to set up the desired retry processing, see [SMP/E User’s Guide](#).

**SAVEMTS**

indicates that MTSMAC entries should not be deleted from the SMPMTS after the SYSMODs that affect those macros have been successfully accepted.

The UCL operand is **SAVEMTS**.

- If SAVEMTS is not specified, SMP/E deletes the entries after all the SYSMODs that affect the macros have been successfully accepted. (This is true only if the SYSMOD has been applied and **BYPASS(APPLYCHECK)** was not specified on the ACCEPT command.)

- Although this operand can be specified in any OPTIONS entry, it is effective only when the OPTIONS entry is used during ACCEPT processing.

**SAVESTS**

indicates that STSSRC entries should not be deleted from the SMPSTS after the SYSMODs that affect the source have been successfully accepted.

The UCL operand is **SAVESTS**.

- If SAVESTS is not specified, SMP/E deletes the entries after all the SYSMODs that affect the source have been successfully accepted. (This is true only if the SYSMOD has been applied and **BYPASS(APPLYCHECK)** was not specified on the ACCEPT command.)

- Although this operand can be specified in any OPTIONS entry, it is effective only when the OPTIONS entry is used during ACCEPT processing.
**OPTIONS entry (global zone)**

**SUPPHOLD**

specifies a list of HOLD reason IDs for which the HOLDDATA card image should not be displayed on the following reports:

- Unresolved HOLD Reason Report
- Bypassed HOLD Reason Report
- SYSMOD Comparison HOLDDATA Report

The UCL operand is `SUPPHOLD(value...)`, where:

- `value` is a HOLD reason ID, which is from 1 to 7 alphanumeric characters, #, @, and $. The first character cannot start with 0-9.

**UPDATE**

is the name of the UTILITY entry that SMP/E is to use to obtain information when calling the update utility.

The UCL operand is `UPDATE(name)`.

- The name can contain from 1 to 8 alphanumeric characters.
- If no entry name is specified, SMP/E uses a default UTILITY entry. For details, see [Table 6 on page 350](#).

**ZAP**

specifies the name of the UTILITY entry that SMP/E is to use to obtain information when calling the superzap utility.

The UCL operand is `ZAP(name)`.

- The name can contain from 1 to 8 alphanumeric characters.
- If no entry name is specified, SMP/E uses a default UTILITY entry with NAME(IMASPZAP), RC(4), PRINT(SYSPRINT), and PARM().

**LIST Examples**

To list all the OPTIONS entries in a global zone, you can use the following commands:

```
SET BDY(GLOBAL) /* Set to requested zone. */.
LIST OPTIONS /* List all OPTIONS entries. */.
```

To list specific OPTIONS entries in a global zone, you can use these commands:

```
SET BDY(GLOBAL) /* Set to requested zone. */.
LIST OPTIONS(OPT1, /* List only these three */
  TGT1, /* entries. */
  DLIB1) /* */.
```

The format of the LIST output for each OPTIONS entry is the same for both of these commands. The only difference is the number of OPTIONS entries listed. **Figure 48 on page 300** is an example of LIST output for OPTIONS entries.
UCLIN Examples

You can use the ADD, REP, and DEL UCL statements to change subentries in an OPTIONS entry. When you use UCLIN to update an OPTIONS entry, remember that if a DEL statement deletes all the existing subentries in the entry, SMP/E deletes the entire entry.

The following examples are provided to help you use the OPTIONS entry.

Example 1: Connecting an OPTIONS entry to UTILITY entries

Assume that you have set up three UTILITY entries: IEUASM, IEWL, and MYX37. For examples of setting up these entries, see "UTILITY entry (global zone)" on page 350. Now you wish to define an OPTIONS entry, OPT1, that will be used during APPLY processing. The following UCL can be used to set up that entry:

```
SET BDY(GLOBAL) /* Set to global zone. */.
UCLIN /* */.
ADD OPTIONS(OPT1) /* New OPTIONS entry. */
   ASM(IEUASM) /* Connect to assembler data. */
   LKED(IEWL) /* Connect to link data. */
   RETRY(MYX37) /* Connect to retry. */
RETRY(OPT1) /* */.
ENDUCL /* */.
```
To use the OPTIONS entry, you need either to define it as the default OPTIONS entry for the desired GLOBALZONE, DLIBZONE, or TARGETZONE entry, or to specify it on the OPTIONS operand of the SET command for the zone you want to process with these values.

**Example 2: Changing the SMPOUT page length**
Assume that you wish to modify the OPTIONS entry created in "Example 1: Connecting an OPTIONS entry to UTILITY entries" on page 300 to indicate that your printer now prints 120 lines per page. The following UCL will do this:

```
SET BDY(GLOBAL)  /* Set to global zone. */.
UCLIN /* */.
ADD OPTIONS(OPT1) /* Add because page length is not there yet. */.
    PAGELEN(120) /* Change page length. */.
ENDUCL /* */.
```

**Example 3: Preparing to receive RELFILEs**
To receive a new function packaged in RELFILE format, you have to define the amount of space to allocate the SMPTLIB data sets and, optionally, the prefix to be used in building the data set names. Assume that the program directory indicates that 300 tracks of primary space should be used, 50 tracks of secondary space, and 25 directory blocks, and that you want the data set names to start with SMP.RELFILE.M9801. The following UCL can be used to add this information to the OPTIONS entry created in "Example 1: Connecting an OPTIONS entry to UTILITY entries" on page 300:

```
SET BDY(GLOBAL)  /* Set to global zone. */.
UCLIN /* */.
ADD OPTIONS(OPT1) /* Add because data is not there yet. */.
    DSSPACE(300,50,25) /* Space allocation. */.
    DSPREFIX(SMP.RELFILE.M9801) /* Prefix. */.
ENDUCL /* */.
```

**Example 4: Identifying libraries for retry processing**
During processing of commands that update product libraries (such as ACCEPT, APPLY, RESTORE, and LINK), utilities called by SMP/E may issue x37 abends when the libraries they are updating run out of space. SMP/E can attempt to recover from such out-of-space errors if RETRY(YES) was specified on the command being processed and if a RETRYDDN list is available in the OPTIONS entry that is in effect.

Assume that you want SMP/E to attempt this retry processing for all libraries except LINKLIB, MIGLIB, and NUCLEUS. The following UCL can be used to specify the desired ddnames in the OPTIONS entry created in "Example 1: Connecting an OPTIONS entry to UTILITY entries" on page 300:

```
SET BDY(GLOBAL)  /* Set to global zone. */.
UCLIN /* */.
ADD OPTIONS(OPT1) /* Add because data is not there yet. */.
    RETRYDDN(ALL) /* Retry all ddnames */.
    EXRTYDD(LINKLIB,MIGLIB,NUCLEUS) /* except these. */.
ENDUCL /* */.
```
ORDER entry (global zone)

ORDER entry (global zone)

The ORDER entry describes a HOLDDATA or PTF order initiated with the RECEIVE ORDER command. When SMP/E sends an order request to the IBM Automated Delivery Request server and the server accepts the order, SMP/E creates an ORDER entry in the global zone. The ORDER entry is used to record information about the order so that SMP/E can query the server for status of orders that have not been completed. Once the processing for an order has been completed by the server, SMP/E can download the HOLDDATA or PTF package for the order and store the package files in the SMPNTS directory.

Subentries

These are the subentries for the ORDER entry as they appear in the LIST output:

name

is the name of the ORDER entry.

The name can contain from 1 to 8 alphanumeric characters.

CONTENT

the HOLDDATA or PTF content for this order. The value can be one of the following:

ALL

indicates a request for all available PTFs.

APARS

indicates a request for PTFs that resolve the APARS specified in the APARS subentry.

CRITICAL

indicates a request for all available PTFs that resolve a critical problem. A critical problem is a high impact pervasive (HIPER) or a PTF in error (PE).

HOLDDATA

indicates a request for HOLDDATA only. A HOLDDATA package contains the last 2-years worth of Enhanced HOLDDATA for the entire z/OS software platform. See http://service.software.ibm.com/holdata/390holddata.html for further information about Enhanced HOLDDATA.

PTFS

indicates a request for the PTFs indicated in the PTFS subentry.

sourceid

indicates a request for all the recommended PTFs identified with the indicated Recommended Service Update SOURCEID (RSUyymm), and PTFs that resolve a critical problem (HIPER or PE). Recommended service includes PTFs through the indicated RSU level, which is the most current RSU level at the time the order is created.

STATUS

indicates the status of the HOLDDATA or PTF order. Status values can be any one of the following:

PENDING

the order request has been submitted to the server but the order's package has not yet been downloaded.

DOWNLOADED

the order package has been downloaded and stored in the SMPNTS directory.
ORDER entry (global zone)

ERROR
the server reported an uncorrectable error with the processing of this order.

APARS
identifies the APARS for which resolving PTFs were requested.

PTFS
identifies the specific PTFs that were requested.

ORDERDATE
indicates the date on which the order request was sent to the server.

ORDERTIME
indicates the time at which the order request was sent to the server.

DOWNLDATE
indicates the date on which the order's package was downloaded and stored in the SMPNTS directory.

DOWNLTIME
indicates the time at which the order's package was downloaded and stored in the SMPNTS directory.

ORDERID
the order identifier assigned by the IBM Automated Delivery Request server when the order request was sent. The value is used to correlate ORDER entries in the global zone with orders being processed by the server.

USERID
the userid that submitted the SMP/E job which created the ORDER entry.

PKGID
indicates the subdirectory name within the SMPNTS directory that contains the package associated with the order.

ZONES
indicates the target zones used to represent the software inventory associated with this order.

ORDERSERVER
contains the <ORDERSERVER> tags used to identify the IBM Automated Delivery Request server scheduled to fulfill the order request. This information is obtained from the ORDERSERVER data set specified in the RECEIVE ORDER command when the ORDER entry was created.

This subentry exists only when the order has a STATUS of PENDING. This subentry is deleted once the order package has been downloaded.

LIST Examples
To list all the ORDER entries in a global zone, you can use the following commands:

```
SET BDY(GLOBAL) /* Set to requested zone. */.
LIST ORDER /* List all ORDER entries. */.
```

To list specific ORDER entries in a global zone, you can use these commands:

```
SET BDY(GLOBAL) /* Set to requested zone. */.
LIST ORDER(ORD00034, ORD00035, ORD00036) /* List only these three entries. */.
```

The format of the LIST output for each ORDER entry is the same for both of these commands. The only difference is the number of ORDER entries listed.
ORDER entry (global zone)

Figure 49 is an example of LIST output for ORDER entries. In this example, processing for ORDER entries ORD00034 and ORD00035 is completed and the packages have been downloaded. The order described by ORDER entry ORD00036 is in the PENDING state and the package has not yet been downloaded.

Note: An alternative url for the IBM Automated Delivery Request server is https://eccgw02.rochester.ibm.com/services/projects/ecc/ws/.

UCLIN Examples

You can use the DEL UCL statement to delete an ORDER entry as shown in the following example. UCLIN does not support either adding entries or updating subentries for an ORDER entry.

Example: Deleting an ORDER entry

The following UCL can be used to delete an ORDER entry:

```ucl
SET BDY(GLOBAL) /* Set to global zone */.
UCLIN /* Start UCLIN processing */.
DEL ORDER(ORD00035) /* Delete ORDER entry */.
ENDUCL /* End UCLIN processing */.
```
PRODUCT entry (global zone)

ORDER entries may reside only in the GLOBALZONE.
PRODUCT entry (global zone)

The PRODUCT entry describes a software product. A product is known to SMP/E by the combination of its \textit{prodid} and the \textit{vv.rr.mm} values. PRODUCT entries reside only in the global zone.

**Subentries**

\textit{prodid}

specifies the product identifier for the product described in the PRODUCT entry. For IBM products, it is recommended that the \textit{prodid} be the IBM program product number (for example, “5647-A01”).

\textit{vv.rr.mm}

specifies the version, release and modification level for the product described in the PRODUCT entry.

**DESCRIPTION**

describes the product.

The UCL operand is \texttt{DESCRIPTION(description)}.

**SREL**

specifies the system or subsystem releases on which the PRODUCT can be installed. There can be multiple SREL subentries associated with a PRODUCT.

The UCL operand is \texttt{SREL(srel,...)}.

**PRODSUP**

specifies the PRODUCTS that are superseded by this PRODUCT. Multiple PRODSUP subentries can be associated with a PRODUCT. The combination of \textit{prodid} and \textit{vv.rr.mm} determines the uniqueness of a subentry in the PRODSUP subentry list. Only one subentry value for a given \textit{prodid} and \textit{vv.rr.mm} combination is saved in the PRODSUP subentry list.

The UCL operand is \texttt{PRODSUP((prodid,vv.rr.mm),...)}.

**URL**

specifies the URL that can be accessed to obtain additional information about the product.

The UCL operand is \texttt{URL(product_url)}.

**VENDOR**

specifies the name of the vendor supplying the product.

The UCL operand is \texttt{VENDOR(vendor_name)}.

**RECDATE**

specifies the date on which the PRODUCT was received.

There is no UCL support for this subentry.

**RECTIME**

specifies the time at which the PRODUCT was received.

There is no UCL support for this subentry.

**UCLDATE**

specifies the date on which the PRODUCT was last processed using the UCLIN command.

There is no UCL support for this subentry.
PRODUCT entry (global zone)

UCLTIME
specifies the time at which the PRODUCT was last processed using the UCLIN command.

There is no UCL support for this subentry.

REWORK
identifies the level of this PRODUCT, which was received again for minor changes.

There is no UCL support for this subentry.

LIST Examples
To list all the PRODUCT entries in a particular zone, you can use the following commands:

```
SET BDY(GLOBAL) /* Set to global zone. */.
LIST PRODUCT /* List all PRODUCT entries.*/.
```

To list specific PRODUCT entries, you can use these commands:

```
SET BDY(GLOBAL) /* Set to global zone. */.
LIST PRODUCT(5647-A01) /* List all PRODUCT entries */
/* with prodid of 5647-A01 */.
LIST PRODUCT((5645-001,01.03.00))
/* List PRODUCT entry with */
/* prodid of 5645-001 */
/* and VRM of 01.03.00 */.
LIST PRODUCT(5668-949,(5645-001,01.2.0))
/* List all PRODUCT entries */
/* with prodid of 5668-949, */
/* plus PRODUCT entry with */
/* prodid of 5645-001 */
/* and VRM of 01.02.00 */.
```

The format of the LIST output for each PRODUCT entry is the same for all of these commands. The only difference is the number of PRODUCT entries listed. Figure 50 shows an example of LIST output for PRODUCT entries. PRODUCT entries are listed alphanumerically by prodid. PRODUCT entries with the same prodid value are further sorted by vv.rr.mm.

![Figure 50. PRODUCT entry: sample LIST output](image)

Notes:
1. The entry name value for the PRODUCT entries formatted to the left of the subentry names is the prodid value.
2. If the URL value extends past column 120, it is continued on the next line.
3. The display for each PRODSUP subentry occupies two columns. The first column is the prodid and the second column is the vv.rr.mm. Four subentries can fit on a row.

UCLIN Examples
You can use the ADD, REP, and DEL UCL statements to change subentries in the PRODUCT entry. When you use UCLIN to update a PRODUCT entry, keep in mind that after the UCLIN changes are made, the PRODUCT entry must contain at least the DESCRIPTION and SREL subentries.

Example: Adding a PRODUCT entry
Assume you have a product, SAMPLE V1R2, for which you wish to create a description and put in an entry named “1234-567,1.2.0”. You could set up a PRODUCT entry as follows:

```
SET BDY(GLOBAL) /* Set to global zone */.
UCLIN /* Start UCLIN processing */.
ADD PRODUCT(1234-567,1.2.0) /* Identify the product */.
  DESCRIPTION(SAMPLE) /* - Name */.
  VENDOR(IBM) /* - Vendor */.
  PRODSUP((1234-456,01.04.00)) /* - Product sups */.
  SREL(Z038) /* - SREL */.
  URL(http://www.ibm.com/) /* URL */.
ENDUCL /* End UCLIN processing */.
```
PROGRAM entry (distribution and target zone)

The PROGRAM entry describes a program element (a pre-built load module or a program object). Program elements may exist in distribution or target libraries. A PROGRAM entry is created the first time you install a SYSMOD that contains a ++PROGRAM MCS for a program element that does not yet have a PROGRAM entry.

SMP/E records the function and service level of the program element in the entry. Once a PROGRAM entry exists, it is updated as subsequent SYSMODs that affect the program element are installed.

Subentries

These are the subentries for the PROGRAM entry as they appear in the LIST output:

- **name** is the name of the program element represented by the entry. It can contain from 1 to 8 alphanumeric characters and $, #, @, or hex C0.

- **ALIAS** specifies a list of alias names for the element.
  The UCL operand is `ALIAS(name...)`.
  Each alias name can contain from 1 to 8 alphanumeric characters.

- **DISTLIB** specifies the ddname of the distribution library for the program element.
  The UCL operand is `DISTLIB(ddname)`.
  - The ddname can contain from 1 to 8 alphanumeric characters.
  - The DISTLIB subentry is required. Without it, SMP/E cannot process any changes for the program element.

- **FMID** specifies the functional owner of this program element. The functional owner is the last function SYSMOD that replaced this element.
  The UCL operand is `FMID(sysmod_id)`.
  - The SYSMOD ID must contain 7 alphanumeric characters.

- **LASTUPD** identifies the cause of the last change to this PROGRAM entry.
  The UCL operand is `LASTUPD(value)`. This subentry can contain one of the following values:
    - **UCLIN** indicates that the change was made as a result of UCLIN processing.
    - **sysmod_id** indicates that the change was made during the installation of the indicated SYSMOD.
      - The SYSMOD ID must contain 7 alphanumeric characters.

- **LASTUPD TYPE** indicates how the entry was last changed.
  The UCL operand is `LASTUPDTYPE(value)`. This subentry may contain one of the following values:
**PROGRAM entry (distribution and target zone)**

- **ADD** The entry was added.
- **UPD** The entry was updated.

**RMID**

Identifies the last SYSMOD that replaced this program element. Any subsequent SYSMOD that modifies this program element must have a defined relationship (such as PRE or SUP) with this SYSMOD.

The UCL operand is RMID(sysmod_id).

- The SYSMOD ID must contain 7 alphanumeric characters.
- If RMID is not specified but FMID is, SMP/E sets the RMID value to the specified FMID.

**SYSLIB**

Specifies the ddname of the target library for the program element.

The UCL operand is SYSLIB(ddname).

- You can specify only one SYSLIB value.
- The ddname can contain from 1 to 8 alphanumeric characters.

**LIST Examples**

To list all the PROGRAM entries in a particular zone, you can use the following commands:

```plaintext
SET BDY(TGT1) /* Set to requested zone. */.
LIST PROGRAM /* List all PROGRAM entries. */.
```

To list specific PROGRAM entries, you can use these commands:

```plaintext
SET BDY(TGT1) /* Set to requested zone. */.
LIST PROGRAM(PGM1 /* List only these two */
PGM2) /* entries. */.
```

The format of the LIST output for each PROGRAM entry is the same for both of these commands. The only difference is the number of PROGRAM entries listed. [Figure 51](#) is an example of LIST output for PROGRAM entries.

---

**Figure 51. PROGRAM entry: sample LIST output**

By specifying the FORFMID operand, you can reduce the number of PROGRAM entries listed. When FORFMID is specified, SMP/E lists a PROGRAM entry only if its FMID matches one of the FMIDs specified on the FORFMID operand. For example, to list PROGRAM entries whose FMIDs are defined in FMIDSET ABC or else are MYPROG1, you can use these commands:
PROGRAM entry (distribution and target zone)

```plaintext
SET BDY(TGT1) /* Set to target zone. */.
LIST PROGRAM /* List all PROGRAM entries */
  FORFMID(ABC /* for the ABC FMIDSET */
  MYPROG1) /* and FMID MYPROG1. */
```

You can use the LIST command to find out the names of all SYSMODs that have modified program elements. To include the names of these SYSMODs in the LIST output, you can use the XREF operand, as shown in these commands:

```plaintext
SET BDY(TGT1) /* Set to requested zone. */.
LIST PROGRAM /* List all PROGRAM entries */
  XREF /* and related SYSMODs. */
```

Notes:

1. You can use XREF in either mass mode or select mode.
2. SMP/E obtains the data included for the XREF operand by checking for entries for this program element in all the SYSMOD entries. Because this data is not contained in the PROGRAM entry itself, you cannot use UCLIN to change it in the PROGRAM entry.

Figure 52 is an example of the LIST output produced when the XREF operand is used.

```
<table>
<thead>
<tr>
<th>NAME</th>
<th>LASTUPD</th>
<th>LIBRARIES</th>
<th>FMID</th>
<th>RMID</th>
<th>SYSMOD HISTORY</th>
</tr>
</thead>
<tbody>
<tr>
<td>PGM1</td>
<td>MYPROG1</td>
<td>DISTLIB=ADSTLIB</td>
<td>MYPROG1</td>
<td>MYPROG1</td>
<td>MYPROG1 FUNCTION 07.100 PROGRAM APP ACC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SYSLIB=SPGMLIB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PGM2</td>
<td>PGMPFT1</td>
<td>DISTLIB=ADSTLIB</td>
<td>MYPROG1</td>
<td>PGMPFT1</td>
<td>PGMPFT1 PTF 07.150 PROGRAM APP ACC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SYSLIB=SPGMLIB</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Figure 52. PROGRAM entry: sample LIST output when XREF is specified

UNLOAD Examples

To dump the PROGRAM entries in UCL format, you can use the UNLOAD command. To unload all the PROGRAM entries in a particular zone, you can use the following commands:

```plaintext
SET BDY(TGT1) /* Set to requested zone. */.
UNLOAD PROGRAM /* Unload all PROGRAM entries. */
```

To unload specific PROGRAM entries, you can use these commands:

```plaintext
SET BDY(TGT1) /* Set to requested zone. */.
UNLOAD PROGRAM(PGM1) /* Unload only these two */
  PGM2) /* entries. */
```

The format of the UNLOAD output for each PROGRAM entry is the same for both of these commands. The only difference is the number of PROGRAM entries listed.
Figure 53 is an example of UNLOAD output for PROGRAM entries.

```
UCLIN.
REP      PROGRAM  ( PGM1 )
       LASTUPD   ( MYPROG1 )
       LASTUPDTYPE ( ADD )
       DISTLIB   ( ADSTLIB )
       SYSLIB    ( SPGMLIB )
       FMID      ( MYPROG1 )
       RMID      ( MYPROG1 )
REP      PROGRAM  ( PGM2 )
       LASTUPD   ( PGMPTF1 )
       LASTUPDTYPE ( ADD )
       DISTLIB   ( ADSTLIB )
       SYSLIB    ( SPGMLIB )
       FMID      ( MYPROG1 )
       RMID      ( PGMPTF1 )
ENDUCL.
```

Figure 53. PROGRAM entry: sample UNLOAD output

By specifying the FORFMID operand, you can reduce the number of PROGRAM entries unloaded. When FORFMID is specified, SMP/E unloads a PROGRAM entry only if its FMID matches one of the FMIDs specified on the FORFMID operand. For example, to unload PROGRAM entries whose FMIDs are either defined in FMISET ABC or are MYPROG1, you can use these commands:

```
SET BDY(TGT1) /* Set to target zone. */.
UNLOAD PROGRAM /* Unload all PROGRAM entries */.
       FORFMID(ABC /* for the ABC FMISET */.
       MYPROG1) /* and FMID MYPROG1. */.
```

**UCLIN Examples**

You can use the ADD, REP, and DEL UCL statements to change subentries in the PROGRAM entry. After the UCLIN changes are made, the PROGRAM entry must contain at least the following subentries:

- DISTLIB
- FMID
- RMID

Otherwise, there is not enough information in the entry to process the program element. If any of these subentries are missing, SMP/E does not make the requested UCL updates to the entry, and the entry remains as it was before the UCL command.

**Example: Adding a new PROGRAM entry**

Assume you have installed PGM3 outside of SMP/E, but now you want to start using SMP/E to track changes to that program element. Here is an example of the UCL statements you would use to define entries for that program element in the appropriate target and distribution zones:

```
SET BDY(TGT1) /* Set to target zone. */.
UCLIN /* */.
ADD      PROGRAM(PGM3) /* Define new PROGRAM entry. */
       DISTLIB(ADSTLIB) /* Define DLIB. */
       SYSLIB(SPGMLIB) /* System library. */
       FMID(MYPROG1) /* Functional owner. */
ENDUCL /* */.
```
PROGRAM entry (distribution and target zone)

SET BDY(DLB1) /* Now do same to DLIB. */.
UCLIN /* */.
ADD PROGRAM(PGM3) /* Define new PROGRAM entry. */
DISTLIB(ADSTLIB) /* Define DLIB. */
/* no SYSLIB info in DLIB. */
FMID(MYPROG1) /* Functional owner. */.
ENDUCL /* */.
The SRC entry describes source that exists in the distribution or target libraries. (SMP/E assumes that for each SRC entry in a particular zone there exists a MOD entry with the same name.) There are two ways a SRC entry can be created:

- **Installing a SYSMOD that contains the source.** SRC entries are created the first time you install a SYSMOD that contains a ++SRC statement for source that does not yet have a SRC entry.
- **Processing JCLIN.** SRC entries can be built during JCLIN processing when SMP/E scans the assembler step and determines that the assembler input is a member of a partitioned data set.
  SRC entries can also be built when SMP/E scans copy steps and finds a SELECT statement that specifies TYPE=SRC.

SMP/E records the function and service level of the source in the SRC entry, as well as information about how that source affects the structure of the distribution or target libraries and modules. Once a SRC entry exists for source, it is updated as subsequent SYSMODs that affect the source are installed.

### Subentries

These are the subentries for the SRC entry as they appear in the LIST output:

- **name**
  
  is the name of the source represented by the SRC entry.

  The name can contain from 1 to 8 alphanumeric characters and $, #, @, or hex C0.

- **DISTLIB**
  
  specifies the ddname of the distribution library for the source.

  The UCL operand is DISTLIB(ddname).

  The ddname can contain from 1 to 8 alphanumeric characters.

- **FMID**
  
  identifies the functional owner of this source. The functional owner is the last function SYSMOD that replaced this module.

  The UCL operand is FMID(sysmod_id).

  The SYSMOD ID must contain 7 alphanumeric characters.

- **LASTUPD**
  
  identifies the cause of the last change to this SRC entry.

  The UCL operand is LASTUPD(value). This subentry can contain one of the following values:

  - **JCLIN**
    
    indicates that the change was made during JCLIN command processing.

  - **UCLIN**
    
    indicates that the change was made as a result of UCLIN processing.

  - **sysmod_id**
    
    indicates that the change was made during the installation of the indicated SYSMOD.

    The SYSMOD ID must contain 7 alphanumeric characters.
SRC entry (distribution and target zone)

**LASTUPD TYPE**
indicates how the entry was last changed.

The UCL operand is `LASTUPDTYPE(value)`. This subentry can contain one of the following values:
- **ADD** The entry was added.
- **MOV** The entry was moved.
- **UPD** The entry was updated.

**RMID**
identifies the last SYSMOD that replaced this source. Any subsequent SYSMOD that modifies this module must have a defined relationship (such as PRE or SUP) with this SYSMOD.

The UCL operand is `RMID(sysmod_id)`.
- The SYSMOD ID must contain 7 alphanumeric characters.
- If **RMID** is not specified, but **FMID** is, SMP/E sets the RMID value to the specified FMID.

**SYSLIB**
specifies the ddname of the target library for the source.

The UCL operand is `SYSLIB(ddname)`.
- Only one SYSLIB value can be specified.
- The ddname can contain from 1 to 8 alphanumeric characters.

**UMID**
identifies all the SYSMODs that have updated this source since it was last replaced. Any subsequent SYSMOD that modifies this module must have a defined relationship (such as PRE or SUP) with all these SYSMODs.

The UCL operand is `UMID(sysmod_id...)`.
- The SYSMOD ID must contain 7 alphanumeric characters.

**LIST Examples**

To list all the SRC entries in a particular zone, you could use the following commands:

```
SET BDY(TGT1) /* Set to requested zone. */
LIST SRC /* List all SRC entries. */
```

To list specific SRC entries, you could use these commands:

```
SET BDY(TGT1) /* Set to requested zone. */
LIST SRC(SRC01 /* List only these two */
         SRC02) /* entries. */
```

The format of the LIST output for each SRC entry is the same for both of these commands. The only difference is the number of SRC entries listed. Figure 54 is an example of LIST output for SRC entries.
By specifying the FORFMID operand, you can reduce the number of SRC entries listed. When **FORFMID** is specified, SMP/E lists a SRC entry only if its FMID matches one of the FMIDs specified on the FORFMID operand. For example, to list SRC entries whose FMIDs either are defined in FMIDSET TP or are JXY1102, you could use these commands:

```
SET BDY(TGT1) /* Set to target zone. */
LIST SRC /* List all source entries */
FORFMID(TP) /* for the TP FMIDSET */
JXY1102 /* and FMID JXY1102. */
```

You can also use the LIST command to find out the name of every SYSMOD that has modified source. To include the names of these SYSMODs in the LIST output, you can use the XREF operand, as shown in these commands:

```
SET BDY(TGT1) /* Set to requested zone. */
LIST SRC /* List all source entries */
XREF /* and related SYSMODs. */
```

**Notes:**
1. XREF can be used either in mass mode or in select mode.
2. SMP/E obtains the data included for the XREF operand by checking for SRC and SRCUPD entries for this module in all the SYSMOD entries. Because this data is not contained in the SRC entry itself, you cannot use UCLIN to change it in the SRC entry.

**Figure 54. SRC entry: sample LIST output**

```
<table>
<thead>
<tr>
<th>NAME</th>
<th>LASTUPD</th>
<th>LIBRARIES</th>
<th>FMID</th>
<th>RMID</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRC01</td>
<td>JXY1102</td>
<td>DISTLIB=ASRCLIB SYSLIB=SRCLIB</td>
<td>JXY1102</td>
<td>JXY1102</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SRC02</td>
<td>JXY1000</td>
<td>DISTLIB=ASRCLIB SYSLIB=SRCLIB</td>
<td>JXY1121</td>
<td>UZ00010</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

**Figure 55 on page 317** is an example of the LIST output produced when the XREF operand is used.
**UNLOAD Examples**

To dump the SRC entries in UCL format, you can use the UNLOAD command. To unload all the SRC entries in a particular zone, you could use the following commands:

```sql
SET BDY(TGT1) /* Set to requested zone. */.
UNLOAD SRC /* Unload all SRC entries. */.
```

To unload specific SRC entries, you could use these commands:

```sql
SET BDY(TGT1) /* Set to requested zone. */.
UNLOAD SRC(SRC01 SRC02) /* Unload only these two entries. */.
```

The format of the UNLOAD output for each SRC entry is the same for both of these commands. The only difference is the number of SRC entries listed. Figure 56 is an example of UNLOAD output for SRC entries.

---

**Figure 55. SRC entry: sample LIST output when XREF is specified**

---

**Figure 56. SRC entry (distribution and target zone)**
By specifying the FORFMID operand, you can reduce the number of SRC entries unloaded. When FORFMID is specified, SMP/E unloads a SRC entry only if its FMID matches one of the FMIDs specified on the FORFMID operand. For example, to unload SRC entries whose FMIDs either are defined in FMIDSET TP or are JXY1102, you could use these commands:

```
SET BDY(TGT1) /* Set to target zone. */
UNLOAD SRC /* Unload all source entries*/
FORFMID(TP /* for the TP FMIDSET */
   JXY1102) /* and FMID JXY1102. */.
```

### UCLIN Examples

You can use the ADD, REP, and DEL UCL statements to change subentries in the SRC entry. After the UCLIN changes are done, the SRC entry must contain at least the following subentries:
- DISTLIB
- FMID
- RMID

Otherwise, there is not enough information in the entry to process the macro. If any of these subentries are missing, SMP/E does not make the requested UCL updates to the entry, and the entry remains as it was before the UCL command.

The following examples are provided to help you use the SRC entry.

**Example 1: Adding a new SRC entry**

Assume that you have a user application installed and want to support it with SMP/E. To do this, you must record the modules in the target and distribution zones. You should identify the MOD and LMOD entries for your application by running the JCLIN function, using the appropriate link and copy steps as input. The source elements in your application will have to be entered through UCLIN. Assume that the FMID you wish to assign the product is ZUSR001 and that no service level is to be recorded. The following UCL statements should be used for each such source:

```
SET BDY(TGT1) /* Set to target zone. */
UCLIN /* */.
ADD SRC(SRC01) /* Define new source entry. */
DISTLIB(ASRCLIB) /* Define DLIB, */
```

Figure 56. SRC entry: sample UNLOAD output

UCLIN Examples

You can use the ADD, REP, and DEL UCL statements to change subentries in the SRC entry. After the UCLIN changes are done, the SRC entry must contain at least the following subentries:
- DISTLIB
- FMID
- RMID

Otherwise, there is not enough information in the entry to process the macro. If any of these subentries are missing, SMP/E does not make the requested UCL updates to the entry, and the entry remains as it was before the UCL command.

The following examples are provided to help you use the SRC entry.

**Example 1: Adding a new SRC entry**

Assume that you have a user application installed and want to support it with SMP/E. To do this, you must record the modules in the target and distribution zones. You should identify the MOD and LMOD entries for your application by running the JCLIN function, using the appropriate link and copy steps as input. The source elements in your application will have to be entered through UCLIN. Assume that the FMID you wish to assign the product is ZUSR001 and that no service level is to be recorded. The following UCL statements should be used for each such source:

```
SET BDY(TGT1) /* Set to target zone. */
UCLIN /* */.
ADD SRC(SRC01) /* Define new source entry. */
DISTLIB(ASRCLIB) /* Define DLIB, */
```
Example 2: Recording the application of a corrective fix

Assume that you have installed a corrective fix, AZ12345, to source SRC01, outside of SMP/E, and that now you want to record that fix after it has been installed. (This is not necessary if the fix was initially installed with SMP/E.) Three updates must be made:
1. Record that the fix, AZ12345, is in the system.
2. Record that the fix is on the source.
3. Record that the source has been reassembled and installed.

The following UCL can be used to record these changes:

```
SET BDY(TGT1) /* Set to target zone. */.
UCLIN /* */.
ADD SYSMOD(AZ12345) /* Add SYSMOD entry. */
    APAR /* Corrective fix. */
    APPTIME(08:00:00) /* Time applied. */
    FMID(FXY1102) /* Functional owner. */
    SRCUPD(SRC01) /* Updated SRC01. */
    ASSEM(SRC01) /* Assembled it too. */
    SRC(SRC01) /* Updated SRC01. */
    UMID(AZ12345) /* Add update ID of APAR. */
    RMID(AZ12345) /* Update MOD entry with new replacement ID. */
    RMIDASM /* Was assembled. */
REP SRC(SRC01) /* Update SRC01 to */
    UMID(AZ12345) /* Add update ID of APAR. */
    RMID(AZ12345) /* Update MOD entry */
    RMIDASM /* Was assembled. */
ENDUCL /* */.
```
STSSRC entry (SMPSTS)

The STSSRC entry is a copy of source that resides only in a distribution library but is needed temporarily during APPLY processing. The STSSRC entry is in the SMPSTS data set, which serves as a target source library for such modules.

When SMP/E applies the SYSMODs that affect these source, it calls utility programs to store the modules on the SMPSTS. This way, the most current service level of each module is available for use in assemblies. After SMP/E has accepted all the SYSMODs that affect these source modules, it deletes the associated STSSRC entries from the SMPSTS.

Note: If you specify SAVESTS in the ORDER entry that is in effect during ACCEPT processing, SMP/E will not delete STSSRC entries from the SMPSTS after the SYSMODs that affect those source have been successfully accepted.

Subentries
The STSSRC entry contains no SMP/E data and appears to the system as a member of a normal source library.

LIST Examples
You cannot use SMP/E to list the STSSRC entries. However, you can use standard system utility programs (such as IEBGENER, IEBPTPCH, and IEHLIST) or products such as ISPF to display these entries or information about the data set.

UCLIN Examples
You can use the DEL UCL statement to delete an STSSRC entry from the SMPSTS. This can be helpful if you plan to do an APPLY followed by ACCEPT when several target libraries have been created from the same distribution library.

When a SYSMOD is accepted into a distribution zone, the entries associated with it are automatically deleted from the SMPSTS for the related target zone. However, even if the SYSMOD was also applied to other target zones created from the same distribution zone, SMP/E does not clean up the SMPSTS data sets for the other target zones.

To delete the entries from these data sets, you could accept the SYSMOD and name these other target zones as the related zone. However, this would update the distribution library each time; this is time-consuming and could use up space in the distribution library data set.

Instead, you can use the DEL command to delete these entries without updating the distribution library. To determine which entries to specify, check the SMPLOG data set to see which ones SMP/E deleted during ACCEPT processing.

Note: You can also use the CLEANUP command to delete STSSRC entries without specifying them individually. For more information, see the CLEANUP command in SMP/E Commands.

Example: Deleting an STSSRC entry
Assume that you have two target zones, TGT1 and TGT2, generated off the same distribution zone, DLB1. During ACCEPT processing of a SYSMOD, SMP/E has deleted STSSRC SRC01 and SRC02 from the SMPSTS data set associated with target zone TGT2. After performing the ACCEPT, you want to delete the same
source from the SMPSTS associated with target zone TGT1. Assume either that you have a cataloged procedure for TGT1 with the correct SMPSTS specified, or that you have set up the correct DDDEF entries. You can use the following UCLIN to delete the STSSRC entry:

```
SET BDY(TGT1) /* Set to TGT1 zone. */.
UCLIN /* */.
DEL STSSRC(SRC01) /* Delete the source. */.
DEL STSSRC(SRC02) /* Delete the source. */.
ENDUCL /* */.
```

Note: One UCL statement is required for each STSSRC entry to be deleted.

You can make the same changes by using system utilities; however, the SMPLOG will not reflect the processing done.
SYSMOD entry (distribution and target zone)

The SYSMOD entry in a distribution zone or a target zone describes a SYSMOD that has been installed in the corresponding distribution library or target library. This SYSMOD entry contains the same information as the global zone SYSMOD entry, except that it has information from only one ++VER statement, the one to install the SYSMOD.

When SMP/E installs a SYSMOD, it uses SYSMOD entries in the distribution or target zone to do the following:

- Determine the functional level of the system. SMP/E checks which function SYSMODs have been installed, and then uses that information to determine which service SYSMODs may be applicable.
- Determine the service level of the system. SMP/E checks which service SYSMODs have been installed.
- Make sure the requisites are satisfied for each SYSMOD to be installed. SMP/E checks whether a SYSMOD entry exists in the distribution or target zone for each requisite.

Subentries

These are the subentries for the SYSMOD entry as they appear in the LIST output:

- **sysmod_id**
  is the SYSMOD identification.
  The SYSMOD ID must contain 7 alphanumeric characters.
  
- **ACCEPT**
  indicates that the SYSMOD has been successfully accepted.
  The UCL operand is ACCEPT, ACPT, or ACC.
  
  **Note:** This subentry exists only in the distribution zone. It is required in distribution zone SYSMOD entries.

- **APAR**
  indicates that this SYSMOD is an APAR, which provides a corrective fix to a problem.
  The UCL operand is APAR.
  APAR, FUNCTION, PTF, and USERMOD are mutually exclusive. If none of these operands is specified, PTF is the default.

- **APPLY**
  indicates that the SYSMOD has been successfully applied.
  The UCL operand is APPLY, APPL, or APP.

  **Note:** This subentry exists only in the target zone. It is required in target zone SYSMOD entries.

- **ASSEM**
  lists the assemblies done during the installation of this SYSMOD.
  The UCL operand is ASSEM(name...).
  The name can contain from 1 to 8 alphanumeric characters.
SYSMOD entry (distribution and target zone)

**BYPASS**
indicates that the BYPASS operand was specified when this SYSMOD was installed.

The UCL operand is **BYPASS**.

**CIFREQ**
lists the conditional requisites that must be installed when this function SYSMOD is installed.

Note: The data specified is used by SMP/E only when present in a function SYSMOD.

The UCL operand is **CIFREQ((causer,req)...)**.

- **causer** is the SYSMOD that specified this function SYSMOD on the FMID operand of an ++IF statement. **req** is the SYSMOD specified on the REQ operand as the conditional requisite associated with this function SYSMOD.
- The **causer** and **req** fields must contain 7 alphanumeric characters each.
- The CIFREQ operand is mutually exclusive with all other UCL operands. It will not cause LASTUPD and LASTUPDTYPE to be updated.

**DELBY**
specifies the SYSMOD that deleted this SYSMOD.

The UCL operand is **DELBY(sysmod_id)**.

- The SYSMOD ID must contain 7 alphanumeric characters.
- This subentry is valid only for function SYSMODs.
- The DELBY operand is mutually exclusive with all other UCL operands.

**DELETE**
lists the SYSMODs deleted by this SYSMOD.

The UCL operand is **DELETE(sysmod_id...)**.

The SYSMOD ID must contain 7 alphanumeric characters.

**DELLMOD**
indicates that the SYSMOD contained a ++DELETE statement.

The UCL operand is **DELLMOD**.

**DESCRIPTION**
specifies the descriptive name to be associated with this SYSMOD.

The UCL operand is **DESCRIPTION(description)**.

**DLMOD**
lists the load modules deleted by ++DELETE statements contained in this SYSMOD.

The UCL operand is **DLMOD(name...)**.

**element**
lists the data element replacements contained in the SYSMOD.

The UCL operand is **element(name...)**.

- The name can contain from 1 to 8 alphanumeric characters.
- **In place of** **element**, specify one of the replacement values shown in Table 2 on page 10.
- Some types of elements, such as panels, messages, or text, may have been translated into several languages. In these cases, the **element** operand contains **xxx**, which represents the language used for the element. (If an
**SYSMOD entry (distribution and target zone)**

- **element was not translated**, the `element` operand does not contain any `xxx` value.) Table 3 on page 13 shows the `xxx` values and the languages they represent.

**ELEMMOV**
- indicates that the SYSMOD contained a `++MOVE` statement.
  - The UCL operand is `ELEMMOV`.

**EMOVE**
- lists the elements and load modules that were moved by `++MOVE` statements contained in this SYSMOD.
  - The UCL operand is `EMOVE(name...)`.

**ERROR**
- indicates that an error has occurred during the processing of this SYSMOD.
  - The UCL operand is `ERROR`.
  - This operand can also be specified as `ERR`.
  - If the RESTORE subentry is set, the RESDATE operand must be specified, and the RESTIME and ERROR operands should be specified (SMP/E will automatically set it otherwise).

**FEATURE**
- lists the names of the software features that contain this SYSMOD. The identified software features correspond to FEATURE entries in the global zone.
  - The UCL operand is `FEATURE(name,...)`.

**FESN**
- identifies the field engineering (FE) service number.
  - The UCL operand is `FESN(string)`.
  - The string must contain 7 alphanumeric characters.

**FMID**
- identifies the function SYSMOD to which this SYSMOD is applicable.
  - The UCL operand is `FMID(sysmod_id)`.
  - The SYSMOD ID must contain 7 alphanumeric characters.

**FUNCTION**
- indicates that this SYSMOD is a function, which introduces a new function into the system.
  - The UCL operand is `FUNCTION`.
  - APAR, FUNCTION, PTF, and USERMOD are mutually exclusive. If none of these operands is specified, PTF is the default.

**hfs_element**
- lists the hierarchical file system element replacements in the SYSMOD.
  - The UCL operand is `hfs-element(name...)`.
  - The name can contain 1 to 8 uppercase alphabetic, numeric, or national ($, #, @) characters.

**IFREQ**
- lists the conditional requisites that were installed with this SYSMOD.
  - The UCL operand is `IFREQ(sysmod_id...)`.


**INSTALLDATE**

specifies the date on which this SYSMOD was installed.

The UCL operand is **INSTALLDATE(yyddd)**.

- This operand can also be specified as **INSDATE**. For a distribution zone SYSMOD entry, it can be specified as **ACCDATE**. For a target zone SYSMOD entry, it can be specified as **APPDATE**.
- The yyddd must contain 5 numeric characters. SMP/E does not check whether the specified numbers are valid.
- For the distribution zone SYSMOD entry, INSTALLDATE is the date the SYSMOD was accepted. For the target zone SYSMOD entry, INSTALLDATE is the date the SYSMOD was applied.

**INSTALLTIME**

specifies the time at which this SYSMOD was installed.

The UCL operand is **INSTALLTIME(hh:mm:ss)**.

- This operand can also be specified as **INSTIME**. For a distribution zone SYSMOD entry, it can be specified as **ACCTIME**. For a target zone SYSMOD entry, it can be specified as **APPTIME**.
- The hh, mm, and ss must contain 2 numeric characters each. The “:” must be coded as specified. SMP/E does not check whether the specified numbers are valid.
- For a distribution zone SYSMOD entry, INSTALLTIME is the time the SYSMOD was accepted. For a target zone SYSMOD entry, INSTALLTIME is the time the SYSMOD was applied.

**JAR**

lists the JAR file replacements (++JAR) supplied by the SYSMOD.

The UCL operand is **JAR**.

**JARUPD**

lists the JAR file updates (++JARUPD) supplied by the SYSMOD.

The UCL operand is **JARUPD**.

**JCLIN**

indicates that the SYSMOD contained inline JCLIN.

The UCL operand is **JCLIN**.

**LASTSUP**

specifies the most recent SYSMOD that superseded this SYSMOD. All previous superseding SYSMODs are saved in the SUPBY subentry list.

The UCL operand is **LASTSUP(sysmod_id)**.

The SYSMOD ID must contain 7 alphanumeric characters.

**LASTUPD**

identifies the cause of the last change to this entry.

The UCL operand is **LASTUPD(UCLIN)**, indicating that the change was made as a result of UCLIN processing.

**LASTUPDTYPE**

identifies the last type of update made to this entry.

The UCL operand is **LASTUPDTYPE(value)**. This subentry can contain one of the following values:

- **ADD** The entry was added.
SYSMOD entry (distribution and target zone)

**UPD**  The entry was updated.

**MAC**  
lists the macro replacements (++MAC statements) in the SYSMOD.

The UCL operand is **MAC(name...)**.

The name can contain from 1 to 8 alphanumeric characters.

**MACUPD**  
lists the macro updates (++MACUPD statements) in the SYSMOD.

The UCL operand is **MACUPD(name...)**.

The name can contain from 1 to 8 alphanumeric characters.

**MOD**  
lists the module replacements (++MOD statements) in the SYSMOD.

The UCL operand is **MOD(name...)**.

The name can contain from 1 to 8 alphanumeric characters.

**NPRE**  
lists negative prerequisite SYSMODs (that is, SYSMODs that must not be present in the system at the same time as this SYSMOD).

The UCL operand is **NPRE(sysmod_id...)**.

- The SYSMOD ID must contain 7 alphanumeric characters.
- This subentry is valid only for function SYSMODs.

**PRE**  
lists the prerequisite SYSMODs (that is, SYSMODs that must be present before this SYSMOD can be installed).

The UCL operand is **PRE(sysmod_id...)**.

The SYSMOD ID must contain 7 alphanumeric characters.

**PROGRAM**  
lists the program element replacements (++PROGRAM statements) in the SYSMOD.

The UCL operand is **PROGRAM(name...)**.

The name can contain from 1 to 8 alphanumeric characters.

**PTF**  
indicates that this SYSMOD is a PTF, which provides preventive service fixes.

The UCL operand is **PTF**.

APAR, FUNCTION, PTF, and USERMOD are mutually exclusive. If none of these operands is specified, **PTF** is the default.

**RECDATE**  
specifies the date on which this SYSMOD was received.

The UCL operand is **RECDATE(yyddd)**.

The **yyddd** must contain 5 numeric characters. SMP/E does not check whether the specified numbers are valid.

**RECTIME**  
specifies the time at which this SYSMOD was received.

The UCL operand is **RECTIME(hh:mm:ss)**.
SYSMOD entry (distribution and target zone)

The hh, mm, and ss must contain 2 numeric characters each. The “:” must be coded as specified. SMP/E does not check whether the specified numbers are valid.

REGEN
indicates how the SYSMOD was installed in the target libraries.
- In a DLIB zone SYSMOD entry, REGEN is not important. It is automatically set for all SYSMODs when they are accepted.
- In a target zone SYSMOD entry, if REGEN is set, it indicates that SYSGEN was used to install the SYSMOD in the target libraries. (The REGEN indicator was carried over when the distribution zone was copied into the target zone. This is generally done as part of SYSGEN.)
  If REGEN is not set, it indicates that the APPLY command was used to install the SYSMOD in the target libraries.

The UCL operand is REGEN or RGN.

RENLMOD
indicates that the SYSMOD contained a ++RENAME statement.

The UCL operand is RENLMOD.

REQ
lists requisite SYSMODs (that is, SYSMODs that must be installed concurrent with this SYSMOD).

The UCL operand is REQ(sysmod_id...).

The SYSMOD ID must contain 7 alphanumeric characters.

RESDATE
specifies the date on which this SYSMOD was restored.

The UCL operand is RESDATE(yyddd).
- This subentry exists only in the target zone.
- The yyddd must contain 5 numeric characters. SMP/E does not check whether the specified numbers are valid.
- If a SYSMOD is marked “RESTORE”, the RESDATE operand must be specified, and the RESTIME and ERROR operands should be specified (SMP/E will automatically set it otherwise).

RESTIME
specifies the time that this SYSMOD was restored.

The UCL operand is RESTIME(hh:mm:ss).
- This subentry exists only in the target zone.
- The hh, mm, and ss must contain 2 numeric characters each. The “:” must be coded as specified. SMP/E does not check whether the specified numbers are valid.
- If a SYSMOD is marked “RESTORE”, the RESDATE operand must be specified, and the RESTIME and ERROR operands should be specified (SMP/E will automatically set it otherwise).

RESTORE
indicates that a RESTORE attempt has been made for this SYSMOD. The RESTORE was not successful; otherwise, the SYSMOD entry would have been deleted.

The UCL operand is RESTORE.
- This subentry exists only in the target zone.
SYSMOD entry (distribution and target zone)

- This operand can also be specified as **REST** or **RES**.
- If a SYSMOD is marked “RESTORE”, the RESDATE operand **must** be specified, and the RESTIME and ERROR operands **should** be specified (SMP/E will automatically set it otherwise).

**REWORK**
identifies the level of the SYSMOD, which was received again for minor changes.
The UCL operand is **REWORK**(level).
- Up to 8 numeric characters can be specified.
- For SYSMODs supplied by IBM, the REWORK level is yyyyddd, where yyyy is the year the SYSMOD was reworked and ddd is the Julian date.
- SMP/E does not check whether this data is valid.

**RLMOD**
indicates the load modules renamed by ++RENAME statements in this SYSMOD.
The UCL operand is **RLMOD**(name...).

**SOURCEID**
lists the character strings assigned to this SYSMOD during RECEIVE. These values might have been specified by the user on the RECEIVE command, included inline on the ++ASSIGN statement, or assigned by SMP/E when processing the contents of an order.
The UCL operand is **SOURCEID**(source_id...).
- The source ID can be from 1 to 64 characters in length and contain any nonblank character (X’41’ through X’FE’) except single quotation mark (‘), asterisk (*), percent (%), comma (,), left parenthesis (()), and right parenthesis ()),
- A source ID value cannot span lines.
- A source ID value is case sensitive.

**SRC**
lists the source replacements (++SRC statements) in the SYSMOD.
The UCL operand is **SRC**(name...).
The name can contain from 1 to 8 alphanumeric characters.

**SRCUPD**
lists the source updates (++SRCUPD statements) in the SYSMOD.
The UCL operand is **SRCUPD**(name...).
The name can contain from 1 to 8 alphanumeric characters.

**SUPBY**
lists the SYSMODs that superseded this SYSMOD. For functions, this includes SYSMODs that both deleted and superseded this SYSMOD.

**Notes:**
1. The most recent SYSMOD to supersede this SYSMOD is not included in the SUPBY list. It is saved in the LASTSUP field.
2. The SUPBY field may appear as “SUPBY(IN SYSMD)”. For example, this is the case if the superseding SYSMODs were installed separately instead of on the same APPLY or ACCEPT command.

The UCL operand is **SUPBY**(sysmod_id...).
SYSMOD entry (distribution and target zone)

- This operand can also be specified as SUP.
- The SYSMOD ID must contain 7 alphanumeric characters.

SUPING
lists the SYSMODs superseded by this SYSMOD.
The UCL operand is SUPING(sysmod_id...).
The SYSMOD ID must contain 7 alphanumeric characters.

SZAP
lists the module superzaps (++ZAP statements) in the SYSMOD.
The UCL operand is SZAP(name...).
The name can contain from 1 to 8 alphanumeric characters.

UCLDATE
specifies the date on which this SYSMOD was last modified through UCLIN.
The UCL operand is UCLDATE(yyddd).
The yyddd must contain 5 numeric characters. SMP/E does not check whether the specified numbers are valid.

UCLTIME
specifies the time that this SYSMOD was last modified through UCLIN.
The UCL operand is UCLTIME(hh:mm:ss).
The hh, mm, and ss must contain 2 numeric characters each. The “:” must be coded as specified. SMP/E does not check whether the specified numbers are valid.

USERMOD
indicates that this SYSMOD is a USERMOD, which puts a user modification in the system.
The UCL operand is USERMOD.
APAR, FUNCTION, PTF, and USERMOD are mutually exclusive. If none of these operands is specified, PTF is the default.

VERNUM
specifies the relative number of the ++VER statement used when this SYSMOD was installed.
The UCL operand is VERNUM(nnn).
- nnn can contain from 1 to 3 numeric characters.
- When updating an existing entry, you should not specify VERNUM. This causes SMP/E to assume the same VERNUM value as in the current entry.
- If you do not specify VERNUM when adding a new entry, SMP/E assumes a VERNUM value of 1.
- The VERNUM values are kept in each entry built from information from the ++VER statements. (For example, subentries such as PRE and REQ have unique VERNUM values.) If all entries do not have the same VERNUM, an error will result.

VERSION
lists the function SYSMODs that are versioned by this SYSMOD. Versioning indicates that, if there are any elements in common between the SYSMODs listed and this SYSMOD, this SYSMOD’s elements are at a higher functional level, and are thus the ones that should be installed.
SYSMOD entry (distribution and target zone)

The UCL operand is `VERSION(sysmod_id...)`.
The SYSMOD ID must contain 7 alphanumeric characters.

**XZAP**
lists the module superzaps in the SYSMOD (++ZAP statements) that contain an EXPAND statement (indicating that the module should be expanded before it is updated).

The UCL operand is `XZAP(name...)`.
The name can contain from 1 to 8 alphanumeric characters.

**LIST Examples**
To list all the SYSMOD entries in a particular zone, you could use the following commands:

```plaintext
SET DB(Y)TGT1 /* Set to requested zone. */.
LIST SYSMOD /* List all SYSMOD entries. */.
```

To list specific SYSMOD entries, you could use these commands:

```plaintext
SET DB(Y)TGT1 /* Set to requested zone. */.
LIST SYSMOD(UZ00001 /* List only these two */
            UZ00002) /* entries. */.
```

The format of the LIST output for each SYSMOD entry is the same for both of these commands. The only difference is the number of SYSMOD entries listed. 

[Figure 57 on page 331](#) and [Figure 58 on page 332](#) are examples of LIST output for SYSMOD entries.
SYSMOD entry (distribution and target zone)

Figure 57. SYSMOD entry: sample LIST output for a distribution zone
### SYSMOD entry (distribution and target zone)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
<th>Status</th>
<th>Fmid</th>
<th>Date/Time Rec</th>
<th>Ins</th>
<th>LastSup</th>
<th>ZAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>AZ99001</td>
<td>APAR</td>
<td>Sample APAR</td>
<td>REC APP</td>
<td>HZY2102</td>
<td>07.100 08:00:00</td>
<td>07.100 08:00:00</td>
<td>UZ00010</td>
<td>MOD01</td>
</tr>
<tr>
<td>AZ99002</td>
<td>SUPERSEDED</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>UZ00010</td>
<td></td>
</tr>
<tr>
<td>LUS0001</td>
<td>USERMOD</td>
<td>Sample user modification</td>
<td>REC APP</td>
<td>HZY2102</td>
<td>06.100 08:00:00</td>
<td>06.100 08:09:00</td>
<td>UZ00010</td>
<td></td>
</tr>
<tr>
<td>UZ00008</td>
<td>PTF</td>
<td>Sample PTF</td>
<td>REC APP</td>
<td>HZY2102</td>
<td>06.100 08:00:00</td>
<td>06.100 08:08:00</td>
<td>UZ00010</td>
<td></td>
</tr>
<tr>
<td>UZ00009</td>
<td>SUPERSEDED</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>UZ00010</td>
<td></td>
</tr>
<tr>
<td>UZ00010</td>
<td>PTF</td>
<td></td>
<td>REC APP</td>
<td>HZY2102</td>
<td>06.100 08:00:00</td>
<td>06.100 08:10:00</td>
<td>UZ00008</td>
<td>UZ00007</td>
</tr>
<tr>
<td>UZ00011</td>
<td>PTF</td>
<td></td>
<td>REC APP</td>
<td>HXY2102</td>
<td>07.250 08:00:00</td>
<td>07.250 08:10:00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 58. SYSMOD entry: sample LIST output for a target zone
**SYSMOD entry (distribution and target zone)**

By specifying various operands, you can reduce the number of SYSMOD entries listed. If you specify any of these operands on the LIST command, SMP/E automatically assumes that SYSMOD entries are to be processed, regardless of whether the SYSMOD operand was also specified. If you specify more than one of these operands on the same LIST command, only the SYSMODs that meet all the specified conditions are processed. For more information about these operands, see [SMP/E Commands](#).

You can also use the LIST command to find any other SYSMODs that specify this SYSMOD in their DEL, PRE, REQ, NPRE, SUP, or VERSION lists. To include the names of these SYSMODs in the LIST output, you can use the XREF operand, as shown in these commands:

```
SET  BDY(DLIB1) /* Set to requested zone. */.
LIST SYSMOD /* List all SYSMOD entries */.
        XREF /* and SYSMOD that hit them. */.
```

**Note:** XREF can be used either in mass mode or in select mode.

*Figure 59* is an example of the LIST output produced when the XREF operand is used.

```plaintext
PAGE nnnn - NOW SET TO zzzzzz ZONE nnnnnn DATE mm/dd/yy TIME hh:mm:ss SMP/E 36.nn SMPLIST OUTPUT

DLIB1  SYSMOD ENTRIES

NAME

UZ12345 TYPE = PTF
    STATUS = REC
    FMID = HZY2100
    DATE/TIME REC = 06.150 08:00:00
    INS = 06.160 08:08:00
    SUPING VER(001) = AZ11111 AZ11112 AZ11113
    MAC = MAC01 MAC02 MAC03
    MOD = MOD01 MOD02 MOD03 MOD04
    SRC = SRC01 SRC02
    NPREBY (XREF) = UZ00001 UZ00002
    PREBY (XREF) = UZ00003 UZ00004
    REQBY (XREF) = UZ00005 UZ00006
    VERSIONBY(XREF) = UZ00007 UZ00008
    DELBY (XREF) = UZ00009 UZ00010
    IFREQBY (XREF) = UZ00011 UZ00012
    SUPBY (XREF) = UZ00013 UZ00014

Figure 59. SYSMOD entry: sample LIST output when XREF is specified

**Note:** Some of the xxxBY subentries listed in this example are not actually valid for the type of SYSMOD shown (that is, PTF). For example, a SYSMOD does not specify VERSION, NPRE, or DELETE for a PTF SYSMOD. Those subentries are included here only to show the format of the output.

**UNLOAD Examples**

To dump the SYSMOD entries in UCL format, you can use the UNLOAD command. To unload all the SYSMOD entries in a particular zone, you could use the following commands:

```
SET  BDY(TGT1) /* Set to requested zone. */.
UNLOAD SYSMOD /* Unlock all SYSMOD entries. */.
```
To unload specific SYSMOD entries, you could use these commands:

```
SET BDY(TGT1) /* Set to requested zone. */
UNLOAD SYSMOD(UZ00001 /* Unload only these two */
   UZ00002) /* entries. */
```

The format of the UNLOAD output for each SYSMOD entry is the same for both of these commands. The only difference is the number of SYSMOD entries listed. Figure 60 is an example of UNLOAD output for one SYSMOD.

```
UCLIN.
REP  SYSMOD ( HZY2102 ) /* TYPE */ FUNCTION
    DESCRIPTION ( Sample Function )
    FEATURE   ( SMPRG1 ) /* STATUS */ ACC
    RECDATA   ( 07100 )
    RECRTME   ( 08:08:00 )
    INSDATE   ( 07101 )
    INSTIME   ( 10:00:34 )
    VERNUM    ( 002 )
    PRE       ( UZ00001 UZ00002 UZ00003 )
    REQ       ( UZ00004 UZ00005 UZ00006 )
    SUPING    ( AZ00001 AZ00002 AZ00003 )
    MOD       ( MOD01 MOD02 )
    MAC       ( MAC01 )
    SRC       ( SRC01 SRC02 SRC03 )

REP  SYSMOD ( HZY2102 ) CIFREQ ( UZ00087 UZ00088 )
        ( UZ00090 UZ00090 )
ENDUCL.
```

By specifying various operands, you can reduce the number of SYSMOD entries unloaded. If you specify any of these operands on the UNLOAD command, SMP/E automatically assumes that SYSMOD entries are to be processed. If you specify more than one of these operands on the same UNLOAD command, only those SYSMODs that meet all the specified conditions are processed. For more information about these operands, see SMP/E Commands.

**UCLIN Examples**

You can use the ADD, REP, and DEL UCL statements to change subentries in the SYSMOD entry. Generally, after UCLIN changes are done, the SYSMOD entry must contain at least these subentries, unless the entire entry has been deleted:

- ACCEPT or APPLY
- APAR, FUNCTION, PTF, or USERMOD
- INSTALLDATE
- RECDATA
- FMID

Otherwise, there is not enough information in the entry to process the SYSMOD. If any of the required subentries are missing, SMP/E does not make the requested UCL updates to the entry, and the entry remains as it was before the UCL command.
SYSMOD entry (distribution and target zone)

Note: The entry for a deleted SYSMOD can contain just the DELBY subentry. The entry for a superseded SYSMOD can contain just the SUPBY subentry. The entry for a SYSMOD that is both deleted and superseded can contain just the DELBY and SUPBY subentries.

The following examples are provided to help you use the SYSMOD entry.

Example 1: Creating a SYSMOD entry
Assume that you have installed an APAR fix outside of SMP/E and now wish to get that fix recorded in the distribution zone. The APAR number was AZ12345, and was an update to a source SRC01. The following UCL should be used:

```plaintext
SET BDY(DLIB1) /* Set to DLIB zone. */.
UCLIN /* */.
ADD SYSMOD(AZ12345) /* Specify SYSMOD. */.
SRCUPD(SRC01) /* Changed this source. */.
RECDATE(06100) /* Received on this date. */.
RESDATE(06100) /* Accepted same day. */.
ASSEM(SRC01) /* Assembled source. */.
FMID(FZY2102) /* Functional owner. */.
ADD SRC(SRC01) /* Now update source entry */.
UMID(AZ12345) /* with update ID. */.
ADD MOD(SRC01) /* Now update MOD */.
RMID(AZ12345) /* with new replacement ID */.
RMIDASM /* from assembly. */.
ENDUCL /* */.
```

Example 2: Removing the ERROR indicator
Assume that during an APPLY you encountered an error that caused a PTF to be marked as in error. After looking at the output you determine that the PTF actually installed correctly, and rather than reinstall the PTF you decide to make the appropriate changes to make the PTF look installed. The following UCL can be used:

```plaintext
SET BDY(TGT1) /* Set to target zone. */.
UCLIN /* */.
DEL SYSMOD(UZ12345) /* Specify SYSMOD. */.
RESTORE /* Delete restore. */.
RESDATE() /* Delete restore date. */.
RESDATE() /* Delete restore time. */.
ADD SYSMOD(UZ12345) /* Specify SYSMOD. */.
APPLY /* Add apply info. */.
ENDUCL /* */.
```

Note: This method is very prone to errors. The preceding example gets the SYSMOD marked as having been applied; however, you have not made all the changes necessary to get the rest of the target zone entries coordinated. Those changes include updating the RMID and UMID fields of all the elements affected by the PTF, storing superseded SYSMOD entries, and updating the global zone SYSMOD entry.

If you are not extremely familiar with SMP/E internals and how to complete the update process, the recommended method is to reapply the PTF.
The global zone SYSMOD entry describes a SYSMOD that exists as an MCS entry in the SMPPTS. It contains information that SMP/E obtained when it received that SYSMOD. Because SMP/E processing is designed to keep the global zone SYSMOD entry and the MCS entry synchronized, you should use only SMP/E commands to update these entries. If you use system utilities to update the MCS entries, you will get unpredictable results when processing the corresponding SYSMOD. For more information about MCS entries, see "MCS entry (SMPPTS)" on page 273.

When SMP/E processes SYSMODs, it uses the global zone SYSMOD entry to determine which SYSMODs are applicable or whether the SYSMODs you selected are applicable. In addition, after a SYSMOD has been successfully applied or accepted, SMP/E records in the SYSMOD entry the names of the zones to which the SYSMOD was applied or accepted.

Note: Although this information is saved in the global zone SYSMOD entry, it is used only for reporting purposes. SMP/E does not use it during APPLY, ACCEPT, or RESTORE processing to determine the status of a SYSMOD. Instead, SMP/E uses SYSMOD entries in the target and distribution zones to determine whether a SYSMOD has been applied or accepted.

A SYSMOD entry is generally kept in the global zone until the associated SYSMOD is accepted; then the entry is deleted. You may want SMP/E to save the global zone SYSMOD entries after ACCEPT processing (for example, if you plan to do a system generation). To do this, specify NOPURGE in the ORDER entry that is in effect during ACCEPT processing. Likewise, you may want to save the global zone SYSMOD entries after RESTORE processing. To do this, specify NOREJECT in the ORDER entry that is in effect during RESTORE processing.

Subentries

These are the subentries for the global zone SYSMOD entry as they appear in the LIST output. Only a few of these subentries can be changed with UCLIN. The description of each entry indicates whether UCLIN may be used and, if so, what the correct syntax is.

sysmod_id

is the SYSMOD identification.

The SYSMOD ID must contain 7 alphanumeric characters.

ACCEPT ZONE

lists the distribution zones into which the SYSMOD has been successfully accepted.

The UCL operand is ACCID(zone...).

The name can contain from 1 to 7 alphanumeric characters.

APAR

indicates that this SYSMOD is an APAR, which provides a corrective fix to a problem.

There is no UCL support for this subentry in the global zone.

APPLY ZONE

lists the target zones to which the SYSMOD has been successfully applied.

The UCL operand is APPID(zone...).
SYSMOD entry (global zone)

The name can contain from 1 to 7 alphanumeric characters.

**DELLMOD**

indicates that the SYSMOD contained a ++DELETE statement.

The UCL operand is **DELLMOD**.

**DELETE**

lists the SYSMODs deleted by this SYSMOD.

There is no UCL support for this subentry in the global zone.

**DESCRIPTION**

specifies the descriptive name to be associated with this SYSMOD.

The UCL operand is **DESCRIPTION(description)**.

**DLMOD**

lists the load modules deleted by ++DELETE statements contained in this SYSMOD.

The UCL operand is **DLMOD(name...)**.

**element**

lists the data element replacements in the SYSMOD.

There is no UCL support for this subentry in the global zone.

In place of **element**, you will see one of the values shown in Table 2 on page 10. Some types of elements, such as panels, messages, or text, may have been translated into several languages. In these cases, the **element** operand contains **xxx**, which represents the language used for the element. (If an element was not translated, the **element** operand does not contain any **xxx** value.) Table 3 on page 13 shows the **xxx** values and the languages they represent.

**ELEMMOV**

indicates that the SYSMOD contained a ++MOVE statement.

The UCL operand is **ELEMMOV**.

**EMOVE**

lists the elements and load modules that were moved by ++MOVE statements contained in this SYSMOD.

**ERROR**

indicates that an error occurred when this SYSMOD was received.

There is no UCL support for this subentry in the global zone.

**FEATURE**

lists the names of the software features that contain this SYSMOD. The identified software features correspond to FEATURE entries in the global zone.

The UCL operand is **FEATURE(name, ...)**.

**FESN**

identifies the field engineering (FE) service number.

There is no UCL support for this subentry in the global zone.

**FMID**

identifies the function SYSMOD to which this SYSMOD is applicable.

There is no UCL support for this subentry in the global zone.
**FUNCTION**
indicates that this SYSMOD is a function, which introduces a new function into the system.

There is no UCL support for this subentry in the global zone.

**hfs_element**
lists the hierarchical file system element replacements (;++hfs_element statements) in the SYSMOD.

There is no UCL support for this subentry in the global zone.

**HOLERROR**
lists the error hold reason IDs in effect for this SYSMOD.

The actual ++HOLD statement associated with each reason ID (HOLDDATA) is not in the SYSMOD entry. However, if the LIST command specified HOLDDATA and SYSMOD, the listing of the SYSMOD entry includes the HOLDDATA. For more information, see information about listing HOLDDATA with the LIST command in [SMP/E Commands](#).

There is no UCL support for this subentry in the global zone.

**HOLDFIXCAT**
lists the fix category hold reason IDs in effect for this SYSMOD.

There is no UCL support for this subentry in the global zone.

**HOLDSYSTEM**
lists the system hold reason IDs in effect for this SYSMOD.

The actual ++HOLD statement associated with each reason ID (HOLDDATA) is not in the SYSMOD entry. However, if the LIST command specified HOLDDATA and SYSMOD, the listing of the SYSMOD entry includes the HOLDDATA. For more information, see information about listing HOLDDATA with the LIST command in [SMP/E Commands](#).

HOLDSYSTEM subentries show either INT or EXT to indicate the source of the ++HOLD statement.

- **INT** means that the ++HOLD statement was contained in the held SYSMOD. The held SYSMOD can be installed only if the BYPASS operand is specified on the APPLY or ACCEPT command.
- **EXT** means that the ++HOLD statement was obtained from another source, such as SMPHOLD. The held SYSMOD can be installed only if the BYPASS operand is specified on the APPLY or ACCEPT command or if the hold is removed by a ++RELEASE statement.

There is no UCL support for this subentry in the global zone.

**HOLDUSER**
lists the user hold reason IDs in effect for this SYSMOD.

The actual ++HOLD statement associated with each reason ID (HOLDDATA) is not in the SYSMOD entry. However, if the LIST command specified HOLDDATA and SYSMOD, the listing of the SYSMOD entry includes the HOLDDATA. For more information, see information about listing HOLDDATA with the LIST command in [SMP/E Commands](#).

There is no UCL support for this subentry in the global zone.

**JAR**
lists the JAR file replacements (;++JAR) supplied by the SYSMOD.

There is no UCL support for this subentry in the global zone.
SYSMOD entry (global zone)

JARUPD
lists the JAR file updates (++JARUPD) supplied by the SYSMOD.
There is no UCL support for this subentry in the global zone.

JCLIN
indicates that the SYSMOD contained inline JCLIN.
There is no UCL support for this subentry in the global zone.

MAC
lists the macro replacements (++MAC statements) in the SYSMOD.
There is no UCL support for this subentry in the global zone.

MACUPD
lists the macro updates (++MACUPD statements) in the SYSMOD.
There is no UCL support for this subentry in the global zone.

MOD
lists the module replacements (++MOD statements) in the SYSMOD.
There is no UCL support for this subentry in the global zone.

NPRE
lists negative prerequisite SYSMODs (that is, SYSMODs that must not be present in the system at the same time as this SYSMOD).
There is no UCL support for this subentry in the global zone.

PRE
lists prerequisite SYSMODs (that is, SYSMODs that must be present before this SYSMOD can be installed).
There is no UCL support for this subentry in the global zone.

PROGRAM
lists the program element replacements (++PROGRAM statements) in the SYSMOD.
There is no UCL support for this subentry in the global zone.

PTF
indicates that this SYSMOD is a PTF, which provides preventive service fixes.
There is no UCL support for this subentry in the global zone.

RECDATE
specifies the date on which this SYSMOD was received.
There is no UCL support for this subentry in the global zone.

RECTIME
specifies the time at which this SYSMOD was received.
There is no UCL support for this subentry in the global zone.

RENLMOD
indicates that the SYSMOD contained a ++RENAME statement.
The UCL operand is RENLMOD.

REQ
lists requisite SYSMODs (that is, SYSMODs that must be installed concurrent with this SYSMOD).
There is no UCL support for this subentry in the global zone.
**REWORK**
identifies the level of the SYSMOD, which was received again for minor changes.

For SYSMODs supplied by IBM, the REWORK level is *yyyyddd*, where *yyyy* is the year the SYSMOD was reworked and *ddd* is the Julian date.

There is no UCL support for this subentry in the global zone.

**RLMOD**
indicates the load modules renamed by ++RENAME statements in this SYSMOD.

The UCL operand is `RLMOD(name...)`.

**SOURCEID**
lists the character strings assigned to this SYSMOD during RECEIVE. These values might have been specified by the user on the RECEIVE command, included inline on the ++ASSIGN statement, or assigned by SMP/E when processing the contents of an order.

The UCL operand is `SOURCEID(source_id...)`.

- The source ID can be from 1 to 64 characters in length and contain any nonblank character (X'41' through X'FE') except single quotation mark ('), asterisk (*), percent (%), left parenthesis (()), and right parenthesis ()).
- A source ID value cannot span lines.
- A source ID value is case sensitive.

**SRC**
lists the source replacements (++SRC statements) in the SYSMOD.

There is no UCL support for this subentry in the global zone.

**SRCUPD**
lists the source updates (++SRCUPD statements) in the SYSMOD.

There is no UCL support for this subentry in the global zone.

**SREL**
is the system or subsystem release specified on the indicated ++VER MCS in the SYSMOD; for example, SREL VER(001) = Z038. This SREL value matches an SREL value defined in the GLOBALZONE entry that was used when receiving the SYSMOD. These are the systems and subsystems defined by IBM, with their SRELs:

<table>
<thead>
<tr>
<th>System</th>
<th>SREL</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB2</td>
<td>P115</td>
</tr>
<tr>
<td>CICS</td>
<td>C150</td>
</tr>
<tr>
<td>IMS</td>
<td>P115</td>
</tr>
<tr>
<td>MVS</td>
<td>Z038</td>
</tr>
<tr>
<td>NCP</td>
<td>P004</td>
</tr>
</tbody>
</table>

There is no UCL support for this subentry in the global zone.

**SUPING**
lists the SYSMODs superseded by this SYSMOD.

There is no UCL support for this subentry in the global zone.

**SZAP**
lists the module superzaps (++ZAP statements) in the SYSMOD.

There is no UCL support for this subentry in the global zone.
The UCL operand is TLIBPREFIX(prefix).

- The prefix can contain from 1 to 26 alphanumeric characters.
- The prefix must follow standard naming conventions for data sets.

**Note:** If the TLIBPREFIX subentry is deleted from the entry for a SYSMOD packaged in RELFILE format, the SYSMOD cannot be applied or accepted until the TLIBPREFIX subentry is re-created with the current prefix for the SMPTLIB data sets.

**USERMOD**

indicates that this SYSMOD is a USERMOD, which puts a user modification into the system.

There is no UCL support for this subentry in the global zone.

**VERSION**

lists the function SYSMODs that are versioned by this SYSMOD. Versioning indicates that, if there are any elements in common between the SYSMODs listed and this SYSMOD, this SYSMOD’s elements are at a higher functional level, and are therefore the ones that should be installed.

There is no UCL support for this subentry in the global zone.

**XZAP**

lists the module superzaps (++ZAP statements) in the SYSMOD that contain an EXPAND statement (indicating that the module should be expanded before it is updated).

There is no UCL support for this subentry in the global zone.

## LIST Examples

To list all the SYSMOD entries in a global zone, you can use the following commands:

```plaintext
SET BDY(GLOBAL) /* Set to requested zone. */.
LIST SYSMOD /* List all SYSMOD entries. */.
```

To list specific SYSMOD entries in a global zone, you can use these commands:

```plaintext
SET BDY(GLOBAL) /* Set to requested zone. */.
LIST SYSMOD(UZ00001 /* List only these two */
           UZ00002) /* entries. */.
```

To list specific SYSMOD entries in the global zone along with the associated HOLDDATA, you can use these commands:

```plaintext
SET BDY(GLOBAL) /* Set to requested zone. */.
LIST SYSMOD(UZ12345 /* List only these two */
           UZ56789) /* entries */
           HOLDDATA /* plus HOLDDATA for them. */.
```

The format of the LIST output for each SYSMOD entry is the same for all of these commands. The only differences are the number of SYSMOD entries listed and the amount of information presented for each entry (HOLDDATA causes additional information to be listed).
Figure 61. SYSMOD entry: sample LIST output for a global zone (Example 1)
**GLOBAL SYSMOD ENTRIES**

**NAME**

LUS0001 TYPE = USERMOD
STATUS = REC
DATE/TIME REC = 06.100 08:00:00
APPLY ZONE = TGT1 TGT2
ACCEPT ZONE = DLIB1 DLIB2
SREL VER(001) = Z038
FMID VER(001) = JXY2102
PRE VER(001) = UZ00010
MACUPD = MAC02

UZ00010 TYPE = PTF
STATUS = REC
DATE/TIME REC = 06.100 08:00:00
APPLY ZONE = TGT1 TGT2
ACCEPT ZONE = DLIB1 DLIB2
SREL VER(001) = Z038
FMID VER(001) = JXY2102
PRE VER(001) = UZ00008 UZ00007
REQ VER(001) = UZ00040
SUPING VER(001) = AZ99801 AZ99802 UZ00009
MAC = MAC01
MACUPD = MAC02
MOD = MOD01
SRC = SRC01
SRCUPD = SRC01

UZ00011 TYPE = PTF
STATUS = REC
DATE/TIME REC = 06.150 08:00:00
SOURCEID = ABC0706 PUT0704 XAU3380
APPLY ZONE = TGT1 TGT2
ACCEPT ZONE = DLIB1 DLIB2
SREL VER(001) = Z038
FMID VER(001) = JXY2102
PRE VER(001) = UZ00010
MOD = MOD01

---

**Figure 62. SYSMOD entry: sample LIST output for a global zone (Example 2)**
You can use the ADD, REP, and DEL UCL statements to change subentries in a SYSMOD entry. When you use UCLIN to update a SYSMOD entry, remember that if a DEL statement deletes all the existing subentries in the entry, SMP/E deletes the entire entry.

The following examples are provided to help you use the SYSMOD entry.

**Example 1: Changing the SOURCEID of a SYSMOD**

Assume that you received a SYSMOD and assigned it a SOURCEID value of DATALINK, and that you now wish to change the SOURCEID value to PUT0701 so that the SYSMOD will be installed with that service level. The following UCL can be used to change the SOURCEID value:

```
SET BDY(GLOBAL) /* Set to global zone. */.
UCLIN /* */.
REP SYSMOD(UZ00001) /* Specify SYSMOD. */.
  SOURCEID(PUT0701) /* Change SOURCEID value. */.
ENDUCL /* */.
```
Example 2: Indicating that a SYMMD was applied

Assume that you have received some service, accepted it with the BYPASS(APPLYCHECK) operand, and then performed a system generation. That service is now actually in the target libraries, and you would like that recorded in the global zone SYMMD entry. Assume that the name of your target zone is TGT1. The following UCL can be used:

```
SET BDY(GLOBAL) /* Set to global zone. */.
UCLIN /* */.
ADD SYSMOD(UZ00001) /* Specify SYMMD. */.
    APPID(TGT1) /* Applied to this zone. */.
ENDUCL /* */.
```

Note: One UCL statement is required for each SYMMD.
TARGETZONE entry (target zone)

The TARGETZONE entry contains information SMP/E uses to process a specific target zone and the associated target libraries. It is created by UCLIN and must be defined before you can do any other processing for that target zone.

Subentries

These are the subentries for the TARGETZONE entry as they appear in the LIST output:

name

is the name of the target zone. You assign the name when the zone is created.

The name can contain from one to seven alphanumeric characters (A–Z, 0–9) or national characters ($, #, @). The first character must be alphabetic.

OPTIONS

is the name of the OPTIONS entry in the global zone that should be used in processing this target zone. For more information, see “OPTIONS entry (global zone)” on page 292.

The UCL operand is OPTIONS(name).

- The name can contain from one to eight alphanumeric characters.
- This name can be overridden by using the OPTIONS parameter on the SET command. For more information, see SMP/E Commands.
- If no OPTIONS entry name is specified, SMP/E uses a set of default utility values when processing this target zone. For more information, see “OPTIONS entry (global zone)” on page 292.

RELATED

is the name of the distribution zone to which this target zone is related. A target zone is related to the distribution zone used to build the target libraries, such as during system generation.

The UCL operand is RELATED(zone).

- The zone name can contain from one to seven alphanumeric characters.
- Although the entry can be defined without this subentry, you must define the subentry before you can install any SYSMODs in the target libraries.

SREL

lists the system releases to be supported in this target zone.

The UCL operand is SREL(srel...).

- The SREL must contain four alphanumeric characters, usually one alphabetic character followed by three numeric characters. These are the SRELs defined by IBM:

<table>
<thead>
<tr>
<th>System</th>
<th>SREL</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB2</td>
<td>P115</td>
</tr>
<tr>
<td>CICS</td>
<td>C150</td>
</tr>
<tr>
<td>IMS</td>
<td>P115</td>
</tr>
<tr>
<td>MVS</td>
<td>Z038</td>
</tr>
<tr>
<td>NCP</td>
<td>P004</td>
</tr>
</tbody>
</table>
- Although the entry can be defined without this subentry, you must define the subentry before you can install any SYSMODs in the target libraries.

Note: Although you can support multiple products with different SREL values from one target zone, those products are still subject to all other
restrictions related to combining products in one zone. The most common reason for not being able to combine products is common element names. For example, modules or macros with the same name are found in both products, but reside in different libraries.

**TIEDTO**

specifies other target zones that either:

- Supplied modules for load modules controlled by this target zone
- Control load modules that have been link-edited with modules supplied by this target zone

The UCL operand is `TIEDTO(zone...)`.

**Note:** TIEDTO subentries are added automatically during LINK MODULE command processing. However, they are **never** automatically deleted.

**UPGLEVEL**

indicates the highest SMP/E release level that is allowed to make incompatible changes to the target zone. Before making an incompatible change to the target zone, SMP/E will check the UPGLEVEL value for that zone. If the release level of SMP/E is higher than the zone's UPGLEVEL value, SMP/E will not make the incompatible change.

The UPGLEVEL value is in the form `vr.pp`, where `vr` represents the version and release of SMP/E and `pp` represents the PTF level of SMP/E.

There is no UCL support for this subentry. When a zone is created by SMP/E using the UCLIN command or the Administration dialog, SMP/E sets the UPGLEVEL subentry value for that zone to the level of SMP/E used to create the zone. The UPGRADE command is used to change the UPGLEVEL subentry value for a zone.

**XZLINK**

specifies whether APPLY and RESTORE processing in another zone should automatically update load modules in this zone when cross-zone modules previously added to those load modules by the LINK MODULE command are changed.

The UCL operand is `XZLINK(value)`.

- This subentry can contain one of the following values:
  
  **DEFERRED**
  
  Cross-zone load modules controlled by this zone should **not** be automatically updated when modules previously included in them by the LINK MODULE command are updated or deleted.

  To make sure that the modules are synchronized with the cross-zone load modules and that the cross-zone information in SMP/E entries is correct, some combination of the following commands must be run later (depending on how the module changes affect the load modules):
  
  - LINK MODULE command, to include modules in the affected load modules
  - Link-edit (outside of SMP/E), to delete modules from the affected load modules
  - UCLIN command, to update cross-zone subentries as necessary

  **AUTOMATIC**
  
  Cross-zone load modules controlled by this zone should be
TARGETZONE entry (target zone)

automatically updated when modules previously included in them by the LINK MODULE command are updated or deleted.

- If XZLINK is not specified, SMP/E uses the default value of DEFERRED.
- XZLINK does not affect processing of the LINK command.
- The XZLINK(DEFERRED) value is listed only when the TARGETZONE entry contains TIEDTO records.

ZDESC

is a user-written description for this zone.

The UCL operand is ZONEDESCRIPATION(text).

- The zone description can be in single-byte characters (such as English alphanumeric characters) or in double-byte characters (such as Kanji).
- The zone description can contain up to 500 bytes of data, including blanks. (For double-byte data, the 500-byte maximum includes all shift-in and shift-out characters, as well as the double-byte characters.) Extra blanks are deleted. All data beyond column 72 is ignored, including blanks.
- The zone description cannot be only blanks.
- If parentheses are included in the text, they must be in matched pairs.

LIST Examples

To list the TARGETZONE entry for a particular target zone, you can use the following commands:

```
SET BDY(TGT1) /* Set to requested target. */.
LIST TARGETZONE /* List TARGETZONE entry. */.
```

Figure 64 is an example of LIST output for a TARGETZONE entry.

```
TGT1 TZONE ENTRY

NAME

TGT1 TZONE = TGT1
ZDESC = ZONE DESCRIPTION FOR TGT1 ZONE
RELATED = DLIB1
SREL = P115 R020 Z038
OPTIONS = OPITGT1
XZLINK = AUTOMATIC
TIEDTO = CICS1 IMS1
```

Figure 64. TARGETZONE entry: sample LIST output

UCLIN Examples

You can use the ADD, REP, and DEL UCL statements to change subentries in the TARGETZONE entry. When you use UCLIN to update a TARGETZONE entry, remember that if a DEL statement deletes all the existing subentries in the entry, SMP/E deletes the entire entry.

The following examples are provided to help you use the TARGETZONE entry.

Example 1: Defining a TARGETZONE entry

Assume that you are about to build a new set of target libraries and thus wish to define a new target zone named TGT2. The distribution zone that you have done
the system generation from is DLIB1, the OPTIONS entry that will be set up will be called OPTTGT2, and the target zone will contain MVS (Z038) and two additional programs, each with its own SREL value (P010 and R020). The target zone will exist in data set SMPE.SMPCSI.CSI. Use the following UCLIN to define the new zone.

```clist
SET BDY(GLOBAL) /* Set to global zone. */.
UCLIN /* UCLIN for GZONE entry */.
ADD GZONE /* to set up */
  ZONEINDEX( /* index for new zone. */
    (TGT2,SMPE.SMPCSI.CSI,TARGET) /* */
  ) /* */.
ENDUCL /* End global zone update. */.
SET BDY(TGT2) /* Now define new zone. */.
UCLIN /* UCLIN to define it. */.
ADD TARGETZONE(TGT2) /* Identify name. */
  OPTIONS(OPTTGT2) /* OPTIONS entry to use. */
  SREL(Z038, /* SRELs for MVS and */
    P010, /* two other */
    R020) /* programs. */
  RELATED(DLIB1) /* Generated from DLIB. */
/* */.
ENDUCL /* */.
```

Note: Even though the OPTIONS entry has not been set up yet, you can still refer to it in the TARGETZONE entry. Prior to processing this target zone the OPTIONS entry must be created. For examples of setting up the OPTIONS entries, see "OPTIONS entry (global zone)" on page 292.

Example 2: Formatting a zone description

Assume that you enter the following zone description with the first line ending in column 72 and the second line starting in column 1:

```clist
SET BDY(TGT1) /* Set to tgt zone. */.
UCLIN /* UCLIN for TZONE entry */.
ADD TZONE(TGT1) /* to set up. */
  ZDESC( THIS IS THE DESCRIPTION FOR
  THE TGT1 ZONE) /* End of zone description. */.
ENDUCL /* End tgt zone update. */.
```

Because there is no blank between the word ending in column 72 and the next word starting in column 1, SMP/E will run the two together.

Words in a zone description, even words that end in column 72, must be separated by a blank. To format the zone description in this example correctly, you can put a blank at the beginning of the second line:

```clist
SET BDY(TGT1) /* Set to tgt zone. */.
UCLIN /* UCLIN for TZONE entry */.
ADD TZONE(TGT1) /* to set up. */
  ZDESC( THIS IS THE DESCRIPTION FOR
  THE TGT1 ZONE) /* End of zone description. */.
ENDUCL /* End tgt zone update. */.
```

Because there is a blank explicitly coded between the word ending in column 72 and the word starting in column 1, SMP/E will not run the words together.
The UTILITY entry contains information that SMP/E uses when invoking a particular system utility program, such as the load module name of the program and the parameters that should be passed. This information is used only if the UTILITY entry is named in the OPTIONS entry that is in effect. For example, to have SMP/E use certain parameters when it calls a specific link-edit utility, you must do the following:

1. Define a UTILITY entry that names the program and specifies the parameters.
2. Define an OPTIONS entry that specifies that UTILITY entry as the one to use for the link-edit utility.
3. Put that OPTIONS entry into effect, either by specifying it on the SET command or by defining it as the default OPTIONS entry for the zone to be processed.

If the OPTIONS entry does not point to a UTILITY entry for a particular system utility, SMP/E uses default values for that utility. Table 6 lists the default values for the various types of utility programs.

Table 6. Default values for UTILITY entries

<table>
<thead>
<tr>
<th>Utility</th>
<th>NAME (see note 1)</th>
<th>RC</th>
<th>PRINT</th>
<th>PARM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access method</td>
<td>IDCAMS</td>
<td>0</td>
<td>SYSPRINT</td>
<td></td>
</tr>
<tr>
<td>services</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Assembler</td>
<td>ASMA90</td>
<td>4</td>
<td>SYSPRINT</td>
<td>XREF, NOOBJECT, DECK</td>
</tr>
<tr>
<td>Compress</td>
<td>IEBCOPY</td>
<td>0</td>
<td>SYSPRINT</td>
<td></td>
</tr>
<tr>
<td>Copy</td>
<td>IEBCOPY</td>
<td>0</td>
<td>SYSPRINT</td>
<td>SPCLCMOD and CMWA=256K (for program elements and copied load modules)</td>
</tr>
<tr>
<td>Hierarchical file</td>
<td>BPXCOPY</td>
<td>0</td>
<td>SYSPRINT (see note 3)</td>
<td></td>
</tr>
<tr>
<td>system copy</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Link-edit utility</td>
<td>IEWBLINK</td>
<td>8</td>
<td>SYSPRINT</td>
<td>LET, LIST, NCAL, XREF (see note 2)</td>
</tr>
<tr>
<td>Retry after x37</td>
<td>IEBCOPY</td>
<td>0</td>
<td>SYSPRINT</td>
<td></td>
</tr>
<tr>
<td>abends</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Update</td>
<td>IEBUPDTE</td>
<td>0</td>
<td>SYSPRINT</td>
<td>Determined by SMP/E during processing</td>
</tr>
<tr>
<td>Superzap</td>
<td>IMAISPZAP</td>
<td>4</td>
<td>SYSPRINT</td>
<td></td>
</tr>
</tbody>
</table>
### Table 6. Default values for UTILITY entries (continued)

<table>
<thead>
<tr>
<th>Utility</th>
<th>NAME (see note 1)</th>
<th>RC</th>
<th>PRINT</th>
<th>PARM</th>
</tr>
</thead>
</table>

Notes:
1. If you replace a default utility program, the replacement utility program must be compatible with the default utility it replaces, both in the way it processes any control statements and execution parameters generated by SMP/E and in the return codes that it returns to SMP/E.
2. When the load module being link-edited contains a CALLLIBS subentry list, SMP/E does not always use NCAL by default. In this case, SMP/E uses CALL for the link to the actual target library or NCAL for the link to the SMPLTS library. SMP/E always uses NCAL for ACCEPT processing.
3. If SYSTSPRT is specified as the PRINT value, it is ignored and the default of SYSPRINT is used instead.

If a UTILITY entry defines some, but not all, of these subentries, SMP/E uses the default values for the subentries not specified. For more information about OPTIONS entries, see "OPTIONS entry (global zone)" on page 292.

The names of the utilities SMP/E may call include all of SMP/E's default utility program names (BPXCOPY, ASMA90, IEBCOPY, IEBUPDTE, IEWBLINK, IEWL, and IMAZIPAP) and any utility program name specified in an active UTILITY entry. Because SMP/E runs authorized, all the utility programs called by SMP/E must reside in an authorized library. Also, if a particular utility program is to be restricted, the z/OS Security Server must be used to control its execution.

### Subentries

These are the subentries for the UTILITY entry as they appear in the LIST output:

- **name**
  - is the name of the UTILITY entry.
  - The name can contain from 1 to 8 alphanumeric characters.

- **LIST**
  - indicates whether member names should be listed when SMP/E invokes a copy utility to perform compress processing, retry processing, or element installation.

  - The UCL operand is LIST(YES | NO).
    - YES indicates that member names should be listed during copy processing. This is the default.
    - NO indicates that the list of member names should be suppressed during copy processing.
    - The LIST value (including the default value) is ignored when the UTILITY entry is not for a copy utility.

- **NAME**
  - is the name of the load module for the utility program that SMP/E is to call. [Table 6 on page 350] lists the default names used by SMP/E if the OPTIONS entry in effect does not point to a UTILITY entry for a specific system utility.

  - The UCL operand is NAME(prog).
  - The name can contain from 1 to 8 alphanumeric characters. The first character must be alphabetic.
UTILITY entry (global zone)

**PARM**

specifies the parameters to be passed to the utility program. SMP/E may add other parameters to this list. [Table 6 on page 350](#) lists the default parameters used by SMP/E if the OPTIONS entry in effect does not point to a UTILITY entry for a specific system utility.

The UCL operand is **PARM**(string).

- If you specify **PARM**, you must specify all the parameters to be passed. You cannot specify a single value to add or change just one parameter.

- The parameter string is usually a hexadecimal character string with a length of up to 100 characters. This 100-character limit can be exceeded for a binder link-edit step using the OPTION option. For a more detailed explanation, see the GENERATE command chapter in [SMP/E Commands](#). No validity checking is done on the string. If any blanks are specified between the parentheses, they will be deleted by SMP/E during processing.

- If the string contains parentheses, they must be in matched pairs. SMP/E assumes it has reached the end of the specified string when it encounters a closing parenthesis for the opening parenthesis. For example, the following is valid because there are matching parentheses within the string:

  ```
  PARM(LIST,LET,SIZE=(1526K,80K),NCAL)
  ```

  Parentheses within quotation marks are still considered parentheses. The next example is not valid because there are unmatched parentheses. SMP/E would continue scanning after the last parenthesis looking for another “)”.

  ```
  PARM(PRINT(A),SPECIALCHAR('))
  ```

  Likewise, the following example is not valid. SMP/E would stop at the “)” after SPECIALCHAR and give a syntax error on the closing quotation mark.

  ```
  PARM(PRINT(A),SPECIALCHAR'))
  ```

- **Assembler utility**: If no PARM subentry is specified for an assembler utility, SMP/E passes the default PARM values shown for assembler utilities in [Table 6 on page 350](#).

  If you change the PARM subentry, either include DECK in the parameters chosen or ensure that the assembler program produces an object deck.

- **HFS copy utility**: If the UTILITY entry for the HFS copy utility specifies a PARM value, those parameters are passed to the utility in addition to any parameters saved in the hierarchical file system element entry.

- **Link-edit utility**: SMP/E constructs the string of link-edit parameters to be passed to the link-edit utility in the following steps:
  1. If a PARM subentry is specified for a link-edit utility, SMP/E starts the string with the parameters specified in the PARM subentry. Otherwise, SMP/E starts the string with the default parameters LET, LIST, NCAL, and XREF.
  2. If SMP/E finds any link edit parameters in the LMOD entry for the load module being link-edited, SMP/E adds those parameters to the string.
  3. If the load module being link-edited contains a CALLLIBS subentry list, then:
     - SMP/E adds CALL for the link to the actual target library
     - SMP/E adds NCAL for the link to the SMPLTS library

     Any parameter added by this step will override any CALL or NCAL parameters added in previous steps, whether they were explicitly specified or assumed by default.
  4. Finally, if SMP/E determines that the binder is available on the system and SMP/E is processing CSECT deletes, SMP/E adds the STORENX
link-edit parameter to the parameter list. (If you use the linkage-editor instead of the binder, STORENX will cause "INVALID" to appear in the "invocation parameters list" output of the linkage-editor. This has no effect on processing.)

The consolidated parameter list is passed to the link-edit utility.

LET, LIST, and sometimes either CALL or NCAL are ordinarily required for maintenance of IBM operating systems. If you change the PARM subentry, be sure to include LET, LIST, and any other required parameters in the parameters chosen. For more information, see the APPLY command chapter in [SMP/E Commands](SMP/E Commands).

Note: Before using the GENERATE command to create JCL that will update files in a UNIX file system, you may want to add UID(0) to the PARM value so that the JCL created by GENERATE will include UID(0) in the execution parameter string for the link-edit utility. Specify UID(0) if all the following are true:

1. Your UID is not 0 but you are authorized to the BPX.SUPERUSER facility class profile. The UID(0) option, in this case, causes the binder to set an effective UID of 0 for its execution.
2. The binder invoked by the generated JCL is at the proper level to understand the UID option.
3. UID 0 authority is needed for the binder's execution in the JCL created by the GENERATE command in order to update files in a UNIX file system.

If you do specify UID(0) for GENERATE, you must remove it after GENERATE has run, so that it is not used for other SMP/E commands (such as APPLY, which handles setting the effective UID itself prior to invoking the binder).

• **Update utility**: The PARM subentry for an update utility must not specify MOD or NEW.

**PRINT**

Specifies the ddname that is to contain output from the utility (for example, SYSPRINT). [Table 6 on page 350](SMP/E Commands) lists the default ddnames used by SMP/E if the OPTIONS entry in effect does not point to a UTILITY entry for a specific system utility.

The UCL operand is PRINT(ddname).

The value specified can contain from 1 to 8 alphanumeric characters. The first character must be alphabetic.

**Notes:**

1. The ddname specified for PRINT can affect whether you receive print output from this utility. For example, if you specify a DDDEF for a DUMMY data set or a DDDEF for a data set that is sent to a SYSOUT class that suppresses output, you do not receive print output from this utility.
2. If SYSTSPRT is specified as the PRINT value for the HFS copy utility, it is ignored and the default of SYSPRINT is used instead.
3. To exploit utility multi-tasking in SMP/E, ensure that the ddname that is to contain the link edit utility output is defined with either a DDDEF entry that identifies a SYSOUT class or an entry in the SMPPARM GIMDDALC member that identifies a SYSOUT class. SMP/E's default ddname for utility output is SYSPRINT, but can be changed using the PRINT subentry of the LKED UTILITY entry. When multi-tasking, SMP/E will invoke multiple
instances of the link edit utility at the same time, thus decreasing the total
time required to complete an ACCEPT, APPLY, LINK LMODS or RESTORE
command. Multi-tasking of link edit can occur when there are different
target libraries and there are no dependencies on previous and subsequent
link edits. If you do not define the print ddname using either a DDDEF
target or an SMPPARM GIMDDALC member, or if the print ddname
definition identifies something other than SYSOUT class, or if you override
the SYSPRINT DDDEF with a ddname in your JCL, then SMP/E will not
multi-task link edit utility operations.

**RC** specifies the maximum acceptable return code from this utility. If the return
code is higher than this value, SMP/E normally assumes that the requested
processing failed. However, the RETURN CODE value in the LMOD entry
overrides (for that load module) the value in the UTILITY entry. Also, the
success of a link-edit in the SMPLTS library is based on the following:

- Link-edits into the SMPLTS with a return code of 8 or less are considered
  successful regardless of the threshold return code specified in the UTILITY
  entry.
- Link-edits into the SMPLTS with a return code of greater than 8 are
  considered successful or failures based on the threshold (normal processing).
  If the SMPLTS link-edit return code is less than or equal to the threshold
  return code, it is considered successful. If the SMPLTS link-edit return code
  is greater than the threshold return code, it is considered a failure.

Table 6 on page 350 lists the default return codes used by SMP/E if the
OPTIONS entry in effect does not point to a UTILITY entry for a specific
system utility.

The UCL operand is RC(rc).

- The value can be from 0 to 16.
- The installation information for a product (such as the product program
directory) may state the utility return codes you should expect during
SMP/E processing. For example, it may state the expected link-edit return
codes for its load modules during SMP/E processing. The expected return
code may be 4 or 8, because post-SMP/E link-edit work is required (for
example, the load modules may require interface routines or compiler library
routines). Such return codes allow the SYSMODs to be installed, but they
also require you to check the actual link-edit return code in the GIM23903I
or GIM23903W messages in order to determine the actual success of utility
processing.

Before using the default SMP/E return codes (especially for link-edit
processing), check the installation information for the products you plan to
install and determine the appropriate maximum acceptable return codes for
utility processing. You may find that you need to set up more than one
UTILITY entry for a particular utility program in order to accommodate
different maximum return codes for various products. (As a result, you may
need additional OPTIONS entries to point to the appropriate UTILITY
entries.)

**LIST Examples**

To list all the UTILITY entries in a global zone, you can

```bash
SET BDY(GLOBAL) /* Set to requested zone. */.
LIST UTILITY /* List all UTILITY entries. */.
```
UTILITY entry (global zone)

To list specific UTILITY entries in a global zone, you can use these commands:

```
SET BDY(GLOBAL) /* Set to requested zone */.
LIST UTILITY(IEUASM) /* List only these three */.
        IEWL /* entries. */.
        MYX37) /* */.
```

The format of the LIST output for each UTILITY entry is the same for both of these commands. The only difference is the number of UTILITY entries listed.

Figure 65 shows an example of LIST output for UTILITY entries.

---

**Figure 65. UTILITY entry: sample LIST output**

```
GLOBAL UTILITY ENTRIES

NAME
IEUASM PRINT = MYSYSPRT
IEWL PARM = SIZE=(1526K,80K)
MYX37 NAME = USERRUVR
        PARM = TYPE=FAST
        PRINT = X37PRINT
        RC = 4
        LIST = NO
```

---

**UCLIN Examples**

You can use the ADD, REP, and DEL UCL statements to change subentries in a UTILITY entry. When you use UCLIN to update a UTILITY entry, keep these points in mind:

- After the UCLIN changes are done, the UTILITY entry must contain at least one of the following subentries:
  - NAME
  - PARM
  - PRINT
  - RC

  Otherwise, there is not enough information in the entry for SMP/E to use the entry.

- If a DEL statement deletes all the existing subentries in the entry, SMP/E deletes the entire entry.

The following examples are provided to help you use the UTILITY entry.

**Example 1: Changing the DD statement for SYSPRINT output**

Assume that you wish to direct all the SYSPRINT output for assemblies to the MYSYSPRT DD statement rather than to the SYSPRINT DD statement. To do this you need to build a UTILITY entry for the assembler and then specify that UTILITY entry in the OPTIONS entry that will be in effect. Assume that the UTILITY entry to be defined is named IEUASM, and that the OPTIONS entry to be used is MYOPT1. The following UCL will accomplish your objectives:

```
SET BDY(GLOBAL) /* Set to global zone */.
UCLIN /* */.
ADD UTILITY(IEUASM) /* Assembler utility */.
```
UTILITY entry (global zone)

PRINT(MYSYSPRT) /* Alternate SYSPRINT */
/* Other values remain as in SMP/E defaults. */.
REP OPTIONS(MYOPT1) /* Connect to OPTIONS. */
ASM(IEUASM) /* Use IEUASM for ASM. */.
ENDUCL /* */.

Note: Remember, if you specify a DDDEF for a DUMMY data set or a DDDEF for a data set that is sent to a SYSOUT class that suppresses output, you do not receive SYSPRINT output from this utility.

Example 2: Defining link-edit utility parameters
Assume that you wish to always pass the SIZE=(1526K,80K) parameter to the link-edit utility. To do this you need to build a UTILITY entry for the link-edit utility and then specify that UTILITY entry in the OPTIONS entry that will be in effect. Assume that the UTILITY entry to be defined is named IEWL, and that the OPTIONS entry to be used is MYOPT1. The following UCL will accomplish your objectives:

SET BDY(GLOBAL) /* Set to global zone. */.
UCLIN /* */.
ADD UTILITY(IEWL) /* Link-edit utility. */
PARM(LET, /* PARM values. */
SIZE=(1526K,80K), /* */
NCAL) /* */.
/* Other values remain as in SMP/E defaults. */.
REP OPTIONS(MYOPT1) /* Connect to OPTIONS. */
LKED(IEWL) /* Use IEWL for LKED. */.
ENDUCL /* */.

Example 3: Defining a user utility program
Assume that you wish SMP/E to call a user routine, USERRCVR, rather than IEBCOPY, in order to recover from x37 abends. In addition to the program name change, the program must also receive parameter TYPE=FAST. You want to indicate that the output should go to X37PRINT rather than SYSPRINT, and that a return code of 4 or less is acceptable. You also want to suppress the listing of member names during retry processing done by your program. To do this you need to build a UTILITY entry for the program, and then specify the UTILITY entry in the OPTIONS entry that will be in effect. Assume that the UTILITY entry to be defined is named MYX37, and that the OPTIONS entry to be used is MYOPT1. The following UCL will accomplish your objectives:

SET BDY(GLOBAL) /* Set to global zone. */.
UCLIN /* */.
ADD UTILITY(MYX37) /* Retry/recovery program. */
NAME(USERRCVR) /* Program name. */
PARAM(PARM(TYPE=FAST)) /* PARM value. */
PRINT(X37PRINT) /* SYSPRINT ddname. */
RC(4) /* Highest acceptable return code. */
LIST(NO) /* No list of member names. */.
REP OPTIONS(MYOPT1) /* Connect to OPTIONS. */
RETRY(MYX37) /* Use MYX37 for RETRY. */.
ENDUCL /* */.
ZONESET entry (global zone)

The ZONESET entry defines a group of zones to be used to limit the SYSMODs processed by an SMP/E command. For example, you can specify a ZONESET on the ZONESET operand of the REPORT command. This defines which zones SMP/E should check for installed SYSMODs that specify conditional requisites that might be needed for functions in other zones in the ZONESET. A ZONESET may also define a group of zones to be checked or ignored by the REJECT command.

Subentries

These are the subentries for the ZONESET entry as they appear in the LIST output:

name

is the name of the ZONESET. The name can contain from 1 to 8 alphanumeric characters.

To avoid confusion and undesired results, you may want to avoid giving a ZONESET the same name as any of the target or distribution zones defined to the global zone that will contain the ZONESET entry. This is because, on some SMP/E command operands, you can specify zones and ZONESETS. When ZONESETS and zones have the same name, you might not get the results you wanted.

For example, suppose you have a ZONESET named Z001 and a zone named Z001. If you specify Z001 on an SMP/E command operand, SMP/E assumes that you want to use the zones defined in ZONESET Z001 (which might or might not include zone Z001), and not the individual zone Z001.

XZREQCHK

indicates whether this ZONESET should be used when establishing the default zone group for the APPLY, ACCEPT, and RESTORE commands.

The UCL operand is XZREQCHK(YES|NO).

- YES indicates that the ZONESET should be used when establishing the default zone group.
- NO indicates that the ZONESET should not be used when establishing the default zone group. NO is the default.
- The XZREQCHK value (including the default value) is ignored when the XZGROUP operand is specified on the APPLY, ACCEPT, or RESTORE command.

ZONE

lists the target or distribution zones that are to be part of this ZONESET.

The UCL operand is ZONE(zone...).

- Each value can contain from 1 to 7 alphanumeric characters.
- A ZONESET can contain both target and distribution zones.
- All the zones in a ZONESET must be defined in the same global zone as the ZONESET entry.
- The zones cannot be defined in global zones that are in different SMP/CSI data sets. For an example of defining a ZONESET in order to report on zones controlled by different global zones, see the REPORT CROSSZONE command in [SMP/E Commands]

LIST Examples

To list all the ZONESET entries in a global zone, you can use the following commands:
ZONESET entry (global zone)

To list specific ZONESET entries in a global zone, you can use these commands:

```
SET BDY(GLOBAL) /* Set to global zone. */.
LIST ZONESET /* List all ZONESET entries. */.
```

The format of the LIST output for each ZONESET entry is the same for both of these commands. The only difference is the number of ZONESET entries listed.

Figure 66 shows an example of LIST output for ZONESET entries.

---

**Figure 66. ZONESET entry: sample LIST output**

<table>
<thead>
<tr>
<th>ZONESET Entries</th>
<th>ZST2</th>
<th>ZONE =</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>ZONE21</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ZONE22</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ZONE23</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ZONESET Entries</th>
<th>ZST4</th>
<th>ZONE =</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>ZONE41</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ZONE42</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ZONE43</td>
</tr>
</tbody>
</table>

---

**UCLIN Examples**

You can use the ADD, REP, and DEL UCL statements to change subentries in a ZONESET entry. When you use UCLIN to update a ZONESET entry, keep these points in mind:

- After the UCLIN changes are done, the ZONESET entry must contain at least a ZONE subentry. Otherwise, the entry contains so little information that SMP/E cannot use it.
- If a DEL statement deletes all the existing subentries in the entry, SMP/E deletes the entire entry.

**Example: Defining a ZONESET entry**

Assume that you have a system with four target zones: BP111, PROD111, BP999, and PROD999. BP111 and BP999 define two different base control programs. PROD111 and PROD999 are two versions of the same product that must be synchronized with each other and with their base control programs (PROD111 with BP111 and PROD999 with BP999). To keep service for these products at the same level, you can group BP111, PROD111, and PROD999 in one ZONESET (S111) and BP999, PROD999, and PROD111 in a second ZONESET (S999). The following UCL will define the ZONESET entries:

```
SET BDY(GLOBAL) /* Set to global zone. */.
UCLIN /* */.
ADD ZONESET(S111) /* ZONESET S111. */.
  ZONE(BP111, /* Include these target */
  PROD111, /* zones. */
  PROD999) /* */.
ADD ZONESET(S999) /* ZONESET S999. */.
  ZONE(BP999, /* Include these target */
  PROD999, /* zones. */
  PROD111) /* */.
ENDUCL /* */.
```
Chapter 6. SMP/E CSI application programming interface

This chapter documents intended Programming Interfaces that allow the customer to write programs to obtain the services of IBM SMP/E for z/OS, V3R6.

This chapter describes the SMP/E CSI application program interface (GIMAPI), which provides access to data stored and used by SMP/E. This data is stored in a VSAM data set called the CSI (Consolidated Software Inventory). The purpose of GIMAPI is to provide you with read-only access to data maintained by SMP/E in the CSI.

Note: GIMAPI cannot be called from within an SMP/E user exit routine.

Overview of GIMAPI

GIMAPI is a program (load module) provided with IBM SMP/E for z/OS, V3R6 that can be called by a user-written application program to query the contents of the CSI.

GIMAPI uses data structures both to receive the query parameters from the caller and to return the query output to the caller. To request a query, you must set up the data structure containing the appropriate query parameters and call GIMAPI.

The general form of the GIMAPI call is:

\[
\text{GIMAPI}(\text{apicmd, parmptr, outptr, language, rc, cc, msgbuff})
\]

QUERY command parameters

Here is an example of a call to the QUERY command:

\[
\text{GIMAPI('QUERY ', parmptr, outptr, 'language', rc, cc, msgbuff)}
\]

QUERY is a keyword that specifies that GIMAPI is to process a query request. Note that it is necessary to pad the string containing the QUERY command with blanks to bring it to a length of eight characters.

parmptr A 4-byte pointer to an area of storage owned by the calling
program that contains a data structure whose elements are the parameters for the QUERY command. A generic data structures definition can be found in “Data structures for QUERY command” on page 390.

outptr A pointer variable that is to be set by GIMAPI with the address of an area of storage owned by GIMAPI that contains the output from the QUERY command. For the QUERY command, the output is a set of linked lists that both contains and describes the requested data. See section “QUERY command output” on page 387 for more details on the QUERY command output.

language A 3-byte character string specifying the language that QUERY command is to use when issuing messages. The valid values are:

- ENU  US English
- JPN  Japanese

Note: Specifying language as three blanks is also valid and is equivalent to specifying “ENU”.

rc A storage area owned by the calling program representing a 4-byte numeric variable. The value of the variable is set by GIMAPI to the return code of the QUERY command processing. The return code represents the severity of the condition raised by the GIMAPI processing. These codes may be returned:

- 0 Indicates the function terminated successfully. The cc value is also set to zero (0) when the return code is 0.
- 4 Indicates a warning condition was detected during execution of GIMAPI. The cc value given indicates the specific warning condition. When a warning occurs, the GIMAPI command continues processing. Appropriate warning messages are written to the message buffer.
- 8 Indicates an error condition was detected during execution of GIMAPI. The cc value given indicates the specific error condition. When an error occurs, the GIMAPI command terminates after storing the error condition in the message buffer. Other GIMAPI commands may be invoked from the calling program.
- 12 Indicates a severe error condition was detected during execution of GIMAPI. The cc value given indicates the specific error condition. When a severe error occurs, GIMAPI terminates after storing the severe condition in the message buffer. The calling program should not make any more calls to GIMAPI.
- 16 Indicates a terminating error condition was detected during execution of GIMAPI. The cc value given indicates the specific error condition. When a terminating error occurs, GIMAPI terminates after storing the condition in the message buffer. The calling program should not make any more calls to GIMAPI.

cc A storage area owned by the calling program representing a 4-byte numeric variable. The value of the variable is set by GIMAPI to the condition code of the GIMAPI processing. The condition code, also known as a reason code, indicates the specific situation that
occurred during GIMAPI processing. The condition code corresponds to the error message stored for the condition. For example, if error message GIM30400S is to be issued, the return code value is 8 and the condition code is 30400.

More than one error condition may occur during a call to GIMAPI. This is most likely during syntax checking, because GIMAPI checks every parameter for syntax errors. (GIMAPI will stop checking a parameter if it finds an error, so GIMAPI will not report any additional errors that may exist for that parameter.) The return code value set is the one with the highest severity of the raised conditions. If more than one “highest condition” is raised, the first condition of that severity encountered is the value returned as the condition code to the calling program.

GIMAPI sets the cc value to zero (0) for a successful run. The rc value is also zero (0) in this case.

msgbuff A pointer variable that is set by GIMAPI to the head of a linked list of messages issued by the QUERY command processing. The elements of the list are ITEM_LIST structures. See section "Data structures for QUERY command" on page 390 for a description of that structure.

Building the QUERY parameters data structure

The QUERY command parameter list must include a pointer to a data structure (parmptr) that defines the query to be processed. This data structure, in turn, contains pointers to character strings that specify from where data is to be retrieved (that is, which CSI, zones, and entry types) and, optionally, a list of subentries and a filter, which allow you to specify conditions that will be used to determine which entries are retrieved.

The following parameters are part of the structure pointed to by the parmptr parameter of the QUERY command. All the parameters are required except the subentry and filter parameters.

csi Input is accepted in mixed case. A character string that specifies the name of the global CSI to be searched by the QUERY command. The DDNAME used when allocating the data set is SMPCSI.

Input is accepted in mixed case. It is folded to upper case when the command is processed.

csilend A decimal number specifying the size of the character string containing the csi parameter. This value cannot be greater than 44.
A zero (0) value indicates that the calling program has already allocated a global CSI data set as the SMPCSI DDNAME. That allocation is used and the csi parameter is ignored.

zone A character string that specifies the zones from which data is to be retrieved. You may enter one or more specific zone names separated by commas or blanks, or any of these values:

GLOBAL Use the global zone
ALLTZONES Use all target zones
ALLDZONES Use all DLIB zones
* Use all zones defined in the GLOBAL zone index

Except for the asterisk (*), these values can be used together and with specific zone names, provided that all values are separated by commas or blanks. The asterisk (*) must be used alone.

For example, these specifications are valid:

```
GLOBAL, ALLTZONES
ALLTZONES, ALLDZONES
ALLTZONES, MY1DLIB, MY2DLIB
ALLTZONES MY1DLIB, MY2DLIB
```

whereas this is not valid:

```
GLOBAL, *, ALLTZONES
```

A zone can be specified only once in the zone parameter. A specific DLIB zone name along with the ALLDZONES keyword or a specific target zone name along with the ALLTZONES keyword does not cause a message to be issued. However, processing for the duplicated zone names will be done only once.

Spaces can be used freely between values and commas.

Input is accepted in mixed case. It is folded to upper case when the command is processed.

The specific zone name can be a ZONESET name. The command will first check whether the name is a ZONESET. If not, it will check the ZONEINDEX.

**zonelen**
A decimal number specifying the size of the character string containing the zone parameter.

**entry**
A character string that indicates the entry types from the specified zones to be searched. You can specify one or more entries, separated by commas or blanks, or you can specify an asterisk (*) to indicate that all entries are to be searched. If an asterisk is used, no other values may be specified.

An entry can be specified only once in the entry parameter.

Spaces can be used freely between entry names and commas.

Input is accepted in mixed case. It is folded to upper case when the command is processed.

The entry types specified must be valid CSI entry types. Refer to "Valid entry types" on page 365 for a list of the valid types.

**entrylen**
A decimal number specifying the size of the character string containing the entry parameter.

**subentry**
Each CSI entry has a set of subentries associated with it. The subentry parameter is a pointer to a character string used to indicate the subentries for which data is retrieved. Several subentries can be specified separated by commas or blanks, or you can specify an asterisk (*) to indicate that all subentries are to be searched. If an asterisk is used, no other values may be specified.

A subentry can be specified only once in the subentry parameter.

Spaces can be used freely between subentry names and commas.

Input is accepted in mixed case. It is folded to upper case when the command is processed.
**Note:** The *entry* parameter allows multiple entries to be specified. It may be that a subentry to be returned does not apply to all the specified entries. In this case, the subentry is simply ignored for a particular entry to which it does not apply. Refer to “Valid subentry types” on page 366 for more details.

**subentrylen** A decimal number specifying the size of the character string containing the *subentry* parameter.

**filter** A character string that specifies the set of conditions with which to limit the set of entries being retrieved. A condition is in the form

```
subentry operator 'value'
```

For example, FMID = 'HP10230' or INSTALLDATE >= '07203'. Input is accepted in mixed case. The subentry types are folded to upper case when the command is processed. The fixed values are not changed.

*Filter* is an optional parameter. When no filtering is desired, set the length field (filterlen) to zero (0).

Refer to “Filter parameter syntax” for a detailed description of the syntax of the *filter* parameter.

**filterlen** A decimal number specifying the size of the character string containing the *filter* parameter.

### Filter parameter syntax

The *filter* parameter is used to limit the entries returned for a particular query. You specify a filter by listing a set of conditions. These conditions specify a comparison between two values; a subentry value in the CSI is compared with a fixed value specified in the condition.

Multiple filter conditions may be specified using the operators, & (logical AND) and | (logical OR). Parentheses may be used to group search conditions to ensure a desired evaluation sequence for the filter. The parentheses, when used, may be preceded by one or more blanks and also may be followed by one or more blanks.

When a subentry name represents a list of values, such as PRE, which is a list of all the prerequisite SYSMODs, each value in the list from the CSI is compared to the fixed value. If any match, the condition is evaluated as true.

These subentry types may be specified in the *filter* parameter only with the = or != operators and a null value (for example, ASMIN=" or LEPARM!="): 

- ASMIN
- HFSPARM
- HOLDDATA
- LECNTL
- LEPARM
- LMODALIAS
- LMODSYMLINK
- ORDERSERVER
- UTILPARM
- ZDESC
The fixed value must be enclosed within single quotation marks. Any set of characters may be used in the fixed value. If the fixed value is shorter than the subentry value, GIMAPI left-justifies the fixed value and pads it with blanks before doing the comparison.

Any apostrophes specified as part of a DESCRIPTION, VENDOR, or URL subentry (excluding delimiters) must be doubled. Double apostrophes count as two characters in the filter length.

Blanks may be freely used to separate the subentry, operator, and fixed value.

The values of some subentry types are lists of composite values. An example of this is the CIFREQ parameter whose value is `causer, requisite`. The fixed value for conditions containing these types of subentries must match the format of the output provided by the QUERY command. For example:

```
CIFREQ='PTF0001,PTF0002'
```

Refer to "Valid subentry types" on page 366 for the format of each composite subentry type.

A null value for a subentry is specified by using two single quotation marks with no blanks between them (""). This can be used to find all entries whose value is not blank, for example, `PATH!=""`.

The operators that can be used to compare a subentry value to a fixed value, or to join a set of conditions together, are:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>Equal to</td>
</tr>
<tr>
<td>!=</td>
<td>Not equal to</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal to</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal to</td>
</tr>
<tr>
<td>l</td>
<td>Logical OR, at least one of two conditions are true</td>
</tr>
<tr>
<td>&amp;</td>
<td>Logical AND, both conditions must be true</td>
</tr>
</tbody>
</table>

Subentry names that may appear to the left of operators are the subentries defined in "Valid subentry types" on page 366. Potential values for each subentry can be found in Chapter 5, "SMP/E data set entries," on page 171. When a subentry is an indicator, such as PROTECT for the DDDEF entry, its value is either YES or NO.

A requested subentry may not have a value for a particular entry. This value is considered null. This may occur for one of the following reasons:
- The subentry has not been set for the entry.
- The subentry does not apply to that entry.

An example of the latter case is a request for the SYSLIB subentry with the SYSMOD entry type. SYSMOD entries do not contain SYSLIB subentries, causing a request for such a value to always return a null value.

When the not equal sign is used with some value within a filter, for example `SYSLIB!="MACLIB"`, the case where no data is found for the subentry is considered
a match, because “MACLIB” does not equal a null value. Therefore, if this filter is
used when requesting the SYSMOD entry type, all the SYSMODs in the selected
zones are returned. To exclude entries having no value, write the filter as
(SYSLIB!=’MACLIB’ & SYSLIB!=’’).

The >, >=, < and <= operators can be used only with the subentry types that have
date, time, or return code values, which are:

- HOLDDATE
- INSTALLDATE
- INSTALLTIME
- RC
- RECEIVEDATE
- RECTIME
- RESDATE
- RESTIME
- UCLDATE
- UCLTIME

The >, >=, < and <= operators have no meaning when applied to a subentry that has a null value. GIMAPI will always resolve such comparisons to “false”.

A date fixed value must use the format ‘yyddd’. It must be five numeric characters.
For example, specify ‘01365’ to represent “December 31, 2001”.

A time fixed value must use the format ‘hh:mm:ss’, where hh is a two-digit decimal number in the range 00 through 23, and mm and ss are both two-digit decimal numbers in the range 00 through 59. (That is, in 24-hour clock format.)

The filter parameter is made up of a combination of these parts:

- Subentry name
- Operator
- Fixed value
- Conjunction (& or |)
- Left parenthesis
- Right parenthesis

The following list indicates for each part, what other part or parts can follow it.
Any other sequence of parts is considered a syntax error.

<table>
<thead>
<tr>
<th>Filter part</th>
<th>Can be followed by</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subentry</td>
<td>Operator</td>
</tr>
<tr>
<td>Operator</td>
<td>Fixed value</td>
</tr>
<tr>
<td>Fixed value</td>
<td>Conjunction, Right parenthesis, End of filter</td>
</tr>
<tr>
<td>Conjunction</td>
<td>Subentry Left parenthesis</td>
</tr>
<tr>
<td>Left parenthesis</td>
<td>Subentry, Left parenthesis</td>
</tr>
<tr>
<td>Right parenthesis</td>
<td>Conjunction, Right parenthesis, End of filter</td>
</tr>
</tbody>
</table>

Valid entry types

Table 7 on page 366 lists the values that are valid as entry parameters and the zones to which they are applicable.
Table 7. Valid entry values

<table>
<thead>
<tr>
<th>Entry type</th>
<th>Global zone</th>
<th>DLIB zone</th>
<th>Target zone</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASSEM</td>
<td>Valid</td>
<td>Valid</td>
<td></td>
</tr>
<tr>
<td>Data element¹</td>
<td>Valid</td>
<td>Valid</td>
<td></td>
</tr>
<tr>
<td>DDDEF</td>
<td>Valid</td>
<td>Valid</td>
<td>Valid</td>
</tr>
<tr>
<td>DLIB</td>
<td>Valid</td>
<td>Valid</td>
<td></td>
</tr>
<tr>
<td>DLIBZONE</td>
<td></td>
<td>Valid</td>
<td></td>
</tr>
<tr>
<td>ELEMENT²</td>
<td>Valid</td>
<td>Valid</td>
<td></td>
</tr>
<tr>
<td>FEATURE</td>
<td></td>
<td>Valid</td>
<td></td>
</tr>
<tr>
<td>FMIDSET</td>
<td>Valid</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GLOBALZONE</td>
<td></td>
<td>Valid</td>
<td></td>
</tr>
<tr>
<td>Hierarchical file system element³</td>
<td>Valid</td>
<td>Valid</td>
<td></td>
</tr>
<tr>
<td>HOLDDATA</td>
<td>Valid</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JAR</td>
<td>Valid</td>
<td>Valid</td>
<td></td>
</tr>
<tr>
<td>LMOD</td>
<td>Valid</td>
<td>Valid</td>
<td></td>
</tr>
<tr>
<td>MAC</td>
<td>Valid</td>
<td>Valid</td>
<td></td>
</tr>
<tr>
<td>MOD</td>
<td>Valid</td>
<td>Valid</td>
<td></td>
</tr>
<tr>
<td>OPTIONS</td>
<td>Valid</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ORDER</td>
<td>Valid</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRODUCT</td>
<td>Valid</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PROGRAM</td>
<td>Valid</td>
<td>Valid</td>
<td></td>
</tr>
<tr>
<td>SRC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYSMOD</td>
<td>Valid</td>
<td>Valid</td>
<td></td>
</tr>
<tr>
<td>TARGETZONE</td>
<td></td>
<td></td>
<td>Valid</td>
</tr>
<tr>
<td>UTILITY</td>
<td>Valid</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ZONESET</td>
<td>Valid</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Notes:

1. The data element entry type has many possible values. Refer to "Data element entry (distribution and target zone)" on page 191 in Chapter 5, “SMP/E data set entries,” on page 171 for the complete list. Any one of these values can be supplied as part of the entry parameter.

2. ELEMENT is a pseudo-entry value indicating that JAR, MOD, MAC, PROGRAM, SRC, and all data element and hierarchical file system element entries are returned.

3. The hierarchical file system element entry type has many possible values. Refer to "Hierarchical File System Element Entry (Distribution and Target Zone)” in Chapter 5 for the complete list. Any one of these values can be supplied as part of the entry parameter.

When you specify an entry that is not applicable to a particular zone, the result is simply that no data is returned. This is not an error condition.

Valid subentry types

For each entry type there is a set of valid subentry types that may be specified as part of the subentry and filter parameters. The following sections contain a table for
Each entry. Each table lists the valid subentries for that entry. The subentry types generally correspond to the types used for UCLIN processing. However, some additional types are used for GIMAPI processing (for example, the SMODTYPE subentry of the SYSMOD entry type). This specifies what type the SYSMOD is and is used instead of having a separate subentry for each SYSMOD type.

**Notes:**
1. ENAME and zone name are automatically returned for each instance of an entry as part of the ENTRY structure so they do not have to be requested on the subentry parameter. ENAME can be used on the filter parameter.

   For a PRODUCT entry, ENAME has no meaning and is always blank. PRODID and VRM values are automatically returned instead of ENAME for PRODUCT entries. PRODID and VRM can be used on the filter parameter.

2. A maximum length of (*) indicates that a linked list of that subentry is returned to the caller. Details of the return structures passed back to the caller can be found in "Data structures for QUERY command" on page 390.

### ASSEM

**Table 8. Valid subentries for the ASSEM entry**

<table>
<thead>
<tr>
<th>Subentry name</th>
<th>Maximum length (decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASMIN</td>
<td>*</td>
<td>A linked list of records containing the assembler statements that were saved for a module during JCLIN processing. Each record may be up to 80 bytes in length.</td>
</tr>
<tr>
<td>ENAME</td>
<td>8</td>
<td>Name of the entry.</td>
</tr>
<tr>
<td>LASTUPD</td>
<td>7</td>
<td>Cause of last change to assembler entry.</td>
</tr>
<tr>
<td>LASTUPDTYPE</td>
<td>8</td>
<td>Indicates how the entry was last changed.</td>
</tr>
</tbody>
</table>

### Data element

**Table 9. Valid subentries for the data element entries**

<table>
<thead>
<tr>
<th>Subentry name</th>
<th>Maximum length (decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALIAS</td>
<td>*</td>
<td>A linked list containing alias names associated with the data element. Each alias name may be up to 8 bytes in length.</td>
</tr>
<tr>
<td>DISTLIB</td>
<td>8</td>
<td>Identifies the ddnname of the distribution library for the data element</td>
</tr>
<tr>
<td>ENAME</td>
<td>8</td>
<td>Name of the entry.</td>
</tr>
<tr>
<td>FMID</td>
<td>7</td>
<td>Specifies the functional owner of the data element.</td>
</tr>
<tr>
<td>LASTUPD</td>
<td>7</td>
<td>Cause of last change to this data element entry.</td>
</tr>
<tr>
<td>LASTUPDTYPE</td>
<td>3</td>
<td>Indicates how the entry was last changed.</td>
</tr>
<tr>
<td>RMID</td>
<td>7</td>
<td>Identifies the last SYSMOD that replaced this data element.</td>
</tr>
<tr>
<td>SYSLIB</td>
<td>8</td>
<td>Identifies the ddnname of the target library for the data element.</td>
</tr>
</tbody>
</table>
**Table 10. Valid subentries for the DDDEF entry**

<table>
<thead>
<tr>
<th>Subentry name</th>
<th>Maximum length (decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONCAT</td>
<td>*</td>
<td>A linked list containing the names of the DDDEF entries existing in the same zone that should be concatenated during SMP/E processing. Each entry name may be up to 8 bytes in length.</td>
</tr>
<tr>
<td>DATACLAS</td>
<td>8</td>
<td>Specifies the name of the data class to be used for allocating a new data set managed by SMS.</td>
</tr>
<tr>
<td>DATASET</td>
<td>44</td>
<td>Specifies the name of the data set to be allocated.</td>
</tr>
<tr>
<td>DIR</td>
<td>4</td>
<td>Specifies the number of directory blocks to allocate.</td>
</tr>
<tr>
<td>DISP</td>
<td>7</td>
<td>Specifies the final disposition of the data set. • CATALOG • DELETE • KEEP</td>
</tr>
<tr>
<td>DSNTYPE</td>
<td>7</td>
<td>Specifies the type of partitioned data set to be created. • LIBRARY • PDS</td>
</tr>
<tr>
<td>DSPREFIX</td>
<td>26</td>
<td>Specifies the data set prefix to be used to construct the full data set name for SMPTLIB data sets. GLOBAL zone only.</td>
</tr>
<tr>
<td>ENAME</td>
<td>8</td>
<td>Name of the entry.</td>
</tr>
<tr>
<td>INITDISP</td>
<td>3</td>
<td>Specifies the initial disposition of the data set. • MOD • NEW • OLD • SHR</td>
</tr>
<tr>
<td>MGMTCLAS</td>
<td>8</td>
<td>Specifies the name of the management class to be used for allocating a new data set managed under SMS.</td>
</tr>
<tr>
<td>PATH</td>
<td>255</td>
<td>Identifies the name of the path to be allocated in a UNIX file system.</td>
</tr>
<tr>
<td>PROTECT</td>
<td>3</td>
<td>Indicates whether RACF® PROTECT option should be used when a new data set is allocated. (YES or NO).</td>
</tr>
<tr>
<td>SPACE</td>
<td>9</td>
<td>Specifies the primary and secondary space allocation in the format mnnn,mnnn.</td>
</tr>
<tr>
<td>STORCLAS</td>
<td>8</td>
<td>Specifies the name of a storage class used for allocating a new data set managed by SMS.</td>
</tr>
<tr>
<td>SYSOUT</td>
<td>1</td>
<td>Specifies the output class for SYSOUT data sets.</td>
</tr>
<tr>
<td>UNIT</td>
<td>8</td>
<td>Specifies the UNIT type the data set resides on if it is not cataloged.</td>
</tr>
<tr>
<td>UNITS</td>
<td>12</td>
<td>Specifies the space units to be used in allocating the data set. • BLK(size) • CYL • TRK</td>
</tr>
<tr>
<td>VOLUME</td>
<td>*</td>
<td>A linked list containing volume serial numbers of the device the data set resides on if it is not cataloged. Each volume serial number may be up to 6 bytes in length.</td>
</tr>
<tr>
<td>WAITFORDSN</td>
<td>3</td>
<td>Indicates whether SMP/E should wait for the data set to be allocated if the volume is not mounted or if the data set is already in use. (YES or NO).</td>
</tr>
</tbody>
</table>
## DLIB

**Table 11. Valid subentries for the DLIB entry**

<table>
<thead>
<tr>
<th>Subentry name</th>
<th>Maximum length (decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENAME</td>
<td>8</td>
<td>Name of the entry.</td>
</tr>
<tr>
<td>LASTUPD</td>
<td>7</td>
<td>Cause of last change to DLIB entry.</td>
</tr>
<tr>
<td>LASTUPDTYPE</td>
<td>3</td>
<td>Indicates how the entry was last changed.</td>
</tr>
<tr>
<td>SYSLIB</td>
<td>*</td>
<td>A linked list containing the ddnames of the target libraries into which the distribution library should be copied. Each ddname may be up to 8 bytes in length.</td>
</tr>
</tbody>
</table>

## DLIBZONE

**Table 12. Valid subentries for the DLIBZONE entry**

<table>
<thead>
<tr>
<th>Subentry name</th>
<th>Maximum length (decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCJCLIN</td>
<td>3</td>
<td>Indicates whether JCLIN is to be saved in the distribution zone whenever a SYSMOD containing inline JCLIN is accepted. (YES or NO).</td>
</tr>
<tr>
<td>ENAME</td>
<td>8</td>
<td>Name of the entry.</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>8</td>
<td>The name of the OPTIONS entry in the global zone that should be used when processing this distribution zone.</td>
</tr>
<tr>
<td>RELATED</td>
<td>7</td>
<td>The name of the target zone to which this distribution zone is related.</td>
</tr>
<tr>
<td>SREL</td>
<td>*</td>
<td>A linked list containing the names of the system releases supported within this distribution zone. Each name may be up to four bytes in length.</td>
</tr>
<tr>
<td>UPGLEVEL</td>
<td>8</td>
<td>Indicates the highest SMP/E release level that is allowed to make incompatible changes to the zone. If SMP/E attempts to make an incompatible change to a zone and the release level of SMP/E is higher than the UPGRADE level in that zone, then the incompatible change will not be made. This value is in the form ( vr.pp ), where ( vr ) represents the version and release of SMP/E and ( pp ) represents the PTF level of SMP/E.</td>
</tr>
<tr>
<td>ZDESC</td>
<td>500</td>
<td>A user-written description for this zone.</td>
</tr>
</tbody>
</table>

## FEATURE

**Table 13. Valid subentries for the FEATURE entry**

<table>
<thead>
<tr>
<th>Subentry name</th>
<th>Maximum length (decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DESCRIPTION</td>
<td>64</td>
<td>Describes the software feature in readable text.</td>
</tr>
<tr>
<td>ENAME</td>
<td>8</td>
<td>Name of the entry.</td>
</tr>
<tr>
<td>FMID</td>
<td>*</td>
<td>A linked list containing 7-byte names of function SYSMODS that make up the feature.</td>
</tr>
<tr>
<td>PRODUCT</td>
<td>17</td>
<td>The product identifier and version, release, and modification level (( prodid,vr.rr.mm )) of the feature's associated product.</td>
</tr>
<tr>
<td>RECDATE</td>
<td>5</td>
<td>Specifies the date on which this feature was received.</td>
</tr>
<tr>
<td>RECTIME</td>
<td>8</td>
<td>Specifies the time at which the ++FEATURE MCS was received.</td>
</tr>
<tr>
<td>REWORK</td>
<td>8</td>
<td>Specifies the level of this FEATURE entry, which was received again for minor changes.</td>
</tr>
</tbody>
</table>
Table 13. Valid subentries for the FEATURE entry (continued)

<table>
<thead>
<tr>
<th>Subentry name</th>
<th>Maximum length (decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UCLDATE</td>
<td>5</td>
<td>Specifies the date on which this FEATURE entry was last modified through UCLIN.</td>
</tr>
<tr>
<td>UCLTIME</td>
<td>8</td>
<td>Specifies the time at which this FEATURE entry was last modified through UCLIN.</td>
</tr>
</tbody>
</table>

Table 14. Valid subentries for the FMIDSET entry

<table>
<thead>
<tr>
<th>Subentry name</th>
<th>Maximum length (decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENAME</td>
<td>8</td>
<td>Name of the entry.</td>
</tr>
<tr>
<td>FMID</td>
<td>*</td>
<td>A linked list containing 7-byte names of the function SYSMODS (that is, FMIDs) that are to be part of this FMIDSET.</td>
</tr>
</tbody>
</table>

Table 15. Valid subentries for the GLOBALZONE entry

<table>
<thead>
<tr>
<th>Subentry name</th>
<th>Maximum length (decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENAME</td>
<td>8</td>
<td>Name of the entry.</td>
</tr>
<tr>
<td>FMID</td>
<td>*</td>
<td>A linked list containing 7-byte names of the function SYSMODS for which SMP/E is to receive service.</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>8</td>
<td>The name of the OPTIONS entry in the global zone that should be used when processing this global zone.</td>
</tr>
<tr>
<td>SREL</td>
<td>*</td>
<td>A linked list containing the names of the system releases supported within this global zone. Each name may be up to four bytes in length.</td>
</tr>
<tr>
<td>UPGLEVEL</td>
<td>8</td>
<td>Indicates the highest SMP/E release level that is allowed to make incompatible changes to the zone. If SMP/E attempts to make an incompatible change to a zone and the release level of SMP/E is higher than the UPGRADE level in that zone, then the incompatible change will not be made. This value is in the form vr.pp, where vr represents the version and release of SMP/E and pp represents the PTF level of SMP/E.</td>
</tr>
<tr>
<td>ZDESC</td>
<td>500</td>
<td>A user-written description for this zone.</td>
</tr>
<tr>
<td>ZONEINDEX</td>
<td>*</td>
<td>A linked list of entries containing the names, data set names, and types of all the target zones and distribution zones associated with this global zone. The format of each entry is name,dsn,type. Each entry in the list may be up to 59 bytes in length.</td>
</tr>
</tbody>
</table>

Hierarchical file system elements

Table 16. Valid subentries for hierarchical file system element entries

<table>
<thead>
<tr>
<th>Subentry name</th>
<th>Maximum length (decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISTLIB</td>
<td>8</td>
<td>Specifies the DDNAME of the distribution library for the hierarchical file system element.</td>
</tr>
<tr>
<td>ENAME</td>
<td>8</td>
<td>Name of the entry.</td>
</tr>
<tr>
<td>FMID</td>
<td>7</td>
<td>Specifies the functional owner of the hierarchical file system element.</td>
</tr>
</tbody>
</table>
### Table 16. Valid subentries for hierarchical file system element entries (continued)

<table>
<thead>
<tr>
<th>Subentry name</th>
<th>Maximum length (decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HFSPARM</td>
<td>300</td>
<td>Specifies a character string that is to be passed to the hierarchical file system copy utility as an execution-time parameter.</td>
</tr>
<tr>
<td>INSTMODE</td>
<td>6</td>
<td>Indicates the installation mode to be used when the HFS copy utility is invoked to install the element into a UNIX system.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- BINARY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- TEXT</td>
</tr>
<tr>
<td>LASTUPD</td>
<td>7</td>
<td>Cause of last change to this hierarchical file system element entry.</td>
</tr>
<tr>
<td>LASTUPDTYPE</td>
<td>3</td>
<td>Indicates how the entry was last changed.</td>
</tr>
<tr>
<td>LINK</td>
<td>*</td>
<td>A linked list containing the alternate names by which this element can be known in a UNIX file system. Each alternate name may be up to 1023 bytes in length.</td>
</tr>
<tr>
<td>RMID</td>
<td>7</td>
<td>Identifies the last SYSMOD that replaced this hierarchical file system element.</td>
</tr>
<tr>
<td>SHSCRIPT</td>
<td>17</td>
<td>Specifies a shell script, along with the point in processing in which SMP/E passes control to the shell script. The format is script_name,PREE,POST, where PRE and POST are optional.</td>
</tr>
<tr>
<td>SYMLINK</td>
<td>*</td>
<td>A linked list containing the names of symbolic links associated with a hierarchical file system element. Each symbolic link may be up to 1023 bytes in length.</td>
</tr>
<tr>
<td>SYMPATH</td>
<td>*</td>
<td>A linked list containing the path values for symbolic links associated with a hierarchical file system element. Each path value may be up to 1023 bytes in length.</td>
</tr>
<tr>
<td>SYSLIB</td>
<td>8</td>
<td>Specifies the ddname of the target library within a UNIX file system for the element.</td>
</tr>
</tbody>
</table>

### Table 17. Valid subentries for the HOLDDATA entry

<table>
<thead>
<tr>
<th>Subentry name</th>
<th>Maximum length (decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENAME</td>
<td>7</td>
<td>Name of the entry. This is the name of the held SYSMOD.</td>
</tr>
<tr>
<td>HOLDCLASS</td>
<td>7</td>
<td>A 1- to 7-character string indicating an alternative reason to release an exception SYSMOD for processing.</td>
</tr>
<tr>
<td>HOLDDATA</td>
<td>*</td>
<td>A linked list containing records that contain the entire HOLDDATA entry. Each record may be up to 80 bytes in length.</td>
</tr>
<tr>
<td>HOLDDATE</td>
<td>5</td>
<td>Specifies the date that the ++HOLD statement was generated.</td>
</tr>
<tr>
<td>HOLDFIXCAT</td>
<td>*</td>
<td>A linked list that contains Fix Category values. Each value can be up to 64 characters in length.</td>
</tr>
<tr>
<td>HOLDFMID</td>
<td>7</td>
<td>Specifies the FMID to which the held SYSMOD is applicable.</td>
</tr>
<tr>
<td>HOLDREASON</td>
<td>7</td>
<td>A 1- to 7-character string used to help you identify the reason why the SYSMOD has been put into exception SYSMOD status.</td>
</tr>
<tr>
<td>HOLDRESOLVER</td>
<td>7</td>
<td>Specifies the SYSMOD that resolves the exception condition, if known.</td>
</tr>
<tr>
<td>HOLDTYPE</td>
<td>9</td>
<td>Specifies the hold category into which the SYSMOD has been put. ERROR, FIXCAT, SYSTEM, or USER are the only valid values.</td>
</tr>
</tbody>
</table>

Refer to the "++HOLD MCS" section for specific details for each of the subentries listed in the HOLDDATA entry.
### Table 18. Valid subentries for the JAR entry

<table>
<thead>
<tr>
<th>Subentry name</th>
<th>Maximum length (decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISTLIB</td>
<td>8</td>
<td>Specifies the ddname of the distribution library for the JAR element.</td>
</tr>
<tr>
<td>ENAME</td>
<td>8</td>
<td>Name of the entry.</td>
</tr>
<tr>
<td>FMID</td>
<td>7</td>
<td>Specifies the functional owner of the JAR element.</td>
</tr>
<tr>
<td>HFSPARM</td>
<td>300</td>
<td>Specifies a character string that is to be passed to the hierarchical file system copy utility as an execution-time parameter.</td>
</tr>
<tr>
<td>JARPARM</td>
<td>300</td>
<td>Specifies a character string that is to be passed to the jar command when updating the JAR file.</td>
</tr>
<tr>
<td>LASTUPD</td>
<td>7</td>
<td>Cause of last change to this JAR element entry.</td>
</tr>
<tr>
<td>LASTUPDTYPE</td>
<td>3</td>
<td>Indicates how the entry was last changed.</td>
</tr>
<tr>
<td>LINK</td>
<td>*</td>
<td>A linked list containing alternate names by which this element can be known in a UNIX file system. Each alternate name may be up to 1023 bytes in length.</td>
</tr>
<tr>
<td>RMID</td>
<td>7</td>
<td>Identifies the last SYSMOD that replaced this JAR element.</td>
</tr>
<tr>
<td>SHSCRIPT</td>
<td>17</td>
<td>Specifies a shell script, along with the point in processing in which SMP/E passes control to the shell script. The format is script_name,PRE,POST where PRE and POST are optional.</td>
</tr>
<tr>
<td>SYMLINK</td>
<td>*</td>
<td>A linked list containing the names of symbolic links associated with this element. Each symbolic link may be up to 1023 bytes in length.</td>
</tr>
<tr>
<td>SYMPATH</td>
<td>*</td>
<td>A linked list containing the path values for symbolic links associated with this element. Each path value may be up to 1023 bytes in length.</td>
</tr>
<tr>
<td>SYSLIB</td>
<td>*</td>
<td>Specifies the ddname of the target library within a UNIX file system for the element.</td>
</tr>
<tr>
<td>UMID</td>
<td>*</td>
<td>A linked list containing 7-byte names of all SYSMODs that have updated this JAR file since it was last replaced.</td>
</tr>
</tbody>
</table>

### Table 19. Valid subentries for the LMOD entry

<table>
<thead>
<tr>
<th>Subentry name</th>
<th>Maximum length (decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALLLIBS</td>
<td>*</td>
<td>A linked list containing the names of the DDDEF entries, existing in the same zone, that compose the SYSLIB allocation to be used when the load module is link-edited. Each entry name may be up to 8 bytes in length.</td>
</tr>
<tr>
<td>COPIED</td>
<td>3</td>
<td>A special SMP/E indicator meaning that the load module was copied during system generation, and that there is a one-to-one correspondence between the distribution library module (the MOD entry) and the target system load module (the LMOD entry) (YES or NO).</td>
</tr>
<tr>
<td>ENAME</td>
<td>8</td>
<td>Name of the entry.</td>
</tr>
<tr>
<td>LASTUPD</td>
<td>7</td>
<td>Cause of last change to this LMOD entry.</td>
</tr>
<tr>
<td>LASTUPDTYPE</td>
<td>3</td>
<td>Indicates how the entry was last changed.</td>
</tr>
<tr>
<td>LEPARM</td>
<td>*</td>
<td>A character string containing the link-edit attributes that must be used when this load module is link-edited. This is not a linked list.</td>
</tr>
</tbody>
</table>
### Table 19. Valid subentries for the LMOD entry (continued)

<table>
<thead>
<tr>
<th>Subentry name</th>
<th>Maximum length (decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LECNTL</td>
<td>*</td>
<td>A linked list containing records of link-edit control cards necessary to relink-edit this load module. Each record may be up to 80 bytes in length.</td>
</tr>
<tr>
<td>LMODALIAS</td>
<td>*</td>
<td>A linked list containing the names of the aliases associated with this LMOD. Each alias name may be up to 1023 bytes in length.</td>
</tr>
<tr>
<td>LMODSYMLINK</td>
<td>*</td>
<td>A linked list containing the names of symbolic links associated with a load module. The values are the values specified on ALIAS (SYMLINK,symlink_value) link-edit control statements. Each symbolic link may be up to 1023 bytes in length.</td>
</tr>
<tr>
<td>MODDEL</td>
<td>*</td>
<td>A linked list containing the names of modules that were once part of this load module, but have been deleted. Each module name may be up to 8 bytes in length.</td>
</tr>
<tr>
<td>RC</td>
<td>2</td>
<td>Highest acceptable link edit return code for this load module.</td>
</tr>
<tr>
<td>SIDEDECKLIB</td>
<td>8</td>
<td>Specifies the ddname of the library where the load module’s definition side deck resides.</td>
</tr>
<tr>
<td>SYSLIB</td>
<td>*</td>
<td>A linked list containing the ddnames of the target system libraries in which this load module resides. Each ddname may be up to 8 bytes in length.</td>
</tr>
<tr>
<td>UTIN</td>
<td>*</td>
<td>A linked list containing utility input (UTIN) subentries, each of which contains the filename of the input to be included during a link-edit operation, and the library ddname where the file resides. The format is filename,ddname. Each UTIN subentry may be up to 1032 bytes in length. (The filename may be up to 1023 characters in length. The ddname may be up to 8 characters in length. And then there is the delimiting comma.) <strong>Note:</strong> It should be noted that a comma is a valid character within the filename value. The comma is also the delimiter separating the filename from the ddname. Therefore, the utility input data should be scanned from the last byte to the first byte in order to accurately find the comma that separates the ddname from the filename.</td>
</tr>
<tr>
<td>XZMOD</td>
<td>*</td>
<td>A linked list containing the XZMOD subentries containing the names of the modules that were added to the load module by the LINK MODULE command. The name of the zone supplying each module is also indicated. The format is module,zone. Each XZMOD subentry may be up to 16 bytes in length.</td>
</tr>
<tr>
<td>XZMODP</td>
<td>3</td>
<td>Indicates whether the load module contains one or more modules from another zone and that XZMOD subentries exist in this LMOD entry. (YES or NO).</td>
</tr>
</tbody>
</table>

### MAC

### Table 20. Valid subentries for the MAC entry

<table>
<thead>
<tr>
<th>Subentry name</th>
<th>Maximum length (decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISTLIB</td>
<td>8</td>
<td>Specifies the ddname of the macro distribution library.</td>
</tr>
<tr>
<td>ENAME</td>
<td>8</td>
<td>Name of the entry.</td>
</tr>
<tr>
<td>FMID</td>
<td>7</td>
<td>Specifies the functional owner of this macro.</td>
</tr>
<tr>
<td>GENASM</td>
<td>*</td>
<td>A linked list containing the names of the assemblies that have to be done during APPLY each time this macro is modified. Each name may be up to 8 bytes in length.</td>
</tr>
</tbody>
</table>
### Table 20. Valid subentries for the MAC entry (continued)

<table>
<thead>
<tr>
<th>Subentry name</th>
<th>Maximum length (decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LASTUPD</td>
<td>7</td>
<td>Cause of last change to this MAC entry.</td>
</tr>
<tr>
<td>LASTUPDTYPE</td>
<td>3</td>
<td>Indicates how the entry was last changed.</td>
</tr>
<tr>
<td>MALIAS</td>
<td>*</td>
<td>A linked list containing the alias names for this macro. Each name may be up to 8 bytes in length.</td>
</tr>
<tr>
<td>RMID</td>
<td>7</td>
<td>Identifies that last SYSMOD that replaced this macro.</td>
</tr>
<tr>
<td>SYSLIB</td>
<td>8</td>
<td>Specifies the ddname of the target library for the macro.</td>
</tr>
<tr>
<td>UMID</td>
<td>*</td>
<td>A linked list containing the 7-byte names of all those SYSMODs that updated this macro since it was last replaced.</td>
</tr>
</tbody>
</table>

### MOD

**Table 21. Valid subentries for the MOD entry**

<table>
<thead>
<tr>
<th>Subentry name</th>
<th>Maximum length (decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASSEMBLE</td>
<td>3</td>
<td>Indicates whether the source for this module must always be assembled (YES or NO).</td>
</tr>
<tr>
<td>CSECT</td>
<td>*</td>
<td>A linked list containing the names of the CSECTs present in this module. Each name may be up to 8 bytes in length.</td>
</tr>
<tr>
<td>DALIAS</td>
<td>8</td>
<td>Specifies the alias name for the module, where the alias exists only in the distribution library.</td>
</tr>
<tr>
<td>DISTLIB</td>
<td>8</td>
<td>Specifies the ddname of the module distribution library.</td>
</tr>
<tr>
<td>ENAME</td>
<td>8</td>
<td>Name of the entry.</td>
</tr>
<tr>
<td>FMID</td>
<td>7</td>
<td>Identifies the functional owner of this module.</td>
</tr>
<tr>
<td>LASTUPD</td>
<td>7</td>
<td>Cause of last change to this MOD entry.</td>
</tr>
<tr>
<td>LASTUPDTYPE</td>
<td>3</td>
<td>Indicates how the entry was last changed.</td>
</tr>
<tr>
<td>LEPARM</td>
<td>*</td>
<td>A character string containing the link-edit attributes that must be used when this module is link-edited. This is not a linked list.</td>
</tr>
<tr>
<td>LMOD</td>
<td>*</td>
<td>A linked list containing the names of the load modules into which this module was copied or link-edited. Each name may be up to 8 bytes in length.</td>
</tr>
<tr>
<td>RMID</td>
<td>7</td>
<td>Identifies the last SYSMOD that replaced this module.</td>
</tr>
<tr>
<td>RMIDASM</td>
<td>3</td>
<td>Specifies that the last replacement (RMID) to the module was done by a SYSMOD that caused an assembly of the module as a result of a source or macro modification (YES or NO).</td>
</tr>
<tr>
<td>TALIAS</td>
<td>*</td>
<td>A linked list containing the alias names for the module, where the alias exists in the distribution library and, for copied modules, also in the target library. Each name may be up to 8 bytes in length.</td>
</tr>
<tr>
<td>UMID</td>
<td>*</td>
<td>A linked list containing 7-byte names of all those SYSMODs that have updated this module since it was last replaced.</td>
</tr>
<tr>
<td>XZLMOD</td>
<td>*</td>
<td>A linked list containing XZLMOD subentries that contain the name of each load module in another zone into which this module was added by the LINK MODULE command. The name of the zone containing each load module is also indicated. Each subentry has the format <code>load module,zone</code> and may be up to 16 bytes in length.</td>
</tr>
</tbody>
</table>
### OPTIONS

Table 22. Valid subentries for the OPTIONS entry

<table>
<thead>
<tr>
<th>Subentry name</th>
<th>Maximum length (decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMS</td>
<td>8</td>
<td>Specifies the name of the UTILITY entry that SMP/E is to use to obtain information when calling the Access Method Services (AMS) utility.</td>
</tr>
<tr>
<td>ASM</td>
<td>8</td>
<td>Specifies the name of the UTILITY entry that SMP/E is to use to obtain information when calling the assembler utility.</td>
</tr>
<tr>
<td>CHANGEFILE</td>
<td>3</td>
<td>Specifies whether library change file records should be created during APPLY and RESTORE processing (YES or NO)</td>
</tr>
<tr>
<td>COMP</td>
<td>8</td>
<td>Specifies the name of the UTILITY entry that SMP/E is to use to obtain information when calling the utility program to compress data sets.</td>
</tr>
<tr>
<td>COMPACT</td>
<td>3</td>
<td>Specifies whether SMPPTS members should be compacted during RECEIVE and GZONEMERGE command processing (YES or NO)</td>
</tr>
<tr>
<td>COPY</td>
<td>8</td>
<td>Specifies the name of the UTILITY entry that SMP/E is to use to obtain information when calling the copy utility.</td>
</tr>
<tr>
<td>DSPREFIX</td>
<td>26</td>
<td>Specifies the data set prefix to be used to construct the full data set name when SMPTLIB data sets are being allocated for RELFILEs.</td>
</tr>
<tr>
<td>DSSPACE</td>
<td>14</td>
<td>Specifies the primary and secondary space allocation (in tracks) and the number of directory blocks to be allocated for each SMPTLIB data set. It is in the format <code>primary,secondary,directory, mnnn,mnnn,mnnn</code>.</td>
</tr>
<tr>
<td>ENAME</td>
<td>8</td>
<td>Name of the entry.</td>
</tr>
<tr>
<td>EXRTYDD</td>
<td>*</td>
<td>A linked list containing dnames ineligible for retry processing after an x37 abend. Each ddname may be up to 8 bytes in length.</td>
</tr>
<tr>
<td>FIXCAT</td>
<td>*</td>
<td>A linked list that contains the Fix Categories whose HOLDDATA is considered during APPLY, ACCEPT, and REPORT MISSINGFIX command processing. Each value can be up to 64 bytes in length.</td>
</tr>
<tr>
<td>HFSCOPY</td>
<td>8</td>
<td>Specifies the name of the UTILITY entry that SMP/E is to use to obtain information when calling the hierarchical file system copy utility.</td>
</tr>
<tr>
<td>IOSUP</td>
<td>8</td>
<td>Specifies the name of the UTILITY entry that SMP/E is to use to obtain information when calling the IEHIOSUP utility program to process maintenance for an OS/VS1 system.</td>
</tr>
<tr>
<td>LKED</td>
<td>8</td>
<td>Specifies the name of the UTILITY entry that SMP/E is to use to obtain information when calling the link-edit utility.</td>
</tr>
<tr>
<td>MSGFILTER</td>
<td>3</td>
<td>Specifies whether messages issued to SMPOUT should be filtered.</td>
</tr>
<tr>
<td>MSGWIDTH</td>
<td>7</td>
<td>Specifies the width in which the message issued to SMPOUT should be formatted.</td>
</tr>
<tr>
<td>NOPURGE</td>
<td>3</td>
<td>Indicates whether after SMP/E accepts SYSMODS, it should not delete the associated global zone SYSMOD entries, SMPPTS MCS entries, or SMPTLIB data sets (YES or NO).</td>
</tr>
<tr>
<td>NOREJECT</td>
<td>3</td>
<td>Specifies that the global zone SYSMOD entry and the associated MCS entry should not be deleted after the SYSMOD is restored. (YES or NO).</td>
</tr>
</tbody>
</table>
### OPTIONS

Table 22. Valid subentries for the OPTIONS entry (continued)

<table>
<thead>
<tr>
<th>Subentry name</th>
<th>Maximum length (decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORDERRET</td>
<td>4</td>
<td>Indicates the retention period, in days, that ORDER entries are kept in the global zone before being deleted.</td>
</tr>
<tr>
<td>PAGELEN</td>
<td>4</td>
<td>Specifies the page length for SMPHRPT, SMPLIST, SMPOUT, and SMPRPT.</td>
</tr>
<tr>
<td>PEMAX</td>
<td>4</td>
<td>Specifies the maximum number of subentries that can be present in any CSI entry.</td>
</tr>
<tr>
<td>RECZGRP</td>
<td>*</td>
<td>A linked list containing 8-byte zone and zonerset names eligible for APPLYCHECK and ACCEPTCHECK processing during the SYSMOD selection phase of RECEIVE.</td>
</tr>
<tr>
<td>RECEXCGRP</td>
<td>*</td>
<td>A linked list containing 8-byte zone and zonerset names not eligible for APPLYCHECK and ACCEPTCHECK processing during the SYSMOD selection phase of RECEIVE.</td>
</tr>
<tr>
<td>RETRY</td>
<td>8</td>
<td>Specifies the name of the UTILITY entry that SMP/E is to use to obtain information when calling the utility program to compress a data set after an x37 abend.</td>
</tr>
<tr>
<td>RETRYDDN</td>
<td>*</td>
<td>A linked list containing ddnames eligible for retry processing after an x37 abend. Each ddname may be up to 8 bytes in length.</td>
</tr>
<tr>
<td>SAVEMTS</td>
<td>3</td>
<td>Indicates whether MTSMAC entries should be deleted from the SMPMTS after the SYSMODs that affect those macros have been successfully accepted (YES or NO).</td>
</tr>
<tr>
<td>SAVESTS</td>
<td>3</td>
<td>Indicates whether STSSRC entries should be deleted from the SMPSTS after the SYSMODs that affect the source have been successfully accepted (YES or NO).</td>
</tr>
<tr>
<td>SUPPHOLD</td>
<td>*</td>
<td>A linked list containing 7-byte HOLD Reason IDs for which the HOLDDATA card image is not displayed in Unresolved HOLD Reason Report and Bypassed HOLD Reason Report for Apply and Accept command processing, and the SYSMOD Comparison HOLDDATA Report for REPORT SYSMODS command processing.</td>
</tr>
<tr>
<td>UPDATE</td>
<td>8</td>
<td>Specifies the name of the UTILITY entry that SMP/E is to use to obtain information when calling the update utility.</td>
</tr>
<tr>
<td>ZAP</td>
<td>8</td>
<td>Specifies the name of the UTILITY entry that SMP/E is to use to obtain information when calling the superzap utility.</td>
</tr>
</tbody>
</table>

### ORDER

Table 23. Valid subentries for the ORDER entry

<table>
<thead>
<tr>
<th>Subentry name</th>
<th>Maximum length (decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>APARS</td>
<td>*</td>
<td>A linked list containing 7-byte names of all the APARs for which resolving PTFs are to be in the order.</td>
</tr>
<tr>
<td>CONTENT</td>
<td>8</td>
<td>Indicates the content selected for the order. The value may be ALL, APARS, CRITICAL, HOLDDATA, PTFs, or a Recommended Service Update SOURCEID value (RSURYyymm).</td>
</tr>
<tr>
<td>DOWNLDATE</td>
<td>5</td>
<td>Indicates the date (yyddd) on which the order was downloaded and stored in the SMPNTS directory.</td>
</tr>
<tr>
<td>DOWNLTIME</td>
<td>8</td>
<td>Indicates the time (hh:mm:ss) at which the order was downloaded and stored in the SMPNTS directory.</td>
</tr>
<tr>
<td>ENAME</td>
<td>8</td>
<td>Name of the entry.</td>
</tr>
</tbody>
</table>
### Table 23. Valid subentries for the ORDER entry (continued)

<table>
<thead>
<tr>
<th>Subentry name</th>
<th>Maximum length (decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORDERDATE</td>
<td>5</td>
<td>Indicates the date (yyddd) on which the order request was made to the server.</td>
</tr>
<tr>
<td>ORDERID</td>
<td>10</td>
<td>Specifies the order identifier assigned by the server when the order was created. This value is used to correlate ORDER entries in the global zone with orders processed by the server.</td>
</tr>
<tr>
<td>ORDERSERVER</td>
<td>*</td>
<td>A linked list containing the &lt;ORDERSERVER&gt; tags used to identify the server used to fulfill the order request.</td>
</tr>
<tr>
<td>OVERTIME</td>
<td>8</td>
<td>Indicates the time (hh:mm:ss) at which the order request was made to the IBM Automated Delivery Request server.</td>
</tr>
<tr>
<td>PKGID</td>
<td>50</td>
<td>Specifies the order package id value used to create a package subdirectory within the SMPNTS directory to contain the order's package files.</td>
</tr>
<tr>
<td>PTFs</td>
<td>*</td>
<td>A linked list containing 7-byte names of all the PTF SYSMODs to be in the order.</td>
</tr>
<tr>
<td>STATUS</td>
<td>10</td>
<td>Specifies the current status of the ORDER entry. Possible values are PENDING, DOWNLOADED, and ERROR.</td>
</tr>
<tr>
<td>USERID</td>
<td>8</td>
<td>Specifies the userid associated with the address space where SMP/E was executing when the ORDER entry was created.</td>
</tr>
<tr>
<td>ZONES</td>
<td>*</td>
<td>A linked list containing 8-byte names of all the target zones used to define the scope of the software inventory associated with this order.</td>
</tr>
</tbody>
</table>

### Table 24. Valid subentries for the PRODUCT entry

<table>
<thead>
<tr>
<th>Subentry name</th>
<th>Maximum length (decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DESCRIPTION</td>
<td>64</td>
<td>Specifies the name of this product.</td>
</tr>
<tr>
<td>PRODID</td>
<td>8</td>
<td>Product identifier</td>
</tr>
<tr>
<td>PRODSUP</td>
<td>*</td>
<td>A linked list containing the prodid,vv.rr.mm values for the products superseded by this product.</td>
</tr>
<tr>
<td>RECDATE</td>
<td>5</td>
<td>Specifies the date on which this product was received.</td>
</tr>
<tr>
<td>RECTIME</td>
<td>8</td>
<td>Specifies the time at which the ++PRODUCT MCS was received.</td>
</tr>
<tr>
<td>REWORK</td>
<td>8</td>
<td>Specifies the level of this PRODUCT, which was received again for minor changes.</td>
</tr>
<tr>
<td>SREL</td>
<td>*</td>
<td>A linked list containing 4-byte names of the system or subsystem releases on which the PRODUCT can be installed.</td>
</tr>
<tr>
<td>UCLDATE</td>
<td>5</td>
<td>Specifies the date on which this PRODUCT entry was last modified through UCLIN.</td>
</tr>
<tr>
<td>UCLTIME</td>
<td>8</td>
<td>Specifies the time at which this PRODUCT entry was last modified through UCLIN.</td>
</tr>
<tr>
<td>URL</td>
<td>256</td>
<td>Specifies a uniform resource locator (URL) that can be accessed to obtain additional information about this product.</td>
</tr>
<tr>
<td>VENDOR</td>
<td>64</td>
<td>Specifies the name of the vendor supplying the product.</td>
</tr>
<tr>
<td>VRM</td>
<td>8</td>
<td>Specifies the version, release, and modification level (vv.rr.mm) of this product.</td>
</tr>
</tbody>
</table>
**GIMAPI**

**Note:** For a PRODUCT entry, the ENAME value is always blank, because the PRODID and VRM subentries, taken together, make up the “entry name” normally provided by ENAME. For each PRODUCT entry satisfying the QUERY request, the PRODID and VRM subentries are returned instead of ENAME.

### PROGRAM

Table 25. Valid subentries for the PROGRAM entry

<table>
<thead>
<tr>
<th>Subentry name</th>
<th>Maximum length (decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALIAS</td>
<td>*</td>
<td>A linked list containing alias names associated with the program element. Each alias name may be up to 8 bytes in length.</td>
</tr>
<tr>
<td>DISTLIB</td>
<td>8</td>
<td>Identifies the ddname of the distribution library for the program element</td>
</tr>
<tr>
<td>ENAME</td>
<td>8</td>
<td>Name of the entry.</td>
</tr>
<tr>
<td>FMID</td>
<td>7</td>
<td>Specifies the functional owner of the data element.</td>
</tr>
<tr>
<td>LASTUPD</td>
<td>7</td>
<td>Cause of last change to this program element entry.</td>
</tr>
<tr>
<td>LASTUPDTYPE</td>
<td>3</td>
<td>Indicates how the entry was last changed.</td>
</tr>
<tr>
<td>RMID</td>
<td>7</td>
<td>Identifies the last SYSMOD that replaced this program element.</td>
</tr>
<tr>
<td>SYSLIB</td>
<td>8</td>
<td>Identifies the ddname of the target library for the program element.</td>
</tr>
</tbody>
</table>

### SRC

Table 26. Valid subentries for the SRC entry

<table>
<thead>
<tr>
<th>Subentry name</th>
<th>Maximum length (decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISTLIB</td>
<td>8</td>
<td>Specifies the ddname of the distribution library for the source.</td>
</tr>
<tr>
<td>ENAME</td>
<td>8</td>
<td>Name of the entry.</td>
</tr>
<tr>
<td>FMID</td>
<td>7</td>
<td>Specifies the functional owner of this source.</td>
</tr>
<tr>
<td>LASTUPD</td>
<td>7</td>
<td>Cause of last change to this SRC entry.</td>
</tr>
<tr>
<td>LASTUPDTYPE</td>
<td>3</td>
<td>Indicates how the entry was last changed.</td>
</tr>
<tr>
<td>RMID</td>
<td>7</td>
<td>Identifies that last SYSMOD that replaced this source.</td>
</tr>
<tr>
<td>SYSLIB</td>
<td>8</td>
<td>Specifies the ddname of the target library for this source.</td>
</tr>
<tr>
<td>UMID</td>
<td>*</td>
<td>A linked list containing 7-byte names of all those SYSMODs that updated this source since it was last replaced.</td>
</tr>
</tbody>
</table>

### SYSMOD (GLOBAL zone)

Table 27. Valid subentries for the SYSMOD entry (GLOBAL zone)

<table>
<thead>
<tr>
<th>Subentry name</th>
<th>Maximum length (decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCID</td>
<td>*</td>
<td>A linked list containing 7-byte names of the distribution zones into which the SYSMOD has been successfully accepted.</td>
</tr>
<tr>
<td>APPID</td>
<td>*</td>
<td>A linked list containing the 7-byte names of the target zones into which the SYSMOD has been successfully applied.</td>
</tr>
<tr>
<td>Data Element(^1)</td>
<td>*</td>
<td>A linked list containing the names of data element replacements contained in the SYSMOD. Each name may be up to 8 bytes in length.</td>
</tr>
<tr>
<td>DELETE(^2)</td>
<td>*</td>
<td>A linked list containing 7-byte names of SYSMODs deleted by this SYSMOD.</td>
</tr>
</tbody>
</table>
Table 27. Valid subentries for the SYSMOD entry (GLOBAL zone) (continued)

<table>
<thead>
<tr>
<th>Subentry name</th>
<th>Maximum length (decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DELLMOD</td>
<td>3</td>
<td>Indicates whether the SYSMOD contained a ++DELETE statement (YES or NO).</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>64</td>
<td>Descriptive name associated with a SYSMOD</td>
</tr>
<tr>
<td>DLMOD</td>
<td>*</td>
<td>A linked list containing the names of load modules to be deleted by ++DELETE statements contained in this SYSMOD. Each name may be up to 8 bytes in length.</td>
</tr>
<tr>
<td>ELEMENT</td>
<td>*</td>
<td>A &quot;pseudo-subentry&quot; name that refers to all element types. It is the same as individually requesting each of these elements: • All data elements • All hierarchical file system elements • JAR and JARUPD • MAC and MACUPD • MOD • PROGRAM, SZAP, and XZAP • SRC and SRCUPD</td>
</tr>
<tr>
<td>ELEMMOV</td>
<td>3</td>
<td>Indicates whether the SYSMOD contained a ++MOVE statement (YES or NO).</td>
</tr>
<tr>
<td>EMOVE</td>
<td>*</td>
<td>A linked list containing names of elements and load modules to be moved by the ++MOVE statements contained in this SYSMOD. Each name may be up to 8 bytes in length.</td>
</tr>
<tr>
<td>ENAME</td>
<td>8</td>
<td>Name of the entry.</td>
</tr>
<tr>
<td>ERROR</td>
<td>3</td>
<td>Indicates whether an error has occurred during the processing of this SYSMOD (YES or NO).</td>
</tr>
<tr>
<td>FEATURE</td>
<td>*</td>
<td>A linked list containing the 8-byte names of software features that contain this SYSMOD.</td>
</tr>
<tr>
<td>FESN</td>
<td>7</td>
<td>Identifies the field engineering (FE) service number.</td>
</tr>
<tr>
<td>FMID</td>
<td>7</td>
<td>Identifies the function SYSMOD to which this SYSMOD is applicable.</td>
</tr>
<tr>
<td>Hierarchical File System Element(^3)</td>
<td>*</td>
<td>A linked list containing the names of hierarchical file system element replacements (++hfs_element statements) in the SYSMOD. Each name may be up to 8 bytes in length.</td>
</tr>
</tbody>
</table>
| HOLDDATA         | *                        | A linked list of the hold information associated with the SYSMOD. Each data value in the list is in the format holdtype,reason,sysmod with the following definitions:  
  
  holdtype is one of the following values: • ERROR • FIXCAT • SYSTEM(EXT) • SYSTEM(INT) • USER  
  
  reason is the 1- to 7-character reason ID.  
  
  sysmod is the SYSMOD ID specified on the ++HOLD. |
| JAR              | *                        | A linked list containing the names of JAR file replacements (++JAR statements) in the SYSMOD. Each name may be up to 8 bytes in length. |
| JARUPD           | *                        | A linked list containing the names of JAR file updates (++JARUPD statements) in the SYSMOD. Each name may be up to 8 bytes in length. |
### Table 27. Valid subentries for the SYSMOD entry (GLOBAL zone) (continued)

<table>
<thead>
<tr>
<th>Subentry name</th>
<th>Maximum length (decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JCLIN</td>
<td>3</td>
<td>Indicates whether this SYSMOD contained inline JCLIN (YES or NO).</td>
</tr>
<tr>
<td>MAC</td>
<td>*</td>
<td>A linked list containing the names of macro replacements (++MAC statements) in the SYSMOD. Each name may be up to 8 bytes in length.</td>
</tr>
<tr>
<td>MACUPD</td>
<td>*</td>
<td>A linked list containing the names of macro updates (++MACUPD statements) in the SYSMOD. Each name may be up to 8 bytes in length.</td>
</tr>
<tr>
<td>MOD</td>
<td>*</td>
<td>A linked list containing the names of modules replacements (++MOD statements) in the SYSMOD. Each name may be up to 8 bytes in length.</td>
</tr>
<tr>
<td>NPRE2</td>
<td>*</td>
<td>A linked list containing the 7-byte names of negative requisite SYSMODs—that is, SYSMODs that must not be present in the system at the same time as this SYSMOD.</td>
</tr>
<tr>
<td>PRE2</td>
<td>*</td>
<td>A linked list containing the 7-byte names of prerequisite SYSMODs—that is, SYSMODs that must be present before this SYSMOD can be installed.</td>
</tr>
<tr>
<td>PROGRAM</td>
<td>*</td>
<td>A linked list containing the names of program element replacements (++PROGRAM statements) in the SYSMOD. Each name may be up to 8 bytes in length.</td>
</tr>
<tr>
<td>RECDATE</td>
<td>5</td>
<td>Specifies the date on which this SYSMOD was received.</td>
</tr>
<tr>
<td>RECTIME</td>
<td>8</td>
<td>Specifies the time at which this SYSMOD was received.</td>
</tr>
<tr>
<td>RENLMOD</td>
<td>3</td>
<td>Indicates whether the SYSMOD contained a ++RENAME statement (YES or NO).</td>
</tr>
<tr>
<td>REQ2</td>
<td>*</td>
<td>A linked list containing the 7-byte names of requisite SYSMODs—that is, SYSMODs that must be installed concurrent with this SYSMOD.</td>
</tr>
<tr>
<td>REWORK</td>
<td>8</td>
<td>Identifies the level of the SYSMOD, which was received again for minor changes.</td>
</tr>
<tr>
<td>RLMOD</td>
<td>*</td>
<td>A linked list containing the names of the load modules to be renamed by ++RENAME statements in this SYSMOD. Each name may be up to 8 bytes in length.</td>
</tr>
<tr>
<td>SMODTYPE</td>
<td>8</td>
<td>Indicates the type of SYSMOD (APAR, FUNCTION, PTF, UNKNOWN, or USERMOD).</td>
</tr>
<tr>
<td>SOURCEID</td>
<td>*</td>
<td>A linked list containing the names of strings assigned to this SYSMOD during RECEIVE processing. Each name can be up to 64 characters in length.</td>
</tr>
<tr>
<td>SRC</td>
<td>*</td>
<td>A linked list containing the names of source replacements (++SRC statements) in the SYSMOD. Each name may be up to 8 bytes in length.</td>
</tr>
<tr>
<td>SRCUPD</td>
<td>*</td>
<td>A linked list containing the names of source updates (++SRCUPD statements) in the SYSMOD. Each name may be up to 8 bytes in length.</td>
</tr>
<tr>
<td>SREL2</td>
<td>*</td>
<td>A linked list containing the names of the system or subsystem releases on which the SYSMOD can be installed. Each name may be up to four bytes in length.</td>
</tr>
<tr>
<td>SUPING2</td>
<td>*</td>
<td>A linked list containing the 7-byte names of SYSMODs superseded by this SYSMOD.</td>
</tr>
</tbody>
</table>
### Table 27. Valid subentries for the SYSMOD entry (GLOBAL zone) (continued)

<table>
<thead>
<tr>
<th>Subentry name</th>
<th>Maximum length (decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SZAP</td>
<td>*</td>
<td>A linked list containing the names of module superzaps (++ZAP statements) in the SYSMOD. Each name may be up to 8 bytes in length.</td>
</tr>
<tr>
<td>TLIBPREFIX</td>
<td>26</td>
<td>Specifies the high-level data set name qualifier of the SMPTLIB data sets used to receive the SYSMOD, which was packaged in RELFILEs.</td>
</tr>
<tr>
<td>VERSION2</td>
<td>*</td>
<td>A linked list containing 7-byte names of function SYSMODs that are versioned by this SYSMOD.</td>
</tr>
</tbody>
</table>

**Notes:**
1. For the data element subentry, the data is returned to the caller in separate linked lists for each unique data element type requested.
2. When PRE, NPRE, REQ, DELETE, SUPING, VERSION, FMID, or SREL subentries are requested, a subentry structure of type 'VER' is returned, along with the necessary associated VER pseudo-subentries. See “VER” on page 392 for more details.
3. For the hierarchical file system element subentry the data is returned to the caller in separate linked lists for each unique hierarchical file system element type requested.
4. For SYSMODs that are only superseded, a value of "UNKNOWN" is returned for the SMODTYPE subentry.

### SYSMOD

### Table 28. Valid subentries for the SYSMOD entry (DLIB and target zones)

<table>
<thead>
<tr>
<th>Subentry name</th>
<th>Maximum length (decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCEPT</td>
<td>3</td>
<td>Indicates whether the SYSMOD has been successfully accepted (YES or NO). This subentry appears only for DLIB zones.</td>
</tr>
<tr>
<td>APPLY</td>
<td>3</td>
<td>Indicates whether the SYSMOD has been successfully applied (YES or NO). This subentry appears only for target zones.</td>
</tr>
<tr>
<td>ASSEM</td>
<td>*</td>
<td>A linked list containing the names of the assemblies done during the installation of this SYSMOD. Each name may be up to 8 bytes in length.</td>
</tr>
<tr>
<td>BYPASS</td>
<td>3</td>
<td>Indicates whether the BYPASS operand was specified when this SYSMOD was installed (YES or NO).</td>
</tr>
<tr>
<td>CIFREQ</td>
<td>*</td>
<td>A linked list of entries containing CIFREQ data that list the conditional requisites that must be installed when this function SYSMOD is installed. Each CIFREQ entry has the format cause(\ast),requisite and may be up to 15 bytes in length.</td>
</tr>
<tr>
<td>DELBY</td>
<td>7</td>
<td>Specifies the SYSMOD that deleted this SYSMOD.</td>
</tr>
<tr>
<td>DELETE2</td>
<td>*</td>
<td>A linked list containing 7-byte names of SYSMODs deleted by this SYSMOD.</td>
</tr>
<tr>
<td>DELLMOD</td>
<td>3</td>
<td>Indicates whether the SYSMOD contained a ++DELETE statement (YES or NO).</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>64</td>
<td>Descriptive name associated with a SYSMOD</td>
</tr>
<tr>
<td>DLMOD</td>
<td>*</td>
<td>A linked list containing the names of load modules to be deleted by ++DELETE statements contained in this SYSMOD. Each name may be up to 8 bytes in length.</td>
</tr>
</tbody>
</table>
Table 28. Valid subentries for the SYSMOD entry (DLIB and target zones) (continued)

<table>
<thead>
<tr>
<th>Subentry name</th>
<th>Maximum length (decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Element$^1$</td>
<td>*</td>
<td>A linked list containing the 8-byte names of data element replacements contained in the SYSMOD.</td>
</tr>
</tbody>
</table>
| ELEMENT                  | *                        | A "pseudo-subentry" name that refers to all element types. It is the same as individually requesting each of these elements:  
  - All data elements  
  - All hierarchical file system elements  
  - JAR and JARUPD  
  - MAC and MACUPD  
  - MOD  
  - PROGRAM, SZAP, and XZAP  
  - SRC and SRCUPD |
| ELEMMOV                  | 3                        | Indicates whether the SYSMOD contained a ++MOVE statement (YES or NO). |
| EMOVE                    | *                        | A linked list containing the names of elements and load modules to be moved by the ++MOVE statements contained in this SYSMOD. Each name may be up to 8 bytes in length. |
| ENAME                    | 8                        | Name of the entry. |
| ERROR                    | 3                        | Indicates whether an error has occurred during the processing of this SYSMOD (YES or NO). |
| FEATURE                  | *                        | A linked list containing the 8-byte names of software features that contain this SYSMOD. |
| FESN                     | 7                        | Identifies the field engineering (FE) service number. |
| FMID                     | 7                        | Identifies the function SYSMOD to which this SYSMOD is applicable. |
| Hierarchical File System Element$^3$ | *                      | A linked list containing the names of hierarchical file system element replacements (++hfs_element statements) in the SYSMOD. Each name may be up to 8 bytes in length. |
| IFREQ                    | *                        | A linked list containing 7-byte names of conditional requisites that were installed with this SYSMOD. |
| INSTALLDATE              | 5                        | Specifies the date on which this SYSMOD was installed. |
| INSTALLTIME              | 8                        | Specifies the time at which this SYSMOD was installed. |
| JAR                      | *                        | A linked list containing the names of JAR file replacements (++JAR statements) in the SYSMOD. Each name may be up to 8 bytes in length. |
| JARUPD                   | *                        | A linked list containing the names of JAR file updates (++JARUPD statements) in the SYSMOD. Each name may be up to 8 bytes in length. |
| JCLIN                    | 3                        | Indicates whether this SYSMOD contained inline JCLIN (YES or NO). |
| LASTSUP                  | 7                        | Specifies the most recent SYSMOD that superseded this SYSMOD. All previous superseding SYSMODs are saved in the SUPBY subentry list. |
| LASTUPD                  | 7                        | Cause of last change to this SYSMOD entry. |
| LASTUPDTYPE              | 3                        | Indicates how the entry was last changed. |
| MAC                      | *                        | A linked list containing the names of macro replacements (++MAC statements) in the SYSMOD. Each name may be up to 8 bytes in length. |
| MACUPD                   | *                        | A linked list containing the names of macro updates (++MACUPD statements) in the SYSMOD. Each name may be up to 8 bytes in length. |
Table 28. Valid subentries for the SYSMOD entry (DLIB and target zones) (continued)

<table>
<thead>
<tr>
<th>Subentry name</th>
<th>Maximum length (decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOD</td>
<td>*</td>
<td>A linked list containing the names of module replacements (++MOD statements) in the SYSMOD. Each name may be up to 8 bytes in length.</td>
</tr>
<tr>
<td>NPRE2</td>
<td>*</td>
<td>A linked list containing the 7-byte names of negative prerequisite SYSMODS—that is, SYSMODs that must not be present in the system at the same time as this SYSMOD.</td>
</tr>
<tr>
<td>PRE2</td>
<td>*</td>
<td>A linked list containing the 7-byte names of prerequisite SYSMODS—that is, SYSMODs that must be present before this SYSMOD can be installed.</td>
</tr>
<tr>
<td>PROGRAM</td>
<td>*</td>
<td>A linked list containing the names of program element replacements (++PROGRAM statements) in the SYSMOD. Each name may be up to 8 bytes in length.</td>
</tr>
<tr>
<td>RECDATE</td>
<td>5</td>
<td>Specifies the date on which this SYSMOD was received.</td>
</tr>
<tr>
<td>RECTIME</td>
<td>8</td>
<td>Specifies the time at which this SYSMOD was received.</td>
</tr>
<tr>
<td>REGEN</td>
<td>3</td>
<td>Indicates how the SYSMOD was installed in the target libraries (YES or NO).</td>
</tr>
<tr>
<td>RENLMOD</td>
<td>3</td>
<td>Indicates whether the SYSMOD contained a ++RENAME statement (YES or NO).</td>
</tr>
<tr>
<td>REQ2</td>
<td>*</td>
<td>A linked list containing the 7-byte names of requisite SYSMODS—that is, SYSMODs that must be installed concurrent with this SYSMOD.</td>
</tr>
<tr>
<td>RESDATE</td>
<td>5</td>
<td>Specifies the date on which this SYSMOD was restored.</td>
</tr>
<tr>
<td>RESTIME</td>
<td>8</td>
<td>Specifies the time at which this SYSMOD was restored.</td>
</tr>
<tr>
<td>RESTORE</td>
<td>3</td>
<td>Indicates whether a restore attempt has been made for this SYSMOD (YES or NO).</td>
</tr>
<tr>
<td>REWORK</td>
<td>8</td>
<td>Identifies the level of the SYSMOD, which was received again for minor changes.</td>
</tr>
<tr>
<td>RLMOD</td>
<td>*</td>
<td>A linked list containing the names of the load modules to be renamed by ++RENAME statements in this SYSMOD. Each name may be up to 8 bytes in length.</td>
</tr>
<tr>
<td>SMODTYPE</td>
<td>8</td>
<td>Indicates the type of SYSMOD (APAR, FUNCTION, PTF, UNKNOWN, or USERMOD).</td>
</tr>
<tr>
<td>SOURCEID</td>
<td>*</td>
<td>A linked list containing the names of strings assigned to this SYSMOD during RECEIVE processing. Each name can be up to 64 characters in length.</td>
</tr>
<tr>
<td>SRC</td>
<td>*</td>
<td>A linked list containing the names of source replacements (++SRC statements) in the SYSMOD. Each name may be up to 8 bytes in length.</td>
</tr>
<tr>
<td>SRCUPD</td>
<td>*</td>
<td>A linked list containing the names of source updates (++SRCUPD statements) in the SYSMOD. Each name may be up to 8 bytes in length.</td>
</tr>
<tr>
<td>SUPBY</td>
<td>*</td>
<td>A linked list containing the 7-byte names of SYSMODs that superseded this SYSMOD. The most recent SYSMOD to supersede this SYSMOD is not included in the SUPBY list. It is saved in the LASTSUP field. Therefore, to simply determine if a SYSMOD is superseded or not, the LASTSUP subentry should be interrogated instead of SUPBY.</td>
</tr>
<tr>
<td>SUPING2</td>
<td>*</td>
<td>A linked list containing the 7-byte names of SYSMODs superseded by this SYSMOD.</td>
</tr>
</tbody>
</table>
Table 28. Valid subentries for the SYSMOD entry (DLIB and target zones) (continued)

<table>
<thead>
<tr>
<th>Subentry name</th>
<th>Maximum length (decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SZAP</td>
<td>*</td>
<td>A linked list containing the 8-byte names of module superzaps (++ZAP statements) in the SYSMOD. Each name may be up to 8 bytes in length.</td>
</tr>
<tr>
<td>UCLDATE</td>
<td>5</td>
<td>Specifies the date on which this SYSMOD was last modified through UCLIN.</td>
</tr>
<tr>
<td>UCLTIME</td>
<td>8</td>
<td>Specifies the time at which this SYSMOD was last modified through UCLIN.</td>
</tr>
<tr>
<td>VERSION2</td>
<td>*</td>
<td>A linked list containing 7-byte names of function SYSMODs that are versioned by this SYSMOD.</td>
</tr>
<tr>
<td>XZAP</td>
<td>*</td>
<td>A linked list containing the names of module superzaps the SYSMOD (++ZAP statements) that contain an EXPAND statement (indicating that the module should be expanded before it is updated). Each name may be up to 8 bytes in length.</td>
</tr>
</tbody>
</table>

Notes:

1. For the data element subentry, the data is returned to the caller in separate linked lists for each unique data element type requested.
2. When PRE, NPRE, REQ, DELETE, SUPING, or VERSION subentries are requested, a subentry structure of type 'VER' is returned for each unique version statement, along with its associated pseudo-subentries. See "VER" on page 392 for more details.
3. For the hierarchical file system element subentry the data is returned to the caller in separate linked lists for each unique hierarchical file system element type requested.
4. For SYSMODs that are only superseded, a value of "UNKNOWN" is returned for the SMODTYPE subentry.

TARGETZONE

Table 29. Valid subentries for the TARGETZONE entry

<table>
<thead>
<tr>
<th>Subentry name</th>
<th>Maximum length (decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENAME</td>
<td>8</td>
<td>Name of the entry.</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>8</td>
<td>The name of the OPTIONS entry in the global zone that should be used when processing this target zone.</td>
</tr>
<tr>
<td>RELATED</td>
<td>7</td>
<td>The name of the distribution zone to which this target zone is related.</td>
</tr>
<tr>
<td>SREL</td>
<td>*</td>
<td>A linked list containing the names of the system releases supported within this target zone. Each name may be up to four bytes in length.</td>
</tr>
<tr>
<td>TIEDTO</td>
<td>*</td>
<td>A linked list containing the names of other target zones that either supplied modules for load modules controlled by this target zone or control load modules that have been link-edited with modules supplied by this target zone. Each name may be up to 8 bytes in length.</td>
</tr>
<tr>
<td>UPGLEVEL</td>
<td>8</td>
<td>Indicates the highest SMP/E release level that is allowed to make incompatible changes to the zone. If SMP/E attempts to make an incompatible change to a zone and the release level of SMP/E is higher than the UPGRADE level in that zone, then the incompatible change will not be made. This value is in the form vr.pp, where vr represents the version and release of SMP/E and pp represents the PTF level of SMP/E.</td>
</tr>
</tbody>
</table>
Table 29. Valid subentries for the TARGETZONE entry (continued)

<table>
<thead>
<tr>
<th>Subentry name</th>
<th>Maximum length (decimal)</th>
<th>Description</th>
</tr>
</thead>
</table>
| XZLINK        | 9                        | Specifies whether APPLY and RESTORE processing in another zone should automatically update load modules in this zone when cross-zone modules previously added to those load modules by the LINK command are changed. This subentry value can contain one of these values:  
• DEFERRED  
• AUTOMATIC |
| ZDESC         | 500                      | A user-written description for this zone. |

**UTILITY**

Table 30. Valid subentries for the UTILITY entry

<table>
<thead>
<tr>
<th>Subentry name</th>
<th>Maximum length (decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENAME</td>
<td>8</td>
<td>Name of the entry.</td>
</tr>
<tr>
<td>LIST</td>
<td>3</td>
<td>Indicates whether member names should be listed when SMP/E invokes a copy utility to perform compress processing, retry processing, or element installation (YES or NO).</td>
</tr>
<tr>
<td>NAME</td>
<td>8</td>
<td>The name of the load module for the utility program that SMP/E is to call.</td>
</tr>
<tr>
<td>UTILPARM</td>
<td>100</td>
<td>Specifies the parameters to be passed to the utility program.</td>
</tr>
<tr>
<td>PRINT</td>
<td>8</td>
<td>Specifies the ddname that is to contain output from the utility.</td>
</tr>
<tr>
<td>RC</td>
<td>2</td>
<td>Specifies the maximum acceptable return code from this utility. This value can be from 0 to 16.</td>
</tr>
</tbody>
</table>

**ZONESET**

Table 31. Valid subentries for the ZONESET entry

<table>
<thead>
<tr>
<th>Subentry name</th>
<th>Maximum length (decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENAME</td>
<td>8</td>
<td>Name of the entry.</td>
</tr>
<tr>
<td>XZREQCHK</td>
<td>3</td>
<td>Indicates whether this ZONESET should be used when establishing the default zone group for the APPLY, ACCEPT, and RESTORE commands. (YES or NO).</td>
</tr>
<tr>
<td>ZONENAME</td>
<td>*</td>
<td>A linked list containing the 7-byte names of the target or distribution zones that are to be part of this ZONESET.</td>
</tr>
</tbody>
</table>

**QUERY command processing**

GIMAPI checks to see if the SMPCSI DDNAME is already allocated. If it is, GIMAPI notes the data set name. GIMAPI then determines if the caller expects GIMAPI to allocate the global CSI data set. The caller indicates this by setting the csilen parameter to zero. The data set name previously noted is used so a shared enqueue can be done. The enqueue is to protect GIMAPI processing from updates by any other SMP/E processing that might be going on at the same time.
If the `csilen` parameter is not zero, GIMAPI verifies that the `csi` parameter provided is a valid CSI data set name.

If the SMPCSI DDNAME has already been allocated, the data set name of the allocation must be the same as the `csi` parameter value. If they match, GIMAPI treats it as if the calling program already did the allocation. The data set is not allocated again (and is not deallocated at the end of the command).

If the CSI data set must be allocated, GIMAPI dynamically allocates the data set with DISP=SHR.

GIMAPI continues by validating the rest of the required QUERY parameters that it has received. Syntax checking is done for each QUERY parameter received. If GIMAPI encounters an error, it stops syntax checking for that parameter list and moves on to begin syntax checking for the next QUERY parameter. If any errors are found, the QUERY command terminates after syntax checking is done. The command continues if warnings are the highest severity found.

GIMAPI enforces these syntax rules:

- When multiple values are specified on the `zone`, `entry` or `subentry` parameters, they must be appropriately separated by commas or spaces.
- When an asterisk (*) is used on the `zone`, `entry` or `subentry` parameters, no other values can be specified.
- Entry types entered on the `entry` parameter must be a valid CSI entry type.
- A specific value can be specified only once on the `zone`, `entry` or `subentry` parameters.
- A condition using the `>`, `>=`, `<` or `<=` operator with a subentry type that is not a date or time field is not allowed.
- There are a set of subentry types that are not allowed to be used in conditions on the `filter` parameter. See section "Filter parameter syntax" on page 363 for the list.
  A null value for a subentry may be specified by using two single quotation marks with no other characters or blanks between them (""'). A null value may be used with the "not equal" (!=) operator to find all entries that have some value — for example, PATH!='".
- A date fixed value must use the format 'yyddd'. GIMAPI checks the characters entered to make sure they are all numeric.
- A time fixed value must use the format 'hh:mm:ss', where hh is a two-digit decimal number in the range 00 through 23, and mm and ss are both two-digit decimal numbers in the range 00 through 59. (That is, in 24-hour clock format.)
- A null value can be used with date and time fields with the = and != operators to check if the values have been set or not.
- A closing quotation mark "" must be found by the end of the `filter` parameter.
- All parentheses must be closed before the `filter` parameter string ends.
- When any other syntax errors are found in the `filter` parameter, GIMAPI issues a general syntax error message specifying the character position with the error.

Once the syntax is checked and zones other than the GLOBAL zone are specified, specific zone names are checked to see if they are ZONESETEs. If so, the ZONESET name is removed from the list of zones passed by the caller and replaced with the zone names from the ZONESET. An attempt is then made to read the ZONEINDEX for each of the zones to be processed to see if any other CSI data
sets need to be dynamically allocated. If one of the zones specified is not a ZONESET and cannot be found in the ZONEINDEX, the QUERY command terminates.

If any of the target zones or DLIB zones are contained in separate CSI data sets than the global CSI, those data sets must be allocated. The command first determines if they are already allocated by the calling routine. If so, that allocation is used and the data sets are not deallocated later. If not, the data sets are allocated with DISP=SHR. In either case, a shared enqueue is done.

The command now attempts to process the query. The requested entries are read from the CSI and compared with the filter. If there is a match, the requested subentries and their values are attached to the output storage. This is a set of data structures linked together, with storage being allocated for each structure used.

Once all entries are processed, any CSI data sets that were dynamically allocated are dequeued and deallocated.

**Note:** Any successful allocations and enqueues are undone, even if errors are found. The data sets are always freed when the program ends.

If the query requested from the calling program did not result in any data being returned, the command *output* parameter is set to zero.

**QUERY command output**

The CSI data is made up of entries, which can be thought of as record types, and subentries, which can be considered the fields within those records. Each entry has a set of subentries. Those subentry values may be a single value, such as FMID subentry of the MOD entry. Other subentries are made up of a list of values, such as the LMOD subentry of the MOD entry, which lists all the LMODs that contain the MOD. Refer to Chapter 5, "SMP/E Data Set Entries" for a complete description of the CSI entry types and their associated subentries.

The data returned from the query follows this principle in that it is organized by entries and subentries.

What is returned from the QUERY command is a pointer to storage containing the output data. The storage is a set of linked lists that describe the output as well as contain the values.

The pointer returned points to the head of a linked list of records describing each of the entry types returned from the query. The entry type structures (ENTRY_LIST) contain a 12-character name of the entry type (padded with blanks) and a pointer to a link list of entry structures (ENTRY) representing the instances of that entry type. It also contains a pointer to the next entry type record.

The entry record points to a linked list of subentry type structures (SUBENTRY) and to the next instance of that entry type. It also contains a string with the name of the entry and a string with the name of the zone where the entry was found.

**Note:** If no subentries are requested on the *subentry* parameter, the entry name and zone name are still returned with the entry structure, with no subentry structures attached.
The subentry type structure contains a 12-character name of the subentry type and a pointer to the value structure (ITEM_LIST), as well as a pointer to the next subentry type.

The value structure contains the length of the value, a pointer to the character string value and a pointer to the next value structure, for subentries that have lists of values.

Refer to "Data structures for QUERY command" on page 390 for details on the structures defined. "Example of QUERY command" contains pictures of what the output looks like in storage.

Some subentries are lists of values, while others are single values. There are some subentries that are composite values as well as being lists. For example, the ZONEINDEX subentry of the GLOBALZONE entry contains the zone name, the CSI data set containing the zone name, and the zone type. Subentries such as these are returned with their parts combined into one string separated by commas, as shown here:

TARGET,UID,TARG1.CSI,TARGET

The order of the parts for subentries whose values have multiple parts follow the description of that subentry in Chapter 5, "SMP/E Data Set Entries".

Other subentries are indicator variables. This means they do not have a value, but the fact that they do (or do not) exist in the CSI indicates something. The PROTECT subentry of the DDDEF entry is an example of this. These types of subentries will have values in the QUERY output. If the subentry exists, the value is set to 'YES', if it does not, the value is set to 'NO'.

If an asterisk is used in either the entry or subentry parameters, QUERY will create a complete list, internally, of the specified type (entries, subentries, or both). If all entries and subentries are requested, then a list for each type is generated. Each list will be processed as if the user had entered it from the command. The following output will result:

**Table 32. Entry and subentry combinations/output**

<table>
<thead>
<tr>
<th>ENTRY/SUBENTRY contents</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>• ENTRY set to *</td>
<td>All valid subentries are returned for all instances of all valid entry types that fulfill the filter restrictions.</td>
</tr>
<tr>
<td>• SUBENTRY set to *</td>
<td></td>
</tr>
<tr>
<td>• ENTRY set to specific entry types</td>
<td>All valid subentries are returned all instances of each of the specified entry types that fulfill the filter restrictions.</td>
</tr>
<tr>
<td>• SUBENTRY set to *</td>
<td></td>
</tr>
<tr>
<td>• ENTRY set to *</td>
<td>The specific subentries are returned for all instances of all entry types that fulfill the filter restrictions.</td>
</tr>
<tr>
<td>• SUBENTRY set to specific subentries</td>
<td></td>
</tr>
</tbody>
</table>

**Example of QUERY command**

Suppose you want to write an application that will perform the following query:

Find all PTF and USERMOD type SYSMODs having an FMID of HMP1E00 that were received after December 1, 2008 and that were applied with BYPASS(anything). Return the SYSMOD name, the zone, the MODs that are replaced by the SYSMOD, and the installation date.
The storage containing the input parameters may be represented like this:

QPARM → CSI → "USERID.GLOBAL.CSI"
  17
ZONE → "ALLTZONES"
  9
ENTRY → "SYSMOD"
  6
SUBENTRY → "MOD, INSTALLDATE"
  16
FILTER
  101

'(SMODTYPE='PTF' | SMODTYPE='USERMOD') & FMID='HMP1900' & APPLY='YES' & BYPASS='YES' & RECDATE='96335''

**Note:** The SYMORD name and zone are automatically returned without your asking for them.

QPARM is a variable containing a pointer to a QUERY_PARMS data structure. The elements of the data structure point to strings in storage containing the values of the parameters.

Your program would then execute this call:

`GIMAPI('QUERY', QPARM, QRESULT, 'ENU', RC, CC, MSGBUFF)`

The result of the query is two SYMORDS, SMOD019 and SMOD022. SMOD019 is in target zone TARG1 with two MODs, MOD01 and MOD02 and was installed on December 6, 2008. SMOD022 is in target zone MYTARG and has no MODs. It was installed on December 14, 2008.

**Figure 67 on page 390** provides a picture of what the output storage looks like. See section "[Data structures for QUERY command](#)" on page 390 for a description of the data structures used in the picture.
**Note:** There are no values for the MOD subentry for SYSMOD SMOD022. Therefore, no SUBENTRY structure is added to the output for that subentry type.

**Data structures for QUERY command**

The data structures shown in this section are defined in a language-independent format. See "Programming in C" on page 396, "Programming in PL/I" on page 398, and "Programming in assembler" on page 400 to see the syntax of the data structures for those languages. Headers or macros with the language-specific
structure definitions are provided in members named GIMHC370 (for C/370™), GIMMPLI (for PL/I), and GIMMASM (for assembler) in libraries with these DDDEFS:

MACLIB Target library
AMAACLIB Distribution library

Note: The pointer and numeric values found in the tables in this section are 4-byte fields.

QUERYParms

Table 33 lists the elements of the data structure used as input to the QUERY command. Only a brief description of each parameter is given here. See “QUERY command” on page 359 for a more complete description.

Table 33. QUERY command input parameters

<table>
<thead>
<tr>
<th>Element</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>csi</td>
<td>Pointer</td>
<td>Global CSI data set name to be searched by the query.</td>
</tr>
<tr>
<td>csilen</td>
<td>Numeric</td>
<td>Number of characters in the csi parameter.</td>
</tr>
<tr>
<td>zone</td>
<td>Pointer</td>
<td>Pointer to character string representing zones to search during the query.</td>
</tr>
<tr>
<td>zonelen</td>
<td>Numeric</td>
<td>Number of characters in the zone parameter.</td>
</tr>
<tr>
<td>entrytype</td>
<td>Pointer</td>
<td>Pointer to character string representing the list of CSI entry types to be returned from the query.</td>
</tr>
<tr>
<td>entrylen</td>
<td>Numeric</td>
<td>Number of characters in the entry parameter.</td>
</tr>
<tr>
<td>subentrytype</td>
<td>Pointer</td>
<td>Pointer to character string representing the list of subentries to be returned for each entry selected during the query.</td>
</tr>
<tr>
<td>subentrylen</td>
<td>Numeric</td>
<td>Number of characters in the subentry parameter.</td>
</tr>
<tr>
<td>filter</td>
<td>Pointer</td>
<td>Pointer to character string representing the set of conditions used to limit the entries being returned by the query.</td>
</tr>
<tr>
<td>filterlen</td>
<td>Numeric</td>
<td>Number of characters in the filter parameter.</td>
</tr>
</tbody>
</table>

ENTRY_LIST

Table 34. General SMP/E entry list structure

<table>
<thead>
<tr>
<th>Element</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>next</td>
<td>Pointer</td>
<td>Pointer to the next entry list structure. This value is zero (0) for the last item in the list.</td>
</tr>
<tr>
<td>entries</td>
<td>Pointer</td>
<td>Pointer to a linked list of entry structures. This value is zero (0) if there are no entries.</td>
</tr>
<tr>
<td>entrytype</td>
<td>Character(12)</td>
<td>Text representation of the type of entry pointed to by the entry list structure.</td>
</tr>
</tbody>
</table>

ENTRY

Table 35. General SMP/E entry structure

<table>
<thead>
<tr>
<th>Element</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>next</td>
<td>Pointer</td>
<td>Pointer to the next entry structure. This value is zero (0) for the last item in the list.</td>
</tr>
<tr>
<td>subentries</td>
<td>Pointer</td>
<td>Pointer to a linked-list of subentry structures. This value is zero (0) if there are no subentries.</td>
</tr>
<tr>
<td>entryname</td>
<td>Character(8)</td>
<td>The name of an SMP/E entry.</td>
</tr>
</tbody>
</table>
Table 35. General SMP/E entry structure (continued)

<table>
<thead>
<tr>
<th>Element</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>zonename</td>
<td>Character(7)</td>
<td>The name of an SMP/E zone.</td>
</tr>
</tbody>
</table>

Table 36. General SMP/E subentry structure

<table>
<thead>
<tr>
<th>Element</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>next</td>
<td>Pointer</td>
<td>Pointer to the next subentry structure. This value is zero (0) for the last item in the list.</td>
</tr>
<tr>
<td>subentrydata</td>
<td>Pointer</td>
<td>Pointer to a linked list of item list structures. This value is zero (0) if there is no subentry data.</td>
</tr>
<tr>
<td>type</td>
<td>Character(12)</td>
<td>Text representation of the type of subentry pointed to by the subentry structure.</td>
</tr>
</tbody>
</table>

VER

The VER structure is used when SYSMOD entry information is requested from the GLOBAL, target, or distribution zone. More specifically, this structure is used to return information that is typically associated with a ++VER statement. Refer to the "Valid subentry types" on page 366 for valid subentry and pseudo-subentry specification for the SYSMOD entry.

For each request for data associated with any of the information that is contained in a ++VER statement, a VER pseudo-subentry structure is returned.

The type value in the subentry structure is set to the 12-character string ‘VER’ and the subentrydata pointer points to a VER pseudo-subentry structure.

For each VER pseudo-subentry returned, a linked list of subentry structures are returned.

Each subentry structure returned will be associated with one of the types of data related to a ++VER statement. In other words, the 'type' field in the subentry structure will contain one of these text strings: SREL, FMID, PRE, NPRE, REQ, DELETE, SUP, or VERSION.

Each subentry structure returned will point to an item_list structure containing the actual VER-related data.
Table 37. VER pseudo-subentry structure (GLOBAL, target, and DLIB zone)

<table>
<thead>
<tr>
<th>Element</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>next</td>
<td>Pointer</td>
<td>Pointer to the next VER pseudo-subentry structure. This value is zero (0) for the last item in the list.</td>
</tr>
<tr>
<td>verdata</td>
<td>Pointer</td>
<td>Pointer to a linked list of subentry structures. This value is zero (0) if there is no data associated with the VER.</td>
</tr>
<tr>
<td>vernum</td>
<td>Character(3)</td>
<td>Contains the relative number of the ++VER statement when the SYSMOD was installed.</td>
</tr>
</tbody>
</table>

**ITEM_LIST**

This structure is used by many of the entry structures to hold an element in a list of subentry values attached to the entry. For example, it may contain a list of SYMMOD values, SRELs, or ALIAS names.

Table 38. Item list structure

<table>
<thead>
<tr>
<th>Element</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>next</td>
<td>Pointer</td>
<td>Pointer to the next item in the list. This value is zero (0) when there are no more items.</td>
</tr>
<tr>
<td>datalen</td>
<td>Numeric</td>
<td>Number of characters in the item string.</td>
</tr>
<tr>
<td>data</td>
<td>Pointer</td>
<td>Pointer to the data associated with this item.</td>
</tr>
</tbody>
</table>

**FREE command**

The FREE command frees all the storage allocated by GIMAPI for query output, message buffer, and any other data that is saved between different calls to GIMAPI. The FREE command will free storage allocated by any previous queries that has not already been freed. It is suggested that the FREE command be run after each query, but this is not required. It is good practice to use the FREE command prior to terminating the calling program.
GIMAPI

FREE command parameters

Here is an example of a call to the FREE command:

GIMAPI('FREE ',0,0,'ENU',rc,cc,msgbuff)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FREE</td>
<td>is a keyword that specifies that GIMAPI is to process a free request. Note that it is necessary to pad the string containing the FREE command with blanks to bring it to a length of eight characters.</td>
</tr>
<tr>
<td>0,0</td>
<td>The parmptr and outptr parameters must both be set to zero (0) on the FREE command.</td>
</tr>
<tr>
<td>ENU</td>
<td>The language parameter must be specified as either ENU, JPN, or three blanks. The FREE command does not return any messages, so it does not matter which of these choices is specified.</td>
</tr>
<tr>
<td>rc</td>
<td>A storage area owned by the calling program representing a 4-byte numeric variable. This variable will contain the return code for the FREE command. GIMAPI will always set this variable to zero (0) after processing the FREE command.</td>
</tr>
<tr>
<td>cc</td>
<td>A storage area owned by the calling program representing a 4-byte numeric variable. This variable will contain the condition code for the FREE command. GIMAPI will always set this variable to zero (0) after processing the FREE command.</td>
</tr>
<tr>
<td>msgbuff</td>
<td>A pointer variable. No messages are issued by the FREE command, but this pointer variable must be passed to GIMAPI.</td>
</tr>
</tbody>
</table>

The parmptr, outptr, language, rc, cc, and msgbuff parameters must all be specified on the GIMAPI call, even though none of these parameters are used by the FREE command.

FREE command processing

The FREE command frees any storage obtained by GIMAPI on previous QUERY requests made by this caller.

VERSION command

The VERSION command is used to query the version of GIMAPI module being accessed by the calling program.

The version is returned as 8 bytes worth of data representing the current version of SMP/E in the format vvrrmmpp, where:

- vv: version
- rr: release
- mm: modification level
- pp: PTF

The version of IBM SMP/E for z/OS, V3R6 is displayed as 36.00. This translates to an 8 byte value of "03060000".

VERSION command parameters

Here is an example of a call to the VERSION command:
No input parameters are defined for this command. A parameter passed to GIMAPI for this command would be ignored.

The output is a pointer to an API_VERSION structure containing the version of the current running GIMAPI module. Refer to

**VERSION command processing**

GIMAPI allocates the storage for an API_VERSION structure. If the storage cannot be allocated, existing message GIM30700E is issued and the module terminates.

If allocation was successful, GIMAPI sets the elements of the structure with the values of the current version, release, modification and PTF level then returns to the calling program.

For example, the level of IBM SMP/E for z/OS, V3R6 is 36.00. As the result of a call to GIMAPI with the VERSION command, the elements of the API_VERSION structure are set as follows:

```plaintext
apiver="03"
apirel="06"
apimod="00"
apiptf="00"
```

The version can be accessed by the user as the complete string, "03060000" or as the previously listed individual parts.

The calling program needs to issue the FREE command at some point after calling the VERSION command.

**VERSION command output**

The data structure shown in this section are defined in a language independent format. See “Programming in C” on page 396, “Programming in PL/I” on page 398, and “Programming in assembler” on page 400 to see the syntax of the data structures for those languages. Headers or macros with the language-specific structure definitions are provided in members named GIMHC370 (for C/370), GIMMPII (for PL/I), and GIMMASM (for assembler) in libraries with these DDDEFs:

<table>
<thead>
<tr>
<th>MACLIB</th>
<th>Target Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMACLIB</td>
<td>Distribution Library</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 39. API version</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Element</strong></td>
</tr>
<tr>
<td>----------------</td>
</tr>
<tr>
<td>apiver</td>
</tr>
<tr>
<td>apirel</td>
</tr>
<tr>
<td>apimod</td>
</tr>
<tr>
<td>apiptf</td>
</tr>
</tbody>
</table>
For GIMAPI to be invoked from a C/370 program, GIMAPI must first be loaded into storage. The calling program must identify the parameter linkage as standard OS linkage and must declare the routine. Figure 69 shows the statements to load and unload the module, the #pragma statement that identifies the linkage, the function declaration, and the calling syntax.

```c
typedef void APIPGM();
typedef void CFUNC();
#pragma linkage(APIPGM,OS)

APIPGM * gimapi;
gimapi = (APIPGM *) fetch("GIMAPI");

(*gimapi) (apicmd,&parmptr,&outptr,language,&rc,&cc,&msgbuff);

release((CFUNC*) gimapi)
```

Figure 69. C syntax of GIMAPI invocation

The FETCH must be done once, then GIMAPI can be invoked any number of times before it is released. The release function requires a pointer to a C function as its parameter. APIPGM becomes an OS program. The typedef of CFUNC is used to cast the `gimapi` parameter so the program compiles correctly.

- **apicmd**
  - The `apicmd` parameter is a string of length 8 that contains the name of the command to be passed to GIMAPI.

- **&parmptr**
  - A `aparm` variable is declared of type QUERY_PARMS. The `parmptr` pointer variable is set to the address of the query parameter structure. The address of the pointer variable is passed to GIMAPI.

- **&outptr**
  - The `outptr` variable is a pointer variable that will be set to the address of the beginning of storage containing the output of the command processing. The address of `outptr` is passed to GIMAPI.

- **language**
  - A character string of length 3 to indicate the language to use for messaging by GIMAPI. Valid values are ENU and JPN.

- **&rc**
  - Address of a variable defined as `long` to be set to the command’s return code by GIMAPI.

- **&cc**
  - Address of a variable defined as `long` to be set to the command’s condition code by GIMAPI.

- **&msgbuff**
  - The `msgbuff` variable is a pointer variable that is set to the head of a linked list of messages that could be created by GIMAPI processing. The elements to the link list are ITEM_LIST structures. The address of the pointer variable is passed to GIMAPI.

**Data structures in C**

Many of the structures contain character string data. The strings are not null terminated.
A header file will be provided in the MACLIB library. The member name for C/370 is GIMHC370.

**QUERY_PARMS**

typedef _Packed struct QUERY_PARMS
{
    char *csi;
    long csilen;
    char *zone;
    long zonelen;
    char *entrytype;
    long entrylen;
    char *subentrytype;
    long subentrylen;
    char *filter;
    long filterlen;
} QUERY_PARMS, * P_QUERY_PARMS;

**API_VERSION**

typedef _Packed struct API_Version
{
    char apiver[2];
    char apirel[2];
    char apimod[2];
    char apiptf[2];
} API_VERSION, * P_API_VERSION;

**ENTRY**

typedef _Packed struct CSI_ENTRY
{
    _Packed struct CSI_ENTRY *next;
    _Packed struct SUBENTRY *subentries;
    char entryname[8];
    char zonename[7];
} CSI_ENTRY, * P_CSI_ENTRY;

**ENTRY_LIST**

typedef _Packed struct ENTRY_LIST
{
    _Packed struct ENTRY_LIST *next;
    _Packed struct CSI_ENTRY *entries;
    char type[12];
} ENTRY_LIST, * P_ENTRY_LIST;

**ITEM_LIST**

typedef _Packed struct ITEM_LIST
{
    _Packed struct ITEM_LIST *next;
    long datalen;
    char *data;
} ITEM_LIST, * P_ITEM_LIST;

**SUBENTRY**

typedef _Packed struct SUBENTRY
{
    _Packed struct SUBENTRY *next;
    void *subentrydata;
    char *entrytype;
} SUBENTRY, * P_SUBENTRY;

**Note:** Given that C requires pointers to indicate the data type they will point to, a void pointer must be used here since more than one structure may be attached, the ITEM_LIST or the VER structure.
Programming in PL/I

Before the GIMAPI program can be invoked from a PL/I program, it must be declared. This identifies the variable as a callable routine and indicates to PL/I that standard OS linkage is used to pass parameters. The program must be loaded into storage before it can be invoked. The declaration, linkage, load and call syntax are shown in Figure 70.

The FETCH must be done once, then GIMAPI can be invoked any number of times before it is released.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>APICMD</td>
<td>A variable defined as CHAR(8) that contains the string representing the command GIMAPI is to process.</td>
</tr>
<tr>
<td>PARMPTR</td>
<td>A command parameter structure (QUERY_PARMS) that contains the parameters of the command being processed is declared in the calling program. PARMPTR is a pointer variable that contains the address of that structure. The variable is passed to GIMAPI.</td>
</tr>
<tr>
<td>OUTPTR</td>
<td>A pointer variable that will be set to the address of the beginning of storage containing the output of the command processing.</td>
</tr>
<tr>
<td>LANGUAGE</td>
<td>3-character national language identifier to use for messages. Valid values are ENU and JPN.</td>
</tr>
<tr>
<td>RC</td>
<td>A return code variable declared as FIXED BIN(31). The value of the variable is set to the return code of GIMAPI call by GIMAPI program.</td>
</tr>
<tr>
<td>CCPTR</td>
<td>A condition code variable declared as FIXED BIN(31). The value of the variable is set to the condition code of GIMAPI call by GIMAPI program.</td>
</tr>
<tr>
<td>MSGBUFF</td>
<td>The MSGBUFF variable is a pointer variable that is set to the head of a linked list of messages that could be created by GIMAPI processing. The elements to the link list are ITEM_LIST structures.</td>
</tr>
</tbody>
</table>

Data structures in PL/I

This section shows the data structures defined in "Data structures for QUERY command" on page 390 as they would appear in PL/I. The calling program must declare all the structures it is going to use before calling GIMAPI.
The data structures defined for the entries are BASED variables. This means no storage is allocated for the variables, but they are used to reference data allocated by the GIMAPI command. The calling program must declare pointer variables to reference the storage.

A macro file for PL/I is provided in member GIMMPLI of the MACLIB library.

**QUERY_PARMS**

```plaintext
DECLARE
1 QUERY_PARMS UNALIGNED,
  2 CSI POINTER,
  2 CSILEN FIXED BIN(31),
  2 ZONE POINTER,
  2 ZONELEN FIXED BIN(31),
  2 ENTRYTYPE POINTER,
  2 ENTRYLEN FIXED BIN(31),
  2 SUBENTRYTYPE POINTER,
  2 SUBENTRYLEN FIXED BIN(31),
  2 FILTER POINTER,
  2 FILTERLEN FIXED BIN(31);
```

**API_VERSION**

```plaintext
DECLARE
1 API_VERSION BASED UNALIGNED,
  2 APIVER CHAR(2),
  2 APIREL CHAR(2),
  2 APIMOD CHAR(2),
  2 APIPTF CHAR(2);
```

**ENTRY**

```plaintext
DECLARE
1 CSI_ENTRY BASED UNALIGNED,
  2 NEXT POINTER,
  2 SUBENTRIES POINTER,
  2 ENTRYNAME CHAR(8),
  2 ZONENAME CHAR(7);
```

**ENTRY_LIST**

```plaintext
DECLARE
1 ENTRY_LIST BASED UNALIGNED,
  2 NEXT POINTER,
  2 ENTRIES POINTER,
  2 TYPE CHAR(12);
```

**ITEM_LIST**

```plaintext
DECLARE
1 ITEM_LIST BASED UNALIGNED,
  2 NEXT POINTER,
  2 DATALEN FIXED BIN(31),
  2 DATA POINTER;
```

**SUBENTRY**

```plaintext
DECLARE
1 SUBENTRY BASED UNALIGNED,
  2 NEXT POINTER,
  2 SUBENTRYDATA POINTER,
  2 TYPE CHAR(12);
```
Programming in assembler

Before the GIMAPI program can be invoked from an assembler program, the load module must be brought into virtual storage using the LOAD macro. It can then be invoked using the CALL macro passing the required parameters. An example of the declares, LOAD, CALL and FREE syntax are shown in Figure 71.

```asm
LA 2,APIPGM
LOAD EPLOC=(2),LOADPT=PGMADR
L 15,PGMADR
CALL (15),(QUERYCMD,QUERY_PARMS@,CMDOUT,APILANG,RC,CC,MSG@)
L 15,PGMADR
CALL (15),(FREECMD,0,CMDOUT,APILANG,RC,CC,MSG@)

APIPGM  DC  CLB'GIMAPI'
QUERYCMD  DC  CLB'QUERY'
FREECMD  DC  CLB'FREE'
APILANG  DC  CL3'ENU'
DS  OF
QUERY_PARMS@  DC  AL4(QUERY_PARMS)
PGMADR  DC  AL4(0)
CMDOUT  DC  AL4(0)
MSG@  DC  AL4(0)
RC  DS  F'0'
CC  DS  F'0'
```

Figure 71. Assembler syntax of GIMAPI invocation

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QUERYCMD</td>
<td>A variable defined as CHAR(8) that contains the string representing the QUERY command GIMAPI is to process.</td>
</tr>
<tr>
<td>FREECMD</td>
<td>A variable defined as CHAR(8) that contains the string representing the FREE command GIMAPI is to process.</td>
</tr>
<tr>
<td>QUERY_PARMS@</td>
<td>A command parameter structure (QUERY_PARMS) that contains the parameters of the command being processed is declared in the calling program. QUERY_PARMS@ is a pointer variable that contains the address of that structure. The variable is passed to GIMAPI.</td>
</tr>
<tr>
<td>CMDOUT</td>
<td>A pointer variable that will be set to the address of the beginning of storage containing the output of the command processing.</td>
</tr>
<tr>
<td>APILANGE</td>
<td>3-character national language identifier to use for messages. Valid values are ENU and JPN.</td>
</tr>
<tr>
<td>RC</td>
<td>A return code variable declared as a fullword. The value of the variable is set to the return code of GIMAPI call by GIMAPI program.</td>
</tr>
</tbody>
</table>
A condition code variable declared as a fullword. The value of the variable is set to the condition code of GIMAPI call by GIMAPI program.

The MSG@ variable is an address variable that is set to the head of a linked list of messages that could be created by GIMAPI processing. The elements to the link list are ITEM_LIST structures.

Data structures in assembler

This section shows the data structures defined in "Data structures for QUERY command" on page 390 as they would appear in assembler. The calling program must declare all the structures it is going to use before calling GIMAPI.

The data structures defined for the entries are DSECTS. This means no storage is allocated for the variables, but they are used to reference data allocated by the GIMAPI command. The calling program must declare address variables to reference the storage.

A macro file for assembler code is provided in member GIMMASM of the MACLIB library.

QUERY_PARMS

<table>
<thead>
<tr>
<th>STRUCTURE</th>
<th>TYPE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>QUERY_PARMS DS</td>
<td>0CL40</td>
<td>PARAMETERS FOR QUERY COMMAND</td>
</tr>
<tr>
<td>PCS1 DS AL4</td>
<td></td>
<td>PTR TO GLOBAL CSI DATASET</td>
</tr>
<tr>
<td>CSILEN DS FL4</td>
<td></td>
<td>LENGTH OF DATA SET NAME</td>
</tr>
<tr>
<td>PZONE DS AL4</td>
<td></td>
<td>PTR TO LIST OF ZONES</td>
</tr>
<tr>
<td>ZONELEN DS FL4</td>
<td></td>
<td>LENGTH OF ZONE LIST</td>
</tr>
<tr>
<td>PENTRY DS AL4</td>
<td></td>
<td>PTR TO LIST OF ENTRIES</td>
</tr>
<tr>
<td>ENTRYLEN DS FL4</td>
<td></td>
<td>LENGTH OF ENTRY LIST</td>
</tr>
<tr>
<td>PSUBENTRY DS AL4</td>
<td></td>
<td>PTR TO LIST OF SUBENTRIES</td>
</tr>
<tr>
<td>SUBENTRYLEN DS FL4</td>
<td></td>
<td>LENGTH OF SUBENTRY LIST</td>
</tr>
<tr>
<td>PFILTER DS AL4</td>
<td></td>
<td>PTR TO QUERY FILTER</td>
</tr>
<tr>
<td>FILTERLEN DS FL4</td>
<td></td>
<td>LENGTH OF FILTER</td>
</tr>
</tbody>
</table>

ENTRY_LIST

<table>
<thead>
<tr>
<th>STRUCTURE</th>
<th>TYPE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENTRY_LIST DSECT</td>
<td></td>
<td>LIST OF ENTRY TYPES</td>
</tr>
<tr>
<td>NEXT DS AL4</td>
<td></td>
<td>PTR TO NEXT ITEM IN LINKED LIST</td>
</tr>
<tr>
<td>ENTRIES DS AL4</td>
<td></td>
<td>PTR TO HEAD OF LINKED LIST OF CSIENTRY STRUCTURES LISTING THE INSTANCES OF THIS ENTRY TYPE</td>
</tr>
<tr>
<td>TYPE DS CL12</td>
<td></td>
<td>ENTRY TYPE</td>
</tr>
</tbody>
</table>

ENTRY

<table>
<thead>
<tr>
<th>STRUCTURE</th>
<th>TYPE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSIENTRY DSECT</td>
<td></td>
<td>DESCRIPTION OF A SPECIFIC ENTRY</td>
</tr>
<tr>
<td>CSINEXT DS AL4</td>
<td></td>
<td>PTR TO NEXT ITEM IN LINKED LIST</td>
</tr>
<tr>
<td>SUBENTRIES DS AL4</td>
<td></td>
<td>PTR TO HEAD OF LINKED LIST OF SUBENTRY STRUCTURES LISTING THE SUBENTRY VALUES RETURNED FOR THIS ENTRY</td>
</tr>
<tr>
<td>ENTRYNAME DS CL8</td>
<td></td>
<td>NAME OF ENTRY</td>
</tr>
<tr>
<td>ZONENAME DS CL7</td>
<td></td>
<td>ZONE WHERE ENTRY WAS RETRIEVED</td>
</tr>
</tbody>
</table>

SUBENTRY

<table>
<thead>
<tr>
<th>STRUCTURE</th>
<th>TYPE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUBENTRY DSECT</td>
<td></td>
<td>CONTAINS SUBENTRY DATA</td>
</tr>
<tr>
<td>SUBNEXT DS AL4</td>
<td></td>
<td>PTR TO NEXT ITEM IN LINKED LIST</td>
</tr>
<tr>
<td>SUBENTDATA DS AL4</td>
<td></td>
<td>PTR TO HEAD OF LINKED LIST OF DATA VALUES FOR THIS SUBENTRY</td>
</tr>
<tr>
<td>SUBTYPE DS CL12</td>
<td></td>
<td>SUBENTRY TYPE</td>
</tr>
</tbody>
</table>
GIMAPI

Additional programming considerations

GIMAPI is installed in SYS1.LINKLIB and SYS1.MIGLIB with the GIMSMP module. Because GIMAPI resides in LINKLIB, which is considered an authorized library, GIMAPI is available to both authorized and unauthorized callers.

The GIMAPI load module cannot be reused. A new copy must be brought into virtual storage for each use. Each task must load its own copy of GIMAPI. However, a single task can load GIMAPI once and then call GIMAPI several times before deleting GIMAPI.

GIMAPI runs in AMODE 31. It will obtain storage from either above or below the 16MB line, based on the AMODE of the caller upon entry to GIMAPI.

GIMAPI will not provide an ESTAE environment. If an ESTAE environment is required, the calling program must establish it before invoking GIMAPI.

Callers of GIMAPI should not link-edit a copy of the load module into the application program. GIMAPI should be called as an external routine.

The caller of GIMAPI must first load the GIMAPI load module into virtual storage using the appropriate syntax for the language in which the application is written. For example, an assembler application may use the LOAD or LINK macros. Once the GIMAPI module is loaded into virtual storage, it can be called by the application.

The caller of GIMAPI must adhere to standard linkage conventions. For more information regarding standard linkage conventions, refer to "z/OS MVS Programming: Assembler Services Guide" and the appropriate documentation for the language you are using for your application.

GIMAPI can only be invoked from a user-written program. GIMAPI cannot be invoked from JCL.

GIMAPI exposes certain control blocks to the user. These are the data structures defined in section "Data structures for QUERY command" on page 390. A macro...
The data set is provided for you to include the definition of the control blocks in your program. The DDDEF of the macro library is MACLIB for the target library and AMACLIB for the distribution library. Refer to “Programming in C” on page 396, “Programming in PL/I” on page 398, and “Programming in assembler” on page 400 for the member names used for each language.

### Sample programs that use GIMAPI

This section contains three sample programs that use GIMAPI. Each program does the same basic function, where differences are mostly due to the syntax of the languages involved.

**Note:** In addition to the programs documented here, SYS1.SAMPLIB provides other sample programs that use GIMAPI, such as GIMCRSAM and GIMPRSAM.

Each program follows these steps:

1. The query described in “Example of QUERY command” on page 388 is set up to be invoked. The global CSI data set used is 'SMP:VSAM.CSI'.
2. GIMAPI is called with the QUERY command to retrieve the data.
3. If the query is successful, it calls a generic result print function that prints the data retrieved from any query.
4. Once these are done, the storage is freed by calling GIMAPI with the FREE command.

The following is an example of the output of the sample programs:

**Entry Type: SYSMOD**

```
ENAME  : SMOD19  
ZONE   : TARG1
INSTALLDATE : 07340
MOD    : MOD01
        MOD02

ENAME  : SMOD22
ZONE   : MYTARG
INSTALLDATE : 07348
```

### Sample C/370 program

This sample program can be found in the SAMPLIB library as member name GIMCSAMP.

```c
#include <stdio.h>
#include <stdlib.h>
#include <DD:SYSLIB(GIMHC370)> /* Contains API structure definitions */

#define FREE "FREE "
#define QUERY "QUERY 
#define APILANG "ENU"
#define TXT_VER "VER"
#define LEN_ETYPE 12
#define LEN_ENAME 8
#define LEN_ZNAME 7
#define LEN_VERNUM 3
#define LEN_TXTVER 3
#define LEN_MSG 256

typedef void APIPGM();
typedef void cfunc();
#pragma linkage(APIPGM,OS)
```

Chapter 6. SMP/E CSI application programming interface  403
static void errprint(char *, long, long, ITEM_LIST *);
static void valprint(ITEM_LIST *);
static void resprint(ENTRY_LIST *);

void main(int argc, char *argv[]) {
  long rc, cc;
  QUERY_PARMQS qparms;
  P_QUERY_PARMQS pparms = &qparms;
  ENTRY_LIST *qreslt;
  ITEM_LIST *msgbuff;
  APIPGM *gimapi;
  char csi[45];
  char zone[100];
  char ent[100];
  char subent[100];
  char filter[150];
  rc = 0;
  cc = 0;

  /******************************************************************************
  /* Load the GIMAPI load module for use later */
  /******************************************************************************
  gimapi = (APIPGM *) fetch("GIMAPI");

  /******************************************************************************
  /* Create the QUERY. Put the parameter strings into */
  /* variables and put the addresses of those variables */
  /* in the query parameter structure along with the */
  /* length of those strings. */
  /******************************************************************************
  strcpy(csi,"SMP.VSAM.CSI");
  strcpy(zone,"ALLZONES");
  strcpy(ent,"SYSMOD");
  strcpy(subent,"MOD,INSTALLDATE");
  strcpy(filter,"(SMODTYPE='PTF' | SMODTYPE='USERMOD') & FMID='HMP1E00' & APPLY='YES' & BYPASS='YES' & RECDATE>'07335'");
  qparms.csi = csi;
  qparms.csilen = strlen(csi);
  qparms.zone = zone;
  qparms.zonelen = strlen(zone);
  qparms.entrytype = ent;
  qparms.entrylen = strlen(ent);
  qparms.subentrytype = subent;
  qparms.subentrylen = strlen(subent);
  qparms.filter = filter;
  qparms.filterlen = strlen(filter);
  gimapi(QUERY,&pparms,(void**)&qreslt,APILANG,&rc,&cc,&msgbuff);
  if (rc!=0)
    { errprint(QUERY, rc, cc, msgbuff);
      if (rc>4) goto EXIT;
    }

  /******************************************************************************
  /* Call routine to print results of query */
  /******************************************************************************
  resprint(qreslt);

  /******************************************************************************
  /* Free storage returned from the QUERY */
  /******************************************************************************
}
EXIT:

    release ((cfunc*)gimapi);
}

/**************************************************************************
/* Print results of the query */
**************************************************************************/
static void resprint(ENTRY_LIST *head)
{
    ENTRY_LIST *curetype;
    CSI_ENTRY  *curentry;
    SUBENTRY   *cursubent;
    VER         *curver;
    SUBENTRY   *curversub;
    char etype[13];
    char vernumber[13];
    char versubtype[13];
    char stEname[LEN_ENAME+1];
    char stZname[LEN_ZNAME+1];

    /**************************************************************************
    /* Loop through each entry type */
    /**************************************************************************/
    for (curetype=head; curetype!=0 ; curetype=curetype->next)
    {
        /**************************************************************************
        /* Print name of entry type being processed */
        /**************************************************************************/
        strncpy(etype,curetype->type,LEN_ETYPE);
        etype[LEN_ETYPE] = '\0';
        printf("Entry Type: %s\n",etype);

        /**************************************************************************
        /* Loop through each entry printing the ename and zone */
        /* then the list of subentry values. */
        /**************************************************************************/
        for (curentry=curetype->entries;
            curentry!=0;
            curentry=curentry->next)
        {
            printf("--------------------------\n");
            strncpy(stEname,curentry->entryname,LEN_ENAME);
            stEname[LEN_ENAME]='\0';
            strncpy(stZname,curentry->zonename,LEN_ZNAME);
            stZname[LEN_ZNAME]='\0';
            printf(" ENAME : %s\n",stEname);
            printf(" ZONE : %s\n",stZname);
            for (cursubent=curentry->subentries;
                cursubent!=0;
                cursubent=cursubent->next)
            {
                strncpy(etype,cursubent->type,LEN_ETYPE);
                etype[LEN_ETYPE] = '\0';
                if ((strncmp(etype,TXT_VER,LEN_TXTVER)) == 0)
                {
                    for (curver=(P_VER) cursubent->subentrydata;
                        curver!=0;
                        curver=curver->next)
                    {
                        strncpy(vernumber,curver->vernum,LEN_VERNUM);
                        vernumber[LEN_VERNUM]='\0';
                        for (curversub=curver->verdata;
                            curversub!=0;
                            curversub=curversub->next)
                    {
curversub=curversub->next)
{
    /**************************************************************/
    /* Now print ver subentry values */
    /**************************************************************/
    strncpy(versubtype,curversub->type,LEN_ETYPE);
    versubtype[LEN_ETYPE]=\0;  
    printf(" %.6s VER(\%s) ",versubtype,vernumber);
    valprint(curversub->subentrydata);
}
} /* end ver subentry */
else /* not a ver structure */
{
    printf(" %s : ",etype);
    valprint(cursubent->subentrydata);
} /* end non-ver subentry */
} /* end subentry for */
} /* end entries for */
} /* end entry type for */
}
} /* end for */

static void valprint(ITEM_LIST *item1)
{
    char databuff[500];
    ITEM_LIST *curitem;

    for (curitem=item1;
        curitem!=0;
        curitem=curitem->next)
    {
        strncpy(databuff,curitem->data,curitem->datalen);
        databuff[curitem->datalen] = \0;
        printf("%s
",databuff);
        if (curitem->next!=0)
            printf("\n ");
    } /* end item for */
} /* end item for */

static void errprint(char *cmd, long rc, long cc, ITEM_LIST *msgs)
{
    char msgout[LEN_MSG+1];
    ITEM_LIST *curmsg;
    unsigned short i;

    printf("Error processing command: %s. RC=%ld CC=%ld\n", 
        cmd,rc,cc);  
    printf("Messages follow:\n");  
    /**************************************************************/
    /* Loop through a linked list of error messages */
    /**************************************************************/
    for (curmsg=msgs; curmsg!=0; curmsg=curmsg->next)
    {
        strncpy(msgout,curmsg->data,curmsg->datalen);
        msgout[curmsg->datalen]=\0;
        printf("%s\n",msgout);
    } /* end item for */
}

---

**Sample PL/I program**

This sample program can be found in the SAMPLIB library as member name GIMPSAMP.
MAIN:
    PROC OPTIONS(MAIN) REORDER;

%INCLUDE GIMMPI;

DCL QUERY CHAR(8) INIT('QUERY');
DCL FREE CHAR(8) INIT('FREE');
DCL TXT_VER CHAR(12) INIT('VER');
DCL APILANG CHAR(3) INIT('ENU');

DCL GIMAPI ENTRY(CHAR(8),PTR,PTR,CHAR(3),FIXED BIN(31),
    FIXED BIN(31),PTR)
    EXTERNAL OPTIONS(ASSEMBLER,INTER);

DCL SYSCALL BUILTIN;
DCL ADDR BUILTIN;
DCL SUBSTR BUILTIN;
DCL (RC,CC) FIXED BIN(31) INIT(0);
DCL QPARMS POINTER;
DCL MSGBUFF POINTER;
DCL QRESULT POINTER;
DCL NULLPTR POINTER;
DCL CSISTR CHAR(44);
DCL ZONESTR CHAR(100);
DCL ENTRYSTR CHAR(100);
DCL SUBENTSTR CHAR(100);
DCL FILTERSTR CHAR(150);

/********************************************************************/
/**/  
/* SET ADDRESS OF QUERY PARAMETERS TO THE QPARMS VAR */
/**/  
/********************************************************************/

QPARMS = ADDR(QUERY_PARMS);

/********************************************************************/
/**/  
/* LOAD GIMAPI LOAD MODULE */
/**/  
/********************************************************************/

FETCH GIMAPI;

/********************************************************************/
/**/  
/* PLUG QUERY PARMS INTO THE QUERY STRUCTURE */
/**/  
/********************************************************************/

CSISTR = 'SMP.VSAM.CSI';
ZONESTR = 'ALLZONES';
ENTRYSTR = 'SYMSMOD';
SUBENTSTR = 'MOD, INSTALLDATE';
SUBSTR(FILTERSTR,1,41) = 'SMODTYPE='PTF' | SMODTYPE='USERMOD''';
SUBSTR(FILTERSTR,42,19) = ' & FMID='HMP1E00''';
SUBSTR(FILTERSTR,61,16) = ' & APPLY='YES''';
SUBSTR(FILTERSTR,77,37) = ' & BYPASS='YES' & RECDATE='07335''';

CSI = ADDR(CSISTR);
CSILEN = 12;
ZONE = ADDR(ZONESTR);
ZONELEN = 9;
ENTRYTYPE = ADDR(ENTRYSTR);
ENTRYLEN = 6;
GIMAPI

SUBENTRYTYPE = ADDR(SUBENTSTR);
SUBENTRYLEN = 16;
FILTER = ADDR(FILTERSTR);
FILTERLEN = 113;

CALL GIMAPI(QUERY,QPARMS,QRESULT,APILANG,RC,CC,MSGBUFF);

/******************************************************************************
/* PRINT ANY ERROR MESSAGES ENCOUNTERED */
******************************************************************************
IF RC ≠0
   THEN CALL ERRPRINT(QUERY);

******************************************************************************
/* CALL ROUTINE TO PRINT RESULTS OF QUERY IF QUERY WAS SUCCESSFUL */
******************************************************************************
IF RC≤4
   THEN CALL RESPRINT;

******************************************************************************
/* FREE STORAGE RETURNED FROM THE QUERY */
******************************************************************************
CALL GIMAPI(FREE,NULLPTR,NULLPTR,APILANG,RC,CC,MSGBUFF);

EXIT:
   RELEASE GIMAPI;

******************************************************************************
/* INTERNAL SUBROUTINES FOLLOW */
******************************************************************************

/* LOOP THROUGH EACH ENTRY TYPE */
CURETYPE = QRESULT; /* POINT TO HEAD OF LIST */
DO WHILE (CURETYPE¬=SYSNULL);
   /* PRINT NAME OF ENTRY BEING PROCESSED */
   PUT EDIT('Entry Type: ',CURETYPE->ENTRY_LIST.TYPE)
      (SKIP,A(12),A(15));
   /* LOOP THROUGH EACH ENTRY PRINTING THE ENAME AND ZONE */
   /* THEN THE LIST OF SUBENTRY VALUES. */
   CURENTRY = CURETYPE->ENTRIES;
   DO WHILE (CURENTRY¬=SYSNULL);
      PUT SKIP LIST('----------------------------------------');
      PUT EDIT('ENAME',':',CURENTRY->CSI_ENTRY.ENTRYNAME)
         (SKIP,X(2),A(5),X(10),A(1),X(1),A(8));
      PUT EDIT('ZONE',':',CURENTRY->CSI_ENTRY.ZONENAME)
         (SKIP,X(2),A(4),X(11),A(1),X(1),A(7));
      CURSUBENT=CURENTRY->SUBENTRIES;
      DO WHILE (CURSUBENT¬=SYSNULL);
      IF CURSUBENT->SUBENTRY.TYPE=TXT_VER THEN DO;
      CURVER=CURSUBENT->SUBENTRYDATA;
      DO WHILE (CURVER¬=SYSNULL);
      CURVSUB=CURVER->VERDATA;
      DO WHILE (CURVSUB¬=SYSNULL);
      PUT EDIT(CURVSUB->SUBENTRY.TYPE,'VER(',GIMAPI,')
         (SKIP,A(12),A(15));
      EXIT:
      RELEASE GIMAPI;

408   SMP/E V3R6.0 for z/OS V1R13.0 Reference
CURVER->VERNUM,'):')
  (SKIP,X(2),A(6),X(1),A(4),A(3),A(2),X(1));
PRTITEM=CURVSUB->SUBENTRYDATA;
CALL VALPRINT(PRTITEM);
CURVSUB=CURVSUB->SUBENTRY.NEXT;
END;
CURVER=CURVER->VER.NEXT;
END; /* End Process VER type subentries */
ELSE DO;
  PUT EDIT(CURSUBENT->SUBENTRY.TYPE,':')
    (SKIP,X(2),A(15),A(1),X(1));
PRTITEM=CURSUBENT->SUBENTRYDATA;
CALL VALPRINT(PRTITEM);
END; /* End non-VER type subentries */

CURSUBENT = CURSUBENT->SUBENTRY.NEXT;
END; /* END SUBENT TYPE LOOP */
CURENTRY = CURENTRY->CSI_ENTRY.NEXT; /* GET NEXT ENTRY */
END; /* END ENTRY LOOP */
PUT SKIP;
CURENTRY = CURENTRY->CSI_ENTRY.NEXT; /* GET NEXT ENTRY */
END; /* END ENTRY TYPE LOOP */
END RESPRINT;

VALPRINT: PROCEDURE(ITEM1);
DCL ITEM1 POINTER;
DCL CURITEM POINTER;
DCL BUFFPTR POINTER;
DCL DATABUFF CHAR(500) BASED(BUFFPTR);
CURITEM = ITEM1;
DO WHILE (CURITEM ¬=SYSNULL);
  BUFFPTR = CURITEM->DATA;
  PUT EDIT(SUBSTR(BUFFPTR->DATABUFF,1,CURITEM->DATALEN))
    (X(1),A);
  CURITEM = CURITEM->ITEM_LIST.NEXT; /* GET NEXT DATA VALUE */
  IF CURITEM¬=SYSNULL /* LINE UP NEXT VALUE IF THERE IS ONE */
    THEN PUT SKIP LIST('');
END; /* END DATA ITEM LOOP */
END VALPRINT;

ERRPRINT: PROCEDURE(CMD);
DCL CMD CHAR(8);
DCL CURMSG POINTER;
DCL TEXTPTR POINTER;
DCL MSGTEXT CHAR(256) BASED(TEXTPTR);
PUT EDIT('Error processing command: ',CMD,'. ','RC=',RC,'CC=',CC)
  (SKIP,A(26),A(8),A(3),A(3),F(5),X(2),A(3),F(5));
IF MSGBUFF¬=SYSNULL THEN
  DO;
    PUT SKIP LIST('MESSAGES FOLLOW:');
    CURMSG = MSGBUFF;
    DO WHILE (CURMSG¬=SYSNULL);
      TEXTPTR = CURMSG->DATA;
      PUT SKIP LIST(SUBSTR(TEXTPTR->MSGTEXT,1,CURMSG->DATALEN));
      CURMSG = CURMSG->ITEM_LIST.NEXT;
    END;
    END;
END; /* END DATA ITEM LOOP */
END ERRPRINT;
Sample assembler program

This sample program can be found in the SAMPLIB library as member name GIMASAMP.

```
MAIN:
BALAPI CSECT
STM 14,12,12(13)
LR 12,15
@PSTART EQU BALAPI
USING @PSTART,12
ST 13,SAVE+4
LA 14,SAVE
ST 14,B(13)
LR 13,14
*
*
* END OF STANDARD LINKAGE
*
* BALAPI AMODE 31
*  
* SET UP PARAMETERS FOR QUERY
*  
  XC QUERY_PARMS,QUERY_PARMS
LA 3,MYCSI
ST 3,PCSI
LA 3,19
ST 3,CSILEN
LA 3,MYZONE
ST 3,PZONE
LA 3,3
ST 3,ZONELEN
LA 3,MYENTRY
ST 3,PENTRY
LA 3,6
ST 3,ENTRYLEN
LA 3,MYSUBNTY
ST 3,PSUBENTRY
LA 3,1
ST 3,SUBENTRYLEN
LA 3,MYFILTER
ST 3,PFILTER
LA 3,16
ST 3,FILTERLEN
*
*
* NOW LOAD THE API
*  
  LA 2,APIPGM
  LOAD EPLOC=(2),LOADPT=PGMADR
*  
* NOW DO THE QUERY
*  
  L 15,PGMADR
  CALL (15),(QUERYCMD,QUERY_PARMS@,CMDOUT,APILANG,RC,CC,MSG@)
*  
*
* NOW SEE WHAT WAS RETURNED
*  
  L 3,RC
  LTR 3,3
  BNZ ERRPRINT
*  
* ESTABLISH ADDRESSABILITY
```
* OPEN (SYSPRINT,OUTPUT)
  L  3,CMDOUT
  USING ENTRY_LIST,3
  L  4,ENTRIES
  USING CSIENTRY,4
  L  5,SUBENTRIES
  USING SUBENTRY,5
  L  6,SUBENTDATA
  USING ITEM_LIST,6
  L  7,DATA
  USING RETDATA,7

* PRINT ENTRIES

* PRXENT LTR 4,4
  BZ CLOSEOUT
  MVI BUFFER,C' '
  MVC BUFFER+1(119),BUFFER
  MVC BUFFER(8),ENTRYNAME
  MVC BUFFER+10(7),ZONENAME
  LA 2,ANSICHAR
  PUT SYSPRINT,(2)
  B PRXSUB

MOREENT L 4,CSINEXT
  B PRXENT

* PRINT SUBENTRIES

* PRXSUB LTR 5,5
  BZ MOREENT
  MVI BUFFER,C' '
  MVC BUFFER+1(119),BUFFER
  MVC BUFFER(11),SUBTYPE
  PUT SYSPRINT,(2)
  B PRXDATA

MORESUB L 5,SUBNEXT
  B PRXSUB

* PRINT DATA

* PRXDATA LTR 7,7
  BZ MORESUB
  L 8,DATALOG
  CHECKLEN C 8,OUTRECLN
  BNH SETLEN
  L 8,OUTRECLN
  SETLEN LA 15,1
  SLR 8,15
  MVI BUFFER,C' '
  MVC BUFFER+1(119),BUFFER
  EX 8,MOVDATA
  LA 2,ANSICHAR
  PUT SYSPRINT,(2)
  L 8,DATALOG
  C 8,OUTRECLN
  BNH MORESUB
  S 8,OUTRECLN
  ST 8,DATALOG
  A 7,OUTRECLN
  B CHECKLEN

* PRINT ERROR MESSAGES

* ERRPRINT OPEN (SYSPRINT,OUTPUT)
  L 6,MSG@
  L 7,DATA
GIMAPI

L 8,DATALEN
CHKMSGLN C 8,OUTRECLN
BNH SETMSGLN
L 8,OUTRECLN
SETMSGLN LA 15,1
SLR 8,15
MVI BUFFER,C' '
MVC BUFFER+1(119),BUFFER
EX 8,@MOVDATA
LA 2,ANSICHAR
PUT SYSPRINT,(2)
L 8,DATALEN
C 8,OUTRECLN
BNH CLOSEOUT
S 8,OUTRECLN
ST 8,DATALEN
A 4,OUTRECLN
B CHKMSGLN

* CLOSEOUT CLOSE SYSPRINT
* * NOW FREE THE STORAGE OBTAINED DURING THE QUERY
* * CLEANUP L 15,PGMADR
CALL (15),(FREECMD,0,CMDOUT,APILANG,RC,CC,MSG@)
* * NOW DELETE GIMAPI
* * DELETE EPLOC=APIPGM
* * EXIT
* * EXIT LA 15,0
L 13,4(,13)
L 14,12(,13)
LM 00,12,20(13)
BR 14
*
*
@MOVDATA MVC BUFFER(0),RETDATA
SYSPRINT DCB DDNAME=SYSPRINT,DSORG=PS,MACRF=PM,LRECL=121,RECFM=FBA
ANSICHAR DC CL1' '
BUFFER DS CL120
MYCSI DC CL19'TOPGUN.WAG.VSAM.CSI'
MYZONE DC CL3'TZ1'
MYENTRY DC CL6'SYSMOD'
MYFILTER DC CL16'SMODTYPE=''PTF'''
MYSUBNTY DC CL3'x'
APIPGM DC CL8'GIMAPI '
QUERYCMD DC CL8'QUERY '
FREECMD DC CL8'FREE '
APILANG DC CL3'ENU'
DS OF
QUERY_PARM0 DC AL4(QUERY_PARM0)
PGMADR DC AL4(0)
CMDOUT DC AL4(0)
MSG0 DC AL4(0)
OUTRECLN DC F'120'
RC DS F'0'
CC DS F'0'
SAVE DC 18F'0'
GIMMASM
RETDATA DSECT CL0
END
Chapter 7. Writing UNIX shell scripts

This chapter documents intended Programming Interfaces that allow the customer to write programs to obtain the services of IBM SMP/E for z/OS, V3R6.

To simplify the post-install work for z/OS UNIX Services application programs, some UNIX applications include shell scripts. These scripts perform additional processing when SMP/E installs elements into a UNIX file system. A product packager normally includes any necessary shell scripts with the product.

For example, if the hierarchical file system element is a TAR or PAX file, you can provide a shell script that performs the necessary steps to restore the file. As with other products, you use SMP/E to copy the element (a TAR or PAX file) to a directory in a UNIX file system. However, you rely on the element's shell script to actually explode the file into its component subdirectories and files.

SMP/E provides IBM and vendor product packagers with a generic interface for writing UNIX shell scripts. This chapter describes the interface and includes suggestions for designing a shell script for SMP/E processing.

You define the shell script to SMP/E through a hierarchical file system MCS statement, as described in "Hierarchical file system element MCS" on page 26. Once defined, the shell script receives control whenever SMP/E installs or deletes the element.

Designing a shell script for SMP/E processing

To process a file in a UNIX file system, a shell script must be able to cope with both of the actions that SMP/E can potentially perform on the file: copy and delete. That is, SMP/E can copy the file to a directory in a UNIX file system (as a new file or a replacement for an existing file) and, later, SMP/E can delete the file from a UNIX file system directory. A shell script must be able to detect either of these conditions (copy or delete) and respond accordingly.

Assume, for example, that as part of deleting or replacing a product on your system, you delete a function that was shipped in the product's tar file. SMP/E deletes only the original tar file from the directory in a UNIX file system. It is the responsibility of the shell script to clean up (delete) the tar file's exploded component subdirectories and files.

Fortunately, SMP/E provides shell scripts with the necessary input. This input comes in the form of environment variables that SMP/E sets, as follows:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMP_Directory</td>
<td>Directory in a UNIX file system that contains the file to be processed by your shell script. This directory should be considered the working directory for the shell script, and the shell script should not update any directories or files which do not reside within the working directory.</td>
</tr>
</tbody>
</table>
### UNIX shell scripts

| SMP_File | Name of the file to be processed by your shell script. This is the hierarchical file system element processed by SMP/E. |
| SMP_Action | Action that SMP/E is performing on the file. This value can be either of the following: |
| SMP_Phase | Point in processing in which SMP/E calls the shell script. This value can be either of the following: |
| LC_ALL | The default value for all of the UNIX locale environment variables. This value is always set to IBM-1047. |
| PATH | The Java runtime directory concatenated with the base PATH value (/bin:). The Java runtime directory is obtained from either the SMPJHOME DD statement or the SMPJHOME DDDEF entry. The Java runtime directory is required only if the UNIX shell script invokes Java commands. |

**COPY**  
SMP/E is copying the file into a UNIX file system directory.

**DELETE**  
SMP/E is deleting the file from a UNIX file system directory.

**PRE**  
SMP/E is calling the shell script before performing the action indicated by the SMP_Action variable.

**POST**  
SMP/E is calling the shell script after performing the action indicated by the SMP_Action variable.

SMP_Phase is always set to PRE when SMP_Action is DELETE to ensure that SMP/E calls the shell script before deleting its corresponding element.

**IBM-1047**  
Defines the United States English character set. By specifying a specific locale, all shell scripts invoked by SMP/E receive the same environment, regardless of the locale being used on any particular driving system.

**This input allows you to design a shell script that can respond to SMP/E’s actions, as described in the sections that follow.**
Designing for copy actions

For a copy action, SMP/E sets the SMP_Action variable to COPY (as well as setting the other environmental variables) and passes control to the shell script.

The shell script should not assume a first-time environment for the file. For example, if APPLY REDO is in effect, previous copy processing for a file might have already completed successfully. Therefore the shell script should begin by performing some kind of clean-up processing before taking any copy-related actions.

Designing for delete actions

For a delete action, SMP/E sets the SMP_Action variable to DELETE (as well as setting the other environmental variables) and passes control to the shell script.

The shell script should not assume that the file has previously been installed. For example, if during an APPLY, a file is being installed for the first time, and is also being deleted by a subsequent SYSMOD, SMP/E might invoke the shell script to delete the file before it has actually been copied to a UNIX file system. Therefore, the shell script should determine whether the file exists before taking any delete-related actions.

Designing for diagnosis

Code your shell script to write status information to the STDOUT and STDERR files during the course of its processing. Doing so will aid users in verifying the successful completion of shell script processing, and, if necessary, diagnosing problems when failures occur.

SMP/E copies this information from the STDOUT and STDERR files to the print data set specified in the active HFSCOPY UTILITY entry where it can be viewed by the user. By default, SYSPRINT is the print data set.

It is recommended that shell scripts provide sufficient information about the completion of any functions they use, and especially about any detected error conditions.

Returning control to SMP/E

When your shell script completes processing, it must return control to SMP/E. Your shell script returns control to SMP/E through the exit shell command, specifying an appropriate exit status (for example, exit 0 or exit –1). SMP/E uses your shell script’s exit status value to determine whether the shell script completed successfully.

SMP/E recognizes the following exit statuses from shell scripts:

<table>
<thead>
<tr>
<th>Exit status</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Shell script has completed successfully.</td>
</tr>
<tr>
<td>Any other value</td>
<td>Shell script processing has failed.</td>
</tr>
</tbody>
</table>

Example shell script

In the following example, the shell script is designed to process a UNIX tar file. The shell script is written so that it:

1. Explodes the tar file in response to a COPY action by SMP/E
2. Deletes all files from the directory in response to a DELETE action by SMP/E.
UNIX shell scripts
# This script will either explode a tar file file or delete all files in 
# the directory. The script uses the following environment variables 
# for input:
# # SMP_Directory - directory in which the tar file resides
# # SMP_File - name of the tar file
# # SMP_Phase - indicates whether the shell script is being called 
# before or after SMP/E has processed the file
# # SMP_Action - the action that SMP/E is performing: COPY or DELETE
# echo "Starting script processing..."
# status=0 # initialize script status
# # Verify that the required input was received by the shell script.
# if test ! "$SMP_Directory"
# then
# echo "** No SMP_Directory parameter specified." 
# status=-1
# else
# if test ! "$SMP_File"
# then
# echo "** No SMP_File parameter specified." 
# status=-1
# else
# if test ! "$SMP_Phase"
# then
# echo "** No SMP_Phase parameter specified." 
# status=-1
# else
# if test ! "$SMP_Action"
# then
# echo "** No SMP_Action parameter specified." 
# status=-1
# fi
# fi
# # If a parameter error was detected, then exit now.
# if test $status -ne 0 # If status is not 0, an error was detected 
# then
# echo " Ensure the input environment variables for this script have been provided." 
# echo " If SMP/E was not used to invoke the script, correct the caller to specify 
# the input environment variables. If SMP/E invoked this script, contact " 
# echo " the IBM support center." 
# echo "Parameter error. Exiting script with status $status"
# exit $status
# fi
# # If the script is being invoked after the tar file has been copied 
# # to the directory (SMP/E phase is post-copy), and the desired action 
# # is COPY, explode the tar file.
# # if test $SMP_Phase = POST 
# then
# if test $SMP_Action = COPY 
# then
# cd $SMP_Directory # Set the working directory for pax
# echo "Explooding all components of $SMP_Directory$SMP_File using the pax command"
# pax -rvf $SMP_Directory$SMP_File # Explode the tar file
# status=$? # Get the status of the pax command
# if test $status -ne 0 # If pax failed, indicate so to the user 
# then
# echo "** pax command failure: pax ended with status $status"
# else
# echo "pax command completed successfully."
# fi
# ls -l $SMP_Directory # List the contents of the directory
# fi
# fi
# # If the action is DELETE, delete all component files.
# if test $SMP_Action = DELETE 
# then
# echo "Deleting the following from $SMP_Directory using the rm command:"
# ls -l $SMP_Directory # List the contents of the directory
# fi
Chapter 8. Library change file records

This chapter documents intended Programming Interfaces that allow the customer to write programs to obtain the services of IBM SMP/E for z/OS, V3R6.

This chapter documents the various record types that are produced and written to the SMPDATA1 and SMPDATA2 data sets as a result of APPLY or RESTORE processing. You can use these records to propagate the libraries and members modified by SMP/E APPLY and RESTORE processing to other systems that require the same changes.

Note: The CHANGEFILE subentry in the OPTIONS entry must be set to YES to instruct SMP/E to create these records.

SMP/E provides macros containing the mappings of these records:
• GIMMALC for assembler
• GIMMCLC for C
• GIMMPLC for PL/I.

Library change file record structure

The following sections provide samples of the various library change file records that are written to the SMPDATA1 and SMPDATA2 data sets as a result of APPLY or RESTORE processing.

Notes:
1. SMP/E COMPRESS processing does not create any library change records.
2. The library change file records are of varying lengths.
3. All character data in all library change records are in uppercase, except for the following, which contain mixed-case character data:
   • Aliases in Alias Record Type 0 records
   • Link names in Alias Record Type 0 records
   • Symbolic link names in Alias Record Type 1 records
   • Path names in Element Record Type 1 records
   • Path names in Library Record Type 1 records
   See "A0 - Alias record type 0" on page 420 and "L1 - Library record type 1" on page 429 for details.
4. The scale lines shown in the examples do not appear in the actual SMPDATA1 and SMPDATA2 data sets.
5. In Figure 73 on page 421, the binary zeros after the alias name are not shown in the example.
6. In Figure 74 on page 423, the binary zeros after the symbolic link name are not shown in the example.

Library change file record types

The following are the valid record types produced by the library change interface:

<table>
<thead>
<tr>
<th>Record type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A0</td>
<td>Alias Record Type 0</td>
</tr>
<tr>
<td>A1</td>
<td>Alias Record Type 1</td>
</tr>
<tr>
<td>C0</td>
<td>Continuation Record Type 0</td>
</tr>
</tbody>
</table>
Library change file records

<table>
<thead>
<tr>
<th>E0</th>
<th>Element Record Type 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1</td>
<td>Element Record Type 1</td>
</tr>
<tr>
<td>H0</td>
<td>Header Record Type 0</td>
</tr>
<tr>
<td>L0</td>
<td>Library Record Type 0</td>
</tr>
<tr>
<td>L1</td>
<td>Library Record Type 1</td>
</tr>
<tr>
<td>L2</td>
<td>Library Record Type 2</td>
</tr>
<tr>
<td>P0</td>
<td>SYSMOD Status Record Type 0</td>
</tr>
<tr>
<td>S0</td>
<td>SMP/E Environment Record Type 0</td>
</tr>
<tr>
<td>T0</td>
<td>Trailer Record Type 0</td>
</tr>
</tbody>
</table>

A0 - Alias record type 0

An Alias Record Type 0 (A0) is created for each alias for an element or LMOD processed during APPLY or RESTORE processing. For hierarchical file system elements, the linkname is indicated in the record.

Multiple A0 records may be produced for the same element or LMOD, because more than one alias can exist for the same element or LMOD.

The purpose of the A0 record is to identify the aliases associated with the changed elements or LMODs associated with this execution of the APPLY or RESTORE command.

The format and contents of this record type are shown in Table 40.

<table>
<thead>
<tr>
<th>Field name</th>
<th>Position (decimal)</th>
<th>Length (decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Record type</td>
<td>1</td>
<td>2</td>
<td>The characters 'A0'.</td>
</tr>
<tr>
<td>Name</td>
<td>3</td>
<td>8</td>
<td>The name of an SMP/E element or LMOD processed during APPLY or RESTORE processing. The data is left-justified and padded with blanks.</td>
</tr>
</tbody>
</table>
| Type       | 11                 | 12               | The type of SMP/E element or LMOD processed during APPLY or RESTORE processing. The valid element types for an Alias Record Type 0 are:  
  • Data elements  
  • Hierarchical file system elements  
  • JAR  
  • MAC  
  • LMOD  
  • PROGRAM  
  • SIDEDECK  
  The data is left-justified and padded with blanks. |
| Action     | 23                 | 8                | The type of action SMP/E took against this alias of this element or LMOD during APPLY or RESTORE processing. Valid values are ADDREP and DELETE. This data is left-justified and padded with blanks. |
### Field name | Position (decimal) | Length (decimal) | Description
--- | --- | --- | ---
DD name | 31 | 8 | The ddname associated with the target library associated with the named element or LMOD in this A0 record. For action type DELETE, this field indicates the target library ddname from which the alias has been deleted. For action type ADDREP, this field indicates the target library ddname into which the alias has been added or replaced. An A0 record is created for every unique library that has an alias change. Elements with multiple SYSLIBs will have the appropriate A0 records created for each unique SYSLIB changed. Therefore, it is possible to have the same element have the same change in alias structure in multiple SYSLIBs. The data is left-justified and padded with blanks. This field contains the ddname of the associated target library regardless of whether the allocation was done by a DD statement or DDDEF.

Alias | 39 | 1023 | This is an alias name associated with this SMP/E element or LMOD. If the element is a hierarchical file system element, then the linkname is placed in this field. The data is mixed-case characters and is left-justified and padded with binary zeros.

For LMODs that have an associated side deck, there is one A0 record created for each unique alias associated with the LMOD (indicated by LMOD in the element type field of the A0 record) and one A0 record created for each unique alias associated with the LMOD's side deck (indicated by SIDEDECK in the element type field of the A0 record). Both the A0 record with element type LMOD and the A0 record with element type SIDEDECK contain the same name (the LMOD's name) in the Name field of the A0 record.

For example, suppose that SMP/E adds load module LMODA, which has two aliases, LMA and LMDA. Load module LMODA has a side deck. Figure 73 shows the A0 records created for LMODA:

```
A0LMODA LMOD ADDREP LINKLIB LMA
A0LMODA LMOD ADDREP LINKLIB LMDA
A0LMODA SIDEDECK ADDREP SIDELIB LMA
A0LMODA SIDEDECK ADDREP SIDELIB LMDA
```

**Figure 73. Example of alias record type 0 records**

### A1 - Alias record type 1

An Alias Record Type 1 (A1) is created for each symbolic link associated with a hierarchical file system element and for each symbolic link associated with a load module within a UNIX file system that is processed during APPLY or RESTORE processing. For hierarchical file system elements, the symbolic link names are
placed in the records. For load modules, values from the ALIAS (SYMLINK,symlink) control statements are placed in the records.

Multiple A1 records may be produced for the same hierarchical file system element or load module, because more than one symbolic link can exist for the same hierarchical file system element or load module.

The purpose of the A1 record is to identify the symbolic links associated with the changed hierarchical file system elements and the symbolic links associated with load modules processed with this execution of the APPLY or RESTORE command.

The format and contents of this record type are shown in Table 41.

Table 41. Alias record type 1

<table>
<thead>
<tr>
<th>Field name</th>
<th>Position (decimal)</th>
<th>Length (decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Record type</td>
<td>1</td>
<td>2</td>
<td>The characters 'A1'.</td>
</tr>
<tr>
<td>Name</td>
<td>3</td>
<td>8</td>
<td>The name of an SMP/E hierarchical file system element or load module processed during APPLY or RESTORE processing. The data is left-justified and padded with blanks.</td>
</tr>
</tbody>
</table>
| Type           | 11                 | 12               | The type of SMP/E hierarchical file system element or LMOD processed during APPLY or RESTORE processing. The valid values are:  
|                |                    |                  | • a hierarchical file system element type  
|                |                    |                  | • JAR  
|                |                    |                  | • LMOD  
|                |                    |                  | The data is left-justified and padded with blanks.                         |
| Action         | 23                 | 8                | The type of action SMP/E took against this symbolic link during APPLY or RESTORE processing. Valid values are ADDREP and DELETE. The data is left-justified and padded with blanks. |
| DD name        | 31                 | 8                | The ddname associated with the target library for the named hierarchical file system element or load module in this A1 record. The data is left-justified and padded with blanks.  
|                |                    |                  | For action type DELETE, this field indicates the target library ddname from which the symbolic link has been deleted. |
|                |                    |                  | For action type ADDREP, this field indicates the target library ddname into which the symbolic link has been added or replaced. |
|                |                    |                  | An A1 record is created for all libraries modified within a UNIX file system by having a hierarchical file system element replaced or deleted or by having a load module replaced or deleted. |
|                |                    |                  | This field contains the ddname of the associated target library regardless of whether the allocation was done by a DD statement or DDDEF. |
| Symbolic link  | 39                 | 1023             | This is a symbolic link associated with this SMP/E hierarchical file system element or LMOD. The data is mixed-case characters and is left-justified and padded with binary zeros. |
Library change file records

For example, suppose that load module LMODA, which has two symbolic links, LMA and LMDA, is added to the BPXUSER library. Figure 74 shows the A1 records created for LMODA:

C0 - Continuation record type 0

One Continuation Record Type 0 (C0) is created for each delta produced by the results of APPLY or RESTORE processing when spill processing has occurred. The purpose of the C0 record is to uniquely identify the continuation of a set of library change records.

The format and contents of this record type are shown in Table 42.

<table>
<thead>
<tr>
<th>Field name</th>
<th>Position (decimal)</th>
<th>Length (decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Record type</td>
<td>1</td>
<td>2</td>
<td>The characters 'C0'.</td>
</tr>
<tr>
<td>Target zone</td>
<td>3</td>
<td>7</td>
<td>The target zone name that was operated on by the APPLY or RESTORE command.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>It is left-justified and padded with blanks.</td>
</tr>
<tr>
<td>Time stamp</td>
<td>10</td>
<td>13</td>
<td>The date and time that APPLY or RESTORE processing completed for the entire</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>delta. This field is in the form yyyydddhhmmss, where:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>yyyy  year</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>dddd  day</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>hh    hours</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>mm    minutes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ss    seconds</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Time is represented in 24-hour clock format (military time). The time value</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>matches the time value in the T0 record and is the time that the GIM20501I</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>message is issued for the associated APPLY or RESTORE command.</td>
</tr>
<tr>
<td>ERROR count</td>
<td>23</td>
<td>6</td>
<td>The character representation of the count of SYSMODs applied or restored in</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>error. The data is padded on the left with character zeroes (0).</td>
</tr>
<tr>
<td>INCMPLT count</td>
<td>29</td>
<td>6</td>
<td>The character representation of the count of SYSMODs left with a status of</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>INCMPLT by the APPLY or RESTORE. The data is padded on the left with chara</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>cters zeroes (0).</td>
</tr>
<tr>
<td>APPLIED or RESTORED</td>
<td>35</td>
<td>6</td>
<td>The character representation of the count of SYSMODs with a status of APPLI</td>
</tr>
<tr>
<td>count</td>
<td></td>
<td></td>
<td>ED or RESTORED. The data is padded on the left with character zeroes (0).</td>
</tr>
<tr>
<td>DELETED count</td>
<td>41</td>
<td>6</td>
<td>The character representation of the count of SYSMODs with a status of DELE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>TED. The data is padded on the left with character zeroes (0).</td>
</tr>
<tr>
<td>SUPD count</td>
<td>47</td>
<td>6</td>
<td>The character representation of the count of SYSMODs with a status of SUPD</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The data is padded on the left with character zeroes (0).</td>
</tr>
</tbody>
</table>

Figure 74. Example of alias record type 1

Table 42. Continuation record type 0

<table>
<thead>
<tr>
<th>Field name</th>
<th>Position (decimal)</th>
<th>Length (decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1LMODA</td>
<td>1</td>
<td>2</td>
<td>The characters 'C0'.</td>
</tr>
<tr>
<td>LMOD</td>
<td></td>
<td></td>
<td>The target zone name that was operated on by the APPLY or RESTORE command.</td>
</tr>
<tr>
<td>ADDREP</td>
<td></td>
<td></td>
<td>It is left-justified and padded with blanks.</td>
</tr>
<tr>
<td>BPXUSER LMA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A1LMODA</td>
<td>3</td>
<td>7</td>
<td>The date and time that APPLY or RESTORE processing completed for the entire</td>
</tr>
<tr>
<td>LMOD ADDREP</td>
<td></td>
<td></td>
<td>delta. This field is in the form yyyydddhhmmss, where:</td>
</tr>
<tr>
<td>BPXUSER LMDA</td>
<td></td>
<td></td>
<td>yyyy  year</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>dddd  day</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>hh    hours</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>mm    minutes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ss    seconds</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Time is represented in 24-hour clock format (military time). The time value</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>matches the time value in the T0 record and is the time that the GIM20501I</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>message is issued for the associated APPLY or RESTORE command.</td>
</tr>
<tr>
<td>A1LMODA</td>
<td>10</td>
<td>13</td>
<td>The character representation of the count of SYSMODs applied or restored in</td>
</tr>
<tr>
<td>LMOD ADDREP</td>
<td></td>
<td></td>
<td>error. The data is padded on the left with character zeroes (0).</td>
</tr>
<tr>
<td>BPXUSER LMA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A1LMODA</td>
<td>23</td>
<td>6</td>
<td>The character representation of the count of SYSMODs left with a status of</td>
</tr>
<tr>
<td>LMOD ADDREP</td>
<td></td>
<td></td>
<td>INCMPLT by the APPLY or RESTORE. The data is padded on the left with chara</td>
</tr>
<tr>
<td>BPXUSER LMDA</td>
<td></td>
<td></td>
<td>cters zeroes (0).</td>
</tr>
<tr>
<td>A1LMODA</td>
<td>35</td>
<td>6</td>
<td>The character representation of the count of SYSMODs with a status of APPLI</td>
</tr>
<tr>
<td>LMOD ADDREP</td>
<td></td>
<td></td>
<td>ED or RESTORED. The data is padded on the left with character zeroes (0).</td>
</tr>
<tr>
<td>BPXUSER LMA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A1LMODA</td>
<td>41</td>
<td>6</td>
<td>The character representation of the count of SYSMODs with a status of DELE</td>
</tr>
<tr>
<td>LMOD ADDREP</td>
<td></td>
<td></td>
<td>TED. The data is padded on the left with character zeroes (0).</td>
</tr>
<tr>
<td>BPXUSER LMDA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A1LMODA</td>
<td>47</td>
<td>6</td>
<td>The character representation of the count of SYSMODs with a status of SUPD</td>
</tr>
<tr>
<td>LMOD ADDREP</td>
<td></td>
<td></td>
<td>The data is padded on the left with character zeroes (0).</td>
</tr>
</tbody>
</table>
Library change file records

For example, suppose that a delta is produced for target zone TZONEA on January 16, 2008 at 9:57:22 AM. During this execution of SMP/E, two SYSMODs were applied in error, one SYSMOD was left incomplete, 123 SYSMODs were applied successfully, seven SYSMODs were deleted, and three SYSMODs were superseded. Figure 75 shows the C0 records created that for delta:

```
12345678
----+----0----+----0----+----0----+----0----+----0----+----0----+----0----+----0
C0TZONEA 20080160957220000200001000123000007000003
```

Figure 75. Example of continuation record type 0

### E0 - Element record type 0

An Element Record Type 0 (E0) is created for each element or LMOD that changed in a target library during APPLY or RESTORE processing. The term changed in this context refers to an element or LMOD that been deleted or replaced in a target library as a result of SMP/E processing. See “Valid action types” on page 435 for more information.

The purpose of the E0 record is to identify the changed elements or LMODs in the associated target libraries identified by the L0 and L1 records.

The format and contents of this record type are shown in Table 43.

<table>
<thead>
<tr>
<th>Field name</th>
<th>Position (decimal)</th>
<th>Length (decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Record type</td>
<td>1</td>
<td>2</td>
<td>The characters ‘E0’.</td>
</tr>
<tr>
<td>Name</td>
<td>3</td>
<td>8</td>
<td>The name of an SMP/E element or LMOD processed during APPLY or RESTORE processing. The data is left-justified and padded with blanks.</td>
</tr>
<tr>
<td>Type</td>
<td>11</td>
<td>12</td>
<td>The type of SMP/E element or LMOD processed during APPLY or RESTORE processing. The valid element types for an Element Record Type 0 are: data elements, hierarchical file system elements, JAR, LMOD, MAC, PROGRAM, SIDEDECK, SRC. The data is left-justified and padded with blanks.</td>
</tr>
<tr>
<td>Action</td>
<td>23</td>
<td>8</td>
<td>The type of action SMP/E took against this element or LMOD during APPLY or RESTORE processing. Valid values are ADDREP and DELETE. This data is left-justified and padded with blanks.</td>
</tr>
</tbody>
</table>
Table 43. Element record type 0 (continued)

<table>
<thead>
<tr>
<th>Field name</th>
<th>Position (decimal)</th>
<th>Length (decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DD name</td>
<td>31</td>
<td>8</td>
<td>The ddname associated with the target library associated with the named element or LMOD in this E0 record. The data is left-justified and padded with blanks. For cross-zone elements, this field indicates the SMP/E generated ddname associated with the SYSLIB for this element. The associated L0 record also contains the related true name for the SYSLIB in the cross-zone. For action type DELETE, this field indicates the target library ddname from which the element has been deleted. For action type ADDREP, this field indicates the target library ddname into which the element has been added or replaced. An E0 record is created for every unique library that has an element change. Elements with multiple SYSLIBs will have the appropriate E0 records created for each unique SYSLIB changed. Therefore, it is possible to have the same element have the same change take place in multiple SYSLIBs. This field contains the ddname of the associated target library regardless of whether the allocation was done by a DD statement or DDDEF.</td>
</tr>
</tbody>
</table>

For LMODs that have an associated side deck, there is one E0 record created for each LMOD (indicated by LMOD in the element type field of the E0 record) and one E0 record created for the LMOD’s side deck (indicated by SIDEDECK in the element type field of the E0 record). Both the E0 record with element type LMOD and the E0 record with element type SIDEDECK contain the same name (the LMOD’s name) in the name field of the E0 record.

For example, suppose that SMP/E adds load module LMODA, which has a SIDEDECK. Figure 76 shows the E0 records created for LMODA:

![Figure 76. Example of element record type 0](image)

**E1 - Element record type 1**

One Element Record Type 1 (E1) is created for each element having a UNIX shell script that changed in a target library during APPLY or RESTORE processing. The term changed in this context refers to an element that has been deleted or replaced in a target library as a result of SMP/E processing. See “Valid action types” on page 435 for more information.

The purpose of the E1 record is to identify the changed elements in the associated target libraries identified by the L0 and L1 records.

The format and contents of this record type are shown in Table 44 on page 426.
## Library change file records

### Table 44. Element record type 1

<table>
<thead>
<tr>
<th>Field name</th>
<th>Position (decimal)</th>
<th>Length (decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Record type</td>
<td>1</td>
<td>2</td>
<td>The characters 'E1'.</td>
</tr>
<tr>
<td>Name</td>
<td>3</td>
<td>8</td>
<td>The name of an SMP/E element processed during APPLY or RESTORE processing. The data is left-justified and padded with blanks.</td>
</tr>
<tr>
<td>Type</td>
<td>11</td>
<td>12</td>
<td>The type of hierarchical file system element processed by the APPLY or RESTORE command. The data is left-justified and padded with blanks.</td>
</tr>
<tr>
<td>Action</td>
<td>23</td>
<td>8</td>
<td>The type of action SMP/E took against this element during APPLY or RESTORE processing. Valid values are ADDREP and DELETE. This data is left-justified and padded with blanks.</td>
</tr>
<tr>
<td>DD name</td>
<td>31</td>
<td>8</td>
<td>The ddname associated with the target library associated with the named element in this E1 record. The data is left-justified and padded with blanks. For cross-zone elements, this field indicates the SMP/E generated ddname associated with the SYSLIB for this element. The associated L1 record also contains the related true name for the SYSLIB in the cross-zone. For action type DELETE, this field indicates the target library ddname from which the element has been deleted. For action type ADDREP, this field indicates the target library ddname into which the element has been added or replaced. An E1 record is created for every unique library that has an element change. Elements with multiple SYSLIBs will have the appropriate E1 records created for each unique SYSLIB changed. Therefore, it is possible to have the same element have the same change take place in multiple SYSLIBs. It is also possible to have multiple E1 records for the same element in the same SYSLIB if the element is both deleted and replaced. Normally, only the net effect would be represented in the E1 record. However, when a shell script is run, two records are produced because the shell script is run for each action. This field contains the ddname of the associated target library regardless of whether the allocation was done by a DD statement or DDDEF.</td>
</tr>
<tr>
<td>Phase</td>
<td>39</td>
<td>8</td>
<td>The phase of processing when the shell script was invoked to complete the installation of the identified element. Valid values are: PRE: Shell script was invoked before the action was performed. POST: Shell script was invoked after the action was performed. PRE,POST: Shell script was invoked both before and after the action was performed. The data is left-justified and padded with blanks.</td>
</tr>
</tbody>
</table>
Table 44. Element record type 1 (continued)

<table>
<thead>
<tr>
<th>Field name</th>
<th>Position (decimal)</th>
<th>Length (decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Path name</td>
<td>47</td>
<td>263</td>
<td>The full path name of the shell script. The data is left-justified and padded with blanks. The value is of the form /directory/file, in which:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>/directory/ Directory in a UNIX file system in which the shell script resides. This value can be up to 255 characters.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>file Name of the shell script file. This value can be 1 to 8 characters long.</td>
</tr>
</tbody>
</table>

For example, suppose that a ++HFS element named XMPHFS was added to target zone TZONEB using a shell script named MYSHELL in directory /usr/lpp/abc/ and that shell script MYSHELL was invoked after XMPHFS was installed. Figure 77 shows the E1 records created for LMODA:

![Figure 77. Example of element record type 1]

**H0 - Header record type 0**

One Header Record Type 0 (H0) is created for each delta produced by the results of APPLY or RESTORE processing. The purpose of the H0 record is to uniquely identify the start of a set of library change records.

The format and contents of this record type are shown in Table 45.

Table 45. Header record type 0

<table>
<thead>
<tr>
<th>Field name</th>
<th>Position (decimal)</th>
<th>Length (decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Record type</td>
<td>1</td>
<td>2</td>
<td>The characters 'H0'.</td>
</tr>
<tr>
<td>Zone name</td>
<td>3</td>
<td>7</td>
<td>The target zone name that was operated on by the APPLY or RESTORE command. It is left-justified and padded with blanks.</td>
</tr>
<tr>
<td>Time stamp</td>
<td>10</td>
<td>13</td>
<td>The date and time that APPLY or RESTORE processing completed for the entire delta. This field is in the form yyyydddhhmmss, where:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>yyyy year</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ddd day</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>hh hours</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>mm minutes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ss seconds</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Time is represented in 24-hour clock format (military time). The time value matches the time value in the T0 record and is the time that the GIM20501I message is issued for the associated APPLY or RESTORE command.</td>
</tr>
<tr>
<td>ERROR count</td>
<td>23</td>
<td>6</td>
<td>The character representation of the count of SYSMODs applied or restored in error. The data is padded on the left with character zeroes (0).</td>
</tr>
</tbody>
</table>
**Library change file records**

Table 45. Header record type 0 (continued)

<table>
<thead>
<tr>
<th>Field name</th>
<th>Position (decimal)</th>
<th>Length (decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INCMPLT count</td>
<td>29</td>
<td>6</td>
<td>The character representation of the count of SYSMODs left with a status of INCMPLT by the APPLY or RESTORE. The data is padded on the left with character zeroes (0).</td>
</tr>
<tr>
<td>APPLIED or RESTORED count</td>
<td>35</td>
<td>6</td>
<td>The character representation of the count of SYSMODs with a status of APPLIED or RESTORED. The data is padded on the left with character zeroes (0).</td>
</tr>
<tr>
<td>DELETED count</td>
<td>41</td>
<td>6</td>
<td>The character representation of the count of SYSMODs with a status of DELETED. The data is padded on the left with character zeroes (0).</td>
</tr>
<tr>
<td>SUPD count</td>
<td>47</td>
<td>6</td>
<td>The character representation of the count of SYSMODs with a status of SUPD. The data is padded on the left with character zeroes (0).</td>
</tr>
</tbody>
</table>

For example, suppose that a delta is produced for target zone TZONEA on January 16 of 2008 at 9:57:22 AM. During this execution of SMP/E, two SYSMODs were applied in error, one SYSMOD was left incomplete, 123 SYSMODs were applied successfully, seven SYSMODs were deleted, and three SYSMODs were superseded. Figure 78 shows the H0 records created for that delta:

```
12345678
---+---+---+---+---+---+---+---
HOTZONEA 2008016995722000002000001000123000007000003
```

**Figure 78. Example of header record type 0**

**L0 - Library record type 0**

One Library Record Type 0 (L0) is created for each target library changed during APPLY or RESTORE processing that is not associated with a pathname. A Library Record Type 1 (L1) is created for pathnames.

The purpose of the L0 record is to identify the target libraries (excluding pathnames) that were changed by this execution of APPLY or RESTORE.

SYSLIB concatenations used during the APPLY or RESTORE do not produce any L0 records.

The format and contents of this record type are shown in Table 46.

Table 46. Library record type 0

<table>
<thead>
<tr>
<th>Field name</th>
<th>Position (decimal)</th>
<th>Length (decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Record type</td>
<td>1</td>
<td>2</td>
<td>The characters ‘L0’.</td>
</tr>
<tr>
<td>DD name</td>
<td>3</td>
<td>8</td>
<td>The SMP/E ddname associated with this L0 record. The data is left-justified and padded with blanks. This field contains the ddname of the allocated target library regardless of whether the allocation was done by a DD statement or DDDEF. For cross-zone libraries, this field indicates the SMP/E-generated ddname for this library.</td>
</tr>
</tbody>
</table>
### Table 46. Library record type 0 (continued)

<table>
<thead>
<tr>
<th>Field name</th>
<th>Position (decimal)</th>
<th>Length (decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSLIB ddname</td>
<td>11</td>
<td>8</td>
<td>This is the SMP/E SYSLIB ddname associated with cross-zone libraries. This is the true ddname of the cross-zone library in the associated cross-zone. The data is left-justified and padded with blanks. For libraries that are not cross-zone libraries, this field is set to blanks.</td>
</tr>
<tr>
<td>Cross-zone name</td>
<td>19</td>
<td>7</td>
<td>The cross-zone name associated with this cross-zone library. The data is left-justified and padded with blanks. For libraries that are not cross-zone libraries, this field is set to blanks.</td>
</tr>
<tr>
<td>Library type</td>
<td>26</td>
<td>8</td>
<td>The type of library that is associated with this L0 record. The valid library types are LIBRARY, PDS, and SEQ. The data is left-justified and padded with blanks.</td>
</tr>
<tr>
<td>Volume name</td>
<td>34</td>
<td>6</td>
<td>The volume name associated with the SMP/E target library named in this record.</td>
</tr>
<tr>
<td>Data set name</td>
<td>40</td>
<td>44</td>
<td>The data set name of an SMP/E target library that was changed during APPLY or RESTORE processing. The data is left-justified and padded with blanks.</td>
</tr>
<tr>
<td>Catalog data set name</td>
<td>84</td>
<td>44</td>
<td>The catalog data set name associated with the target library data set name identified in this record. The data is left-justified and padded with blanks. If the data set name identified in this record is not cataloged, this field is set to blanks.</td>
</tr>
</tbody>
</table>

For example, Figure 79 shows the L0 record generated for a PDS library named SYSLINKLIB on volume MVSRESSYS1.LINKLIB in catalog CATALOG.MVSICFM.VMVSRES. The library has a ddname of LINKLIB.

![Figure 79. Example of library record type 0](image)

### L1 - Library record type 1

One Library Record Type 1 (L1) is created for each target library changed during APPLY or RESTORE processing that is associated with a pathname. An L0 record is created for libraries that are not associated with pathnames.

The purpose of the L1 record is to identify the target libraries containing pathnames that were changed by the associated APPLY or RESTORE command.

The format and contents of this record type are shown in Table 47.

### Table 47. Library record type 1

<table>
<thead>
<tr>
<th>Field name</th>
<th>Position (decimal)</th>
<th>Length (decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Record type</td>
<td>1</td>
<td>2</td>
<td>The characters 'L1':</td>
</tr>
</tbody>
</table>
Library change file records

Table 47. Library record type 1 (continued)

<table>
<thead>
<tr>
<th>Field name</th>
<th>Position (decimal)</th>
<th>Length (decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DD name</td>
<td>3</td>
<td>8</td>
<td>The SMP/E ddname associated with this Library Type 1 record. The data is left-justified and padded with blanks. This field contains the ddname of the allocated target library regardless of whether the allocation was done by a DD statement or DDDEF. For cross-zone libraries, this field indicates the SMP/E-generated ddname for this library.</td>
</tr>
<tr>
<td>SYSLIB name</td>
<td>11</td>
<td>8</td>
<td>This is the SMP/E SYSLIB ddname associated with cross-zone libraries. This is the true ddname of the cross-zone library in the associated cross-zone. The data is left-justified and padded with blanks. For libraries that are not cross-zone libraries, this field is set to blanks.</td>
</tr>
<tr>
<td>Cross-zone name</td>
<td>19</td>
<td>7</td>
<td>The cross-zone name associated with this cross-zone library. The data is left-justified and padded with blanks. For libraries that are not cross-zone libraries, this field is set to blanks.</td>
</tr>
<tr>
<td>Library type</td>
<td>26</td>
<td>8</td>
<td>The type of library that is associated with this L1 record. The only valid type is HFS. The data is left-justified and padded with blanks.</td>
</tr>
<tr>
<td>Path name</td>
<td>34</td>
<td>255</td>
<td>The path name of an SMP/E target library that was changed during APPLY or RESTORE processing. The data is left-justified and padded with blanks.</td>
</tr>
</tbody>
</table>

For example, Figure 80 shows the L1 record generated for a pathname of ‘/hfs_path_name/’, which is an HFS library with the ddname of HFSLIB1.

![Figure 80. Example of library record type 1](image)

L2 - Library record type 2

A Library Record Type 2 (L2) is created for each physical data set associated with a directory in a UNIX file system that was changed during APPLY or RESTORE processing. The purpose of the L2 record is to identify the physical data set in which a particular directory is located, how the data set is related to the directory, and the ddname that was used to allocate the directory.

The format and contents of this record type are shown in Table 48.

Table 48. Library record type 2

<table>
<thead>
<tr>
<th>Field name</th>
<th>Position (decimal)</th>
<th>Length (decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Record type</td>
<td>1</td>
<td>2</td>
<td>The characters 'L2'.</td>
</tr>
</tbody>
</table>
### Library change file records

#### Table 48. Library record type 2 (continued)

<table>
<thead>
<tr>
<th>Field name</th>
<th>Position (decimal)</th>
<th>Length (decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DD name</td>
<td>3</td>
<td>8</td>
<td>The SMP/E ddname associated with this Library Type 2 record. The data is left-justified and padded with blanks. This field contains the ddname that was used to allocate the directory in a UNIX file system that was updated. This value can be used to correlate the Library Type 2 records with the Library Type 1 records that describe the directory.</td>
</tr>
<tr>
<td>Relationship</td>
<td>11</td>
<td>8</td>
<td>This field describes the relationship between the updated directory and the physical data set identified in the record. Valid relationships are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>PATHHFS</strong> The HFS data set specified in this record contains the directory that contains a file updated by SMP/E.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>SYMHFS</strong> The HFS data set specified in this record contains the directory that contains a symbolic link associated with a file updated by SMP/E.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Library Type 2 records with a type of SYMHFS are created only when the SYMHFS data set names are different than the PATHHFS data set name. In addition, records only for unique SYMHFS data set names are produced. For example, if a file updated by SMP/E has two symbolic link values, and those symbolic link values reside in two directories that reside in the same physical data set, then only one Library Type 2 record with a type of SYMHFS is produced to describe this data set. In addition, if the data set that contains the symbolic links is the same physical data set that contains the file updated by SMP/E, then no Library Type 2 records with a type of SYMHFS are produced; a single record with a type of PATHHFS describes the physical data set that contains the file and its symbolic links. The data is left-justified and padded with blanks.</td>
</tr>
<tr>
<td>Physical data set</td>
<td>19</td>
<td>44</td>
<td>data set name of the physical data set that was updated when SMP/E updated a file in a UNIX file system. The data is left-justified and padded with blanks.</td>
</tr>
</tbody>
</table>

For example, suppose the following PTF is selected for APPLY processing and that the SBKSBIN library is allocated to directory `/service/usr/lpp/booksrv/cgi-bin/IBM/` in a UNIX file system.

```plaintext
++PTF(UW12345).
++VER(Z038) FMID(HYY2900).
++HFS(BKSMAIN) SYSLIB(SBKSBIN) DISTLIB(ABKSBIN)
    PARM(PATHMODE(7,5,5))
    LINK('.../bksmain')
    SYMLINK('.../.../.../bin/bksmain')
    SYMPATH('.../usr/lpp/booksrv/cgi-bin/bkmsmain').
```

The resulting APPLY processing replaces the BKSMAIN file in the `/service/usr/lpp/booksrv/cgi-bin/IBM/` directory with the copy of the HFS element supplied in the PTF. Also, when the file's symbolic link value specified on the MCS is concatenated with the file's directory, the symbolic link value resolves to `/service/bin/bkmsmain`. This file is the symbolic link and resides in the
Library change file records

/service/bin/ directory. In this example, the directory containing the symbolic link is in a different physical data set than the directory that contains the file BKSMAIN.

At the end of APPLY processing, SMP/E produces L1 and L2 records to summarize the updates performed. Figure 81 shows the L1 record that describes the directory that contains the file, as well as the L2 records that describe the data sets that contain the file and its symbolic links.

<table>
<thead>
<tr>
<th>Field name</th>
<th>Position (decimal)</th>
<th>Length (decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Record type</td>
<td>1</td>
<td>2</td>
<td>The characters 'P0'.</td>
</tr>
<tr>
<td>SYSMOD ID</td>
<td>3</td>
<td>7</td>
<td>The SYSMOD ID.</td>
</tr>
<tr>
<td>Status</td>
<td>10</td>
<td>8</td>
<td>The status of the SYSMOD. The following text may appear in this field. Where the text is less than 8 characters, it is left justified and padded with blanks.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• APPLIED</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• DELETED</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• ERROR</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• INCMPLT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• RESTORED</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• SUPD</td>
</tr>
</tbody>
</table>

Note: The superseded SYSMODs are included to aid in analyzing when a set of deltas is ready for distribution. When a SYSMOD in ERROR status is noted as superseded in a subsequent delta, its error status may be considered resolved.

The format and contents of this record type are shown in Table 49.

Table 49. SYSMOD status record type 0

<table>
<thead>
<tr>
<th>Field name</th>
<th>Position (decimal)</th>
<th>Length (decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Record type</td>
<td>1</td>
<td>2</td>
<td>The characters 'P0'.</td>
</tr>
<tr>
<td>SYSMOD ID</td>
<td>3</td>
<td>7</td>
<td>The SYSMOD ID.</td>
</tr>
<tr>
<td>Status</td>
<td>10</td>
<td>8</td>
<td>The status of the SYSMOD. The following text may appear in this field. Where the text is less than 8 characters, it is left justified and padded with blanks.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• APPLIED</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• DELETED</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• ERROR</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• INCMPLT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• RESTORED</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• SUPD</td>
</tr>
</tbody>
</table>

Note: SYSMODs identified in the SYSMOD STATUS REPORT with the following status are not included in the library change records, because they failed during SYSMOD selection processing.

Note: SYSMODs identified in the SYSMOD STATUS REPORT with the following status are not included in the library change records, because they failed during SYSMOD selection processing.

• EXCLUDED |
• HELD |
• NOGO |
• NOGO(E) |
• NOGO(H)
Table 49. SYSMOD status record type 0 (continued)

<table>
<thead>
<tr>
<th>Field name</th>
<th>Position (decimal)</th>
<th>Length (decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FMID</td>
<td>18</td>
<td>7</td>
<td>The FMID of the SYSMOD. If the status is SUPD and the SYSMOD is not in process, this field contains blanks.</td>
</tr>
<tr>
<td>Type</td>
<td>25</td>
<td>8</td>
<td>The SYSMOD type. The following text is valid in this field: • FUNCTION • PTF • APAR • USERMOD If the status is SUPD and the SYSMOD is not in process, this field contains blanks.</td>
</tr>
</tbody>
</table>

The SYSMOD Status (P0) records follow the E0 record in the set of library change records for the command. Figure 82 shows examples of P0 records written for APPLY processing.

Additionally, fields are added to the header and trailer records to represent the count of SYSMODs represented in the P0 records with each status.

This information tells you whether any errors are contained in the current delta. However, just because a delta is error free, it does not mean that it can be distributed without resolving errors in previous deltas. You must perform the task of error resolution among deltas prior to distribution in order to ensure an error free package. Refer to “Usage recommendations” on page 436 for usage information.

S0 - SMP/E environment record type 0

One SMP/E Environment Record Type 0 (S0) is created for each delta produced by the results of APPLY or RESTORE processing.

The purpose of the S0 record is to identify the SMP/E environment from which this set of library change records was derived.

The format and contents of this record type are shown in Table 50.

Table 50. SMP/E environment record type 0

<table>
<thead>
<tr>
<th>Field name</th>
<th>Position (decimal)</th>
<th>Length (decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Record type</td>
<td>1</td>
<td>2</td>
<td>The characters 'S0'.</td>
</tr>
<tr>
<td>Command</td>
<td>3</td>
<td>7</td>
<td>The SMP/E command that was executed. Valid values are APPLY or RESTORE. The data is left-justified and padded with blanks.</td>
</tr>
</tbody>
</table>
**Library change file records**

**Table 50. SMP/E environment record type 0 (continued)**

<table>
<thead>
<tr>
<th>Field name</th>
<th>Position (decimal)</th>
<th>Length (decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMP/E level</td>
<td>10</td>
<td>8</td>
<td>The level of SMP/E under which the APPLY or RESTORE command was executed. This subfield is in the form vvrrmmpp, where:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>* vv version of SMP/E *</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>* rr release of SMP/E *</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>* mm release of SMP/E *</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>* pp PTF level of SMP/E *</td>
</tr>
<tr>
<td>Data set name</td>
<td>18</td>
<td>44</td>
<td>The data set name of the SMP/E target zone CSI that contains the target zone operated on by the SMP/E APPLY or RESTORE function. The data is left-justified and padded with blanks.</td>
</tr>
<tr>
<td>Catalog data set name</td>
<td>62</td>
<td>44</td>
<td>The catalog data set name of the SMP/E target zone CSI that contains the target zone operated on by the SMP/E APPLY or RESTORE function. The data is left-justified and padded with blanks.</td>
</tr>
<tr>
<td>Volume name</td>
<td>106</td>
<td>6</td>
<td>The volume name of the SMP/E target zone CSI that contains the target zone operated on by the SMP/E APPLY or RESTORE function.</td>
</tr>
</tbody>
</table>

For example, Figure 83 shows an S0 record for a target zone CSI data set named SAMPLE.ZOSYS.TARGET.CSI on volume SMPVOL in catalog CATALOG.MVSICF1.VMVSRES. The SMP/E level is 03.06.00, and the SMP/E command executed was APPLY.

**Figure 83. Example of SMP/E environment record type 0**

**T0 - Trailer record type 0**

One Trailer Record Type 0 (T0) is created for each delta produced by the results of APPLY or RESTORE processing. The purpose of the T0 record is to uniquely identify the end of a set of library change records.

The format and contents of this record type are shown in Table 51.

**Table 51. Trailer record type 0**

<table>
<thead>
<tr>
<th>Field name</th>
<th>Position (decimal)</th>
<th>Length (decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Record type</td>
<td>1</td>
<td>2</td>
<td>The characters 'T0'.</td>
</tr>
<tr>
<td>Zone name</td>
<td>3</td>
<td>7</td>
<td>The target zone name that was operated on by the APPLY or RESTORE command. It is left-justified and padded with blanks.</td>
</tr>
</tbody>
</table>
### Table 51. Trailer record type 0 (continued)

<table>
<thead>
<tr>
<th>Field name</th>
<th>Position (decimal)</th>
<th>Length (decimal)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time stamp</td>
<td>10</td>
<td>13</td>
<td>The date and time that APPLY or RESTORE processing completed for the entire delta. This field is in the form yyyydddhhmmss, where: yyyy year, ddd day, hh hours, mm minutes, ss seconds. Time is represented in 24-hour clock format (military time). The time value matches the time value in the T0 record and is the time that the GIM20501I message is issued for the associated APPLY or RESTORE command.</td>
</tr>
<tr>
<td>ERROR count</td>
<td>23</td>
<td>6</td>
<td>The character representation of the count of SYSMODs applied or restored in error. The data is padded on the left with character zeroes (0).</td>
</tr>
<tr>
<td>INCMPLT count</td>
<td>29</td>
<td>6</td>
<td>The character representation of the count of SYSMODs left with a status of INCMPLT by the APPLY or RESTORE. The data is padded on the left with character zeroes (0).</td>
</tr>
<tr>
<td>APPLIED or RESTORED</td>
<td>35</td>
<td>6</td>
<td>The character representation of the count of SYSMODs with a status of APPLIED or RESTORED. The data is padded on the left with character zeroes (0).</td>
</tr>
<tr>
<td>DELETED count</td>
<td>41</td>
<td>6</td>
<td>The character representation of the count of SYSMODs with a status of DELETED. The data is padded on the left with character zeroes (0).</td>
</tr>
<tr>
<td>SUPD count</td>
<td>47</td>
<td>6</td>
<td>The character representation of the count of SYSMODs with a status of SUPD. The data is padded on the left with character zeroes (0).</td>
</tr>
</tbody>
</table>

For example, Figure 84 shows a T0 record for a target zone named MYTZN. The delta processing for this zone ended on January 16, 2008 at 10:12:34 AM. During this execution of SMP/E, one SYSMOD was applied in error, four SYSMODs were left incomplete, ten SYSMODs were applied successfully, one SYSMOD was deleted, and two SYSMODs were superseded.

![Figure 84. Trailer record type 0](image)

### Valid action types

**Action**  
**Meaning**

**ADDREP**  
The ADDREP action indicates either that a new element or LMOD was added to a target library during APPLY or RESTORE processing, or that an existing element or LMOD was replaced in a target library during APPLY or RESTORE processing.

This includes elements added or replaced by way of:
Library change file records

- Data element MCS
- Hierarchical file system element MCS
- Link-edits or copies of load modules:
  - ++MOD MCS (add or replace)
- ++MAC MCS
- ++JAR MCS
- ++JARUPD MCS
- ++MACUPD MCS
- ++MOVE MCS for MACs, SRC, and LMODs
- ++PROGRAM MCS
- ++RENAME MCS for load modules
- ++SRC MCS
- ++SRCUPD MCS
- ++ZAP MCS

This also includes aliases added or replaced by way of:
- Data element MCS with ALIAS operand
- Hierarchical file system element MCS with LINK operand
- Link-edits or copies of load modules:
  - ++MOD MCS with TALIAS operand
  - LMODs defined in JCLIN
- ++JAR or ++JARUPD MCS with LINK operand
- ++MAC MCS with MALIAS operand
- ++MACUPD MCS with MALIAS operand
- ++MOVE MCS for MACs and LMODs
- ++PROGRAM MCS with ALIAS operand
- ++RENAME MCS for load modules

DELETE

The DELETE action indicates that an element or LMOD was deleted from a target library during APPLY processing.

This includes elements and their aliases deleted by way of:
- DELETE operand on data element MCS
- DELETE operand on hierarchical file system element MCS
- DELETE operand on ++JAR MCS
- DELETE operand on ++MAC MCS
- DELETE operand on ++PROGRAM MCS
- DELETE operand on ++SRC MCS
- DELETE operand on ++VER MCS
- ++DELETE MCS for load modules

This also includes aliases deleted by way of:
- ALIAS operand on ++DELETE MCS for load modules

Usage recommendations

Users cannot simply look at the records created by the library change interface and move the identified parts. As is the case whenever SMP/E is executed, each processed SYSMOD has a specific completion status when the particular APPLY or RESTORE completes. Therefore, the results of each APPLY or RESTORE must be looked at collectively. Until a set of SYSMODs has been successfully processed, you cannot distribute the library change records associated with those SYSMODs.

Therefore, you must first get a successful APPLY CHECK or RESTORE CHECK before attempting to use the records produced by library change processing for a specified target zone. Since there are no records produced during CHECK processing, there are no records to be lost until delta processing is activated.
Library change file records

Once activated, delta processing produces records for an execution of APPLY or RESTORE. Since all SYSMODs may not successfully complete in one run, the records from each run against the same set of SYSMODs must be kept.

Once a successful APPLY or RESTORE is obtained, the net result of the processing must be analyzed by elements, LMODs, or both to determine the appropriate action to take to distribute the elements and LMODs.

Alternatively, once a successful APPLY or RESTORE is obtained, the net result of the libraries updated can be analyzed to determine the appropriate action to take to distribute the libraries affected by the APPLY/RESTORE activity.
Chapter 9. SMP/E exit routines

This chapter describes intended Programming Interfaces that allow the customer to write programs to obtain the services of IBM SMP/E for z/OS, V3R6.

This chapter describes how to write exit routines that:
- Process statements in SMPPTFIN at RECEIVE time
- Control retry processing when data sets run out of space during ACCEPT, APPLY, GZONEMERGE, LINK LMODS, LINK MODULE, RECEIVE, or RESTORE processing.

LINK LMODS,

A common parameter list is used to pass information between SMP/E and the exit routines. It is pointed to by register 1 and is mapped by macro GIMMPUXP in SYS1.MA CLIB. Table 52 shows the format and contents of the parameter list.

Table 52. GIMMPUXP: exit routine parameter list

<table>
<thead>
<tr>
<th>Field name</th>
<th>Offset (DEC)</th>
<th>Offset (HEX)</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UXPUXNUM</td>
<td>+ 0</td>
<td>+ 0</td>
<td>2</td>
<td>Exit routine number: X'0001' – RECEIVE X'0002' – RETRY</td>
</tr>
<tr>
<td></td>
<td>+ 2</td>
<td>+ 2</td>
<td>2</td>
<td>Not used</td>
</tr>
<tr>
<td>UXPUXNAM</td>
<td>+ 4</td>
<td>+ 4</td>
<td>8</td>
<td>Name of exit routine</td>
</tr>
<tr>
<td>UXPUXAD</td>
<td>+12</td>
<td>+ C</td>
<td>4</td>
<td>Address of exit routine</td>
</tr>
<tr>
<td>UXPFUNCT</td>
<td>+16</td>
<td>+10</td>
<td>8</td>
<td>SMP/E command</td>
</tr>
<tr>
<td>UXPRRMA D</td>
<td>+24</td>
<td>+18</td>
<td>4</td>
<td>Address of exit routine parameter list</td>
</tr>
<tr>
<td>UXPLOJAD</td>
<td>+28</td>
<td>+1C</td>
<td>4</td>
<td>Not used</td>
</tr>
<tr>
<td>UXPLOEAD</td>
<td>+32</td>
<td>+20</td>
<td>4</td>
<td>Not used</td>
</tr>
<tr>
<td>UXPCTBAD</td>
<td>+36</td>
<td>+24</td>
<td>4</td>
<td>Reserved for future use</td>
</tr>
<tr>
<td>UXP MODAD</td>
<td>+40</td>
<td>+28</td>
<td>4</td>
<td>Reserved for future use</td>
</tr>
</tbody>
</table>

The following sections describe the exit routines for RECEIVE and retry processing.

**RECEIVE exit routine**

The RECEIVE exit routine allows you to scan statements in the SMPPTFIN data set at RECEIVE time. This exit routine must be defined by a GIMEXITS control statement in the GIMEXITS member of SMPPARM, which tells SMP/E RECEIVE processing that an exit routine exists and should be called. Optionally, the GIMEXITS control statement may also specify the data set containing the exit routine. This exit routine must be a load module residing in an authorized library.

For more information about specifying exit routines in GIMEXITS, see "Defining control statements in SMPPARM members," on page 129. The RECEIVE exit routine is loaded at the start of RECEIVE command processing and is deleted at the end of RECEIVE processing.
When this exit routine is called, the parameter list contains the values shown in Table 53 and Table 54.

**Table 53. RECEIVE exit routine: parameter list values**

<table>
<thead>
<tr>
<th>Field name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UXPUXNUM</td>
<td>X'0001' (exit routine number)</td>
</tr>
<tr>
<td>UXPUXNAM</td>
<td>Name of exit routine</td>
</tr>
<tr>
<td>UXPUXAD</td>
<td>Address of exit routine</td>
</tr>
<tr>
<td>UXPFUNCT</td>
<td>RECEIVE</td>
</tr>
<tr>
<td>UXPPRMAD</td>
<td>Address of 81-byte buffer area (see Table 54)</td>
</tr>
</tbody>
</table>

**Table 54. RECEIVE exit routine: buffer passed by UXPPRMAD**

<table>
<thead>
<tr>
<th>Field name</th>
<th>Offset (DEC)</th>
<th>Offset (HEX)</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UX001RC</td>
<td>+ 0</td>
<td>+ 0</td>
<td>1</td>
<td>X'00' – Buffer contains record to be processed</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X'04' – End-of-file on SMPPTFIN</td>
</tr>
<tr>
<td>UX001RCD</td>
<td>+ 1</td>
<td>+ 1</td>
<td>80</td>
<td>Record from SMPPTFIN</td>
</tr>
</tbody>
</table>

According to the input record, the RECEIVE exit may decide to continue RECEIVE processing, change the record, insert data after the record, or skip the record. Or, it may choose to stop processing for the SYSMOD, for the RECEIVE command, or for SMP/E.

When the exit routine returns control to SMP/E, it must set register 15 to one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Continue normal RECEIVE processing.</td>
</tr>
<tr>
<td>8</td>
<td>Stop SYSMOD processing. SMP/E does not receive this SYSMOD, but continues to pass records from the SYSMOD to the exit routine.</td>
</tr>
<tr>
<td>12</td>
<td>Stop RECEIVE processing.</td>
</tr>
<tr>
<td>16</td>
<td>Stop SMP/E processing.</td>
</tr>
<tr>
<td>20</td>
<td>Insert a record after the current one in the buffer.</td>
</tr>
<tr>
<td>24</td>
<td>Skip the record in the buffer area.</td>
</tr>
</tbody>
</table>

If any other value is returned, SMP/E issues an error message and fails.

**Changing records within SMPPTFIN**

To change the current record within the SMPPTFIN data set, the exit routine must change the record currently in UX001RCD and set the return code in register 15 to 0 or 20. A return code of 0 indicates to SMP/E that the changed record should be processed and normal processing should continue. A return code of 20 indicates to SMP/E that the exit routine inserts records after this changed record. For more information, see [Inserting records within SMPPTFIN](Inserting records within SMPPTFIN).

**Inserting records within SMPPTFIN**

To insert one or more records within the SMPPTFIN data set after the current record in UX001RCD, the exit routine must set the return code in register 15 to 20.
SMP/E first processes the current record in UX001RCD; then it calls the exit routine again. On this call, the exit routine must place the record to be inserted into UX001RCD. If more records are to be inserted, the exit routine must set the return code in register 15 to 20. If this is the last record to be inserted, the exit routine must set the return code to 0.

When the exit routine passes a return code of 0, SMP/E processes the record in UX001RCD; then it continues processing with the next record in the SMPPTFIN data set.

**Inserting records at the end of SMPPTFIN**

To insert one or more records at the end of the SMPPTFIN data set, the exit routine must set the return code in register 15 to 20.

SMP/E then calls the exit routine again. On this call, the exit routine must place the record to be inserted into UX001RCD. If more records are to be inserted, the exit routine must set the return code in register 15 to 20. If this is the last record to be inserted, the exit routine must set the return code to 0. SMP/E processes the inserted records as if they were at the end of the SMPPTFIN data set.

When the exit routine passes a return code of 0, SMP/E processes the record in UX001RCD; then it continues with normal end-of-file processing.

**Note:** When you specify the SELECT operand during RECEIVE processing, and all selected SYSMODs have been processed, SMP/E indicates end-of-file to the exit routine. If the exit routine passes a return code of 20 at this point, the add request is ignored, and SMP/E continues with end-of-file processing.

**Skipping records in SMPPTFIN**

To skip a record in SMPPTFIN, the exit routine must set the return code in register 15 to 24. SMP/E does not process the current record in UX001RCD; instead, it processes the next record in SMPPTFIN.

**Retry exit routine**

The retry exit routine enables you to control retry processing when a data set runs out of space during ACCEPT, APPLY, GZONEMERGE, LINK LMODS, LINK MODULE, RECEIVE, or RESTORE processing. (In retry processing, the data set is compressed and the utility that failed is called again.) This exit routine must be defined by a GIMEXITS control statement in the GIMEXITS member of SMPPARM, which tells SMP/E retry processing that an exit routine exists and should be called. Optionally, the GIMEXITS control statement may also specify the data set containing the exit routine. This exit routine must be a load module residing in an authorized library. For more information about specifying exit routines in GIMEXITS, see Chapter 3, “Defining control statements in SMPPARM members,” on page 129. The retry exit routine is loaded at the start of ACCEPT, APPLY, GZONEMERGE, LINK LMODS, LINK MODULE, RECEIVE, or RESTORE command processing and is deleted at the end of command processing.

**Note:** The processing of this routine is not affected by the debatching SMP/E does after retry processing fails, because this routine is called as part of the initial retry processing, before debatching is attempted.
When a data set runs out of space, an x37 abend occurs. If SMP/E determines that a retry can be attempted, it cancels the abend dump and calls the retry exit routine. The parameter list contains these values:

**Table 55. Retry exit routine: parameter list values**

<table>
<thead>
<tr>
<th>Field name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UXPUXNUM</td>
<td>X'0002' (exit routine number)</td>
</tr>
<tr>
<td>UXPUXNAM</td>
<td>Name of exit routine</td>
</tr>
<tr>
<td>UXPUXAD</td>
<td>Address of exit routine</td>
</tr>
<tr>
<td>UXPFUNCT</td>
<td>ACCEPT, APPLY, GZMRG, LINK, RECEIVE, or RESTORE</td>
</tr>
<tr>
<td>UXPPRMAD</td>
<td>Address of 25-byte parameter list (see Table 56)</td>
</tr>
</tbody>
</table>

**Table 56. Retry exit routine: parameter list passed by UXPPRMAD**

<table>
<thead>
<tr>
<th>Field name</th>
<th>Offset (DEC)</th>
<th>Offset (HEX)</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UX002DDN</td>
<td>+ 0</td>
<td>+ 0</td>
<td>8</td>
<td>ddname of data set that ran out of space</td>
</tr>
<tr>
<td>UX002PGM</td>
<td>+ 8</td>
<td>+ 8</td>
<td>8</td>
<td>Name of the utility program that failed</td>
</tr>
<tr>
<td>UX002ACH</td>
<td>+16</td>
<td>+10</td>
<td>3</td>
<td>ABEND code (hex) in the same format as field SDWACMPC in the SDWA control block</td>
</tr>
<tr>
<td>UX002RCH</td>
<td>+19</td>
<td>+13</td>
<td>1</td>
<td>ABEND reason code (hex)</td>
</tr>
<tr>
<td>UX002ACP</td>
<td>+20</td>
<td>+14</td>
<td>3</td>
<td>ABEND code (EBCDIC)</td>
</tr>
<tr>
<td>UX002RCP</td>
<td>+23</td>
<td>+17</td>
<td>2</td>
<td>ABEND reason code (EBCDIC)</td>
</tr>
</tbody>
</table>

According to the input, the retry exit can either cancel retry processing or perform some other method of recovery.

When the exit routine returns control to SMP/E, it must set register 15 to one of the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Continue normal retry processing.</td>
</tr>
<tr>
<td>12</td>
<td>Stop command processing and perform no retry.</td>
</tr>
<tr>
<td>16</td>
<td>Stop SMP/E processing and perform no retry.</td>
</tr>
<tr>
<td>20</td>
<td>Perform modified retry processing. Call the failing utility, but do not compress the failing data set.</td>
</tr>
</tbody>
</table>

If any other value is returned, SMP/E converts it to 12, and command processing fails.
Chapter 10. JCL statements required to invoke SMP/E

Unless you are using the SMP/E dialogs, you must provide the following JCL statements to invoke SMP/E:

- A JOB statement
- An EXEC statement
- DD (data definition) statements

**JOB statement**

The **JOB** statement describes your installation-dependent parameters. The JOB statement (or the EXEC statement, or both) can also include the REGION parameter to set the size of the region in which SMP/E runs. For details, see [z/OS MVS JCL User’s Guide](SA22-7598) or [z/OS MVS JCL Reference](SA22-7597).

**Note:** To enable the SMP/E job step to get the maximum space above 16 megabytes, you can specify REGION=0M.

**EXEC statement**

The **EXEC** statement must specify PGM=GIMSMP or the name of your cataloged procedure for calling SMP/E. (For an example of a cataloged procedure, see [SMP/E User’s Guide](https://www.ibm.com).) The following can be specified in the EXEC statement PARM parameter:

- **COMPAT=WARNBYPASS** or **COMPAT=NOWARNBYPASS**
  - The COMPAT parameter is used to control incompatible behaviors of SMP/E processing.
  - **COMPAT(WARNBYPASS)** indicates that the APPLY and ACCEPT commands will issue warning messages to identify bypassed SYSTEM HOLD exceptions. This is the behavior for releases of SMP/E prior to V3R5.
  - **COMPAT(NOWARNBYPASS)** indicates that the APPLY and ACCEPT commands will issue informational messages to identify bypassed SYSTEM HOLD exceptions. If neither COMPAT(WARNBYPASS) or COMPAT(NOWARNBYPASS) is specified, the default is COMPAT(NOWARNBYPASS).

- **CSI=dsname**
  - where **dsname** is the name of the CSI data set containing the global zone. (This data set is also known as the *master CSI*.) This parameter is used to enable SMP/E to allocate the master CSI data set dynamically.

  **Note:** If there is an SMPCSI DD statement, the **CSI=dsname** operand is not allowed. If both are specified, SMP/E does not run.

- **DATE=date**
  - where **date** can be one of the following:
    - **U** or **IPL** To use the IPL date of the system.
REPLY

To request the date from the operator. As a result, SMP/E issues message GIM399I.

yyddd

To specify a specific date, where yy is the year and ddd is the day of the year (the Julian date).

If DATE is not specified, the IPL date of the system is used.

LANGUAGE=xxx

where xxx can be one of the following:

ENU US English
JPN Japanese

The LANGUAGE option defines which language to use for SMP/E messages.

LANGUAGE can also be specified as L. If LANGUAGE is not specified, the default is LANGUAGE=ENU.

You can specify LANGUAGE=JPN only if you have installed the Japanese language feature of SMP/E. If you have installed the Japanese language feature, you can specify LANGUAGE=ENU or LANGUAGE=JPN. (You do not need to install the English feature along with the Japanese feature.)

The output devices used must support the selected language and English single-byte characters. SMP/E does not check to verify that the output devices provide this support.

PROCESS=WAIT or PROCESS=END

The PROCESS parameter is used to control how long a job should wait if a CSI or PTS data set is not immediately available because it is currently being used either by another job or by a dialog.

• WAIT causes the job to wait until the data set is available. A message is issued to the system operator every 30 minutes while the job is waiting.
• END causes the job to wait for 10 minutes. If the data set is still not available after the 10-minute wait, the command requiring the data set is stopped.

If PROCESS is not specified, the default is PROCESS=WAIT.

For more information about obtaining and sharing CSI data sets, see the “Sharing SMP/E Data Sets” appendix in [SMP/E Commands].

Processing of the PTS data set is also affected by the WAITFORDSN value specified in its DDDEF entry. WAITFORDSN determines whether SMP/E should wait to allocate a data set that is not immediately available. If the DDDEF entry specifies WAITFORDSN=NO (or lets this value default to NO) and the data set is not available, allocation of the data set fails, regardless of the PROCESS value specified on the EXEC statement. If WAITFORDSN=NO, SMP/E does not wait to retry allocation of the data set.

For example, suppose a PTS with a disposition of OLD is already being used by a job, and a second job tries to access the same PTS data set by allocating it through a DDDEF entry. The DDDEF entry used by the second job for the PTS specifies WAITFORDSN=NO. As a result, allocation of the PTS fails for the second job.
DD statements define the data sets that can be used in SMP/E processing. For information about the data sets required for each command, see the chapters on individual SMP/E commands in SMP/E Commands.

Note: You can use DDDEF entries, rather than DD statements, to allocate many of the necessary data sets. For more information, see “DDDEF entry (distribution, target, and global zone)” on page 197.
Sample cataloged procedure
Chapter 11. Service routines

SMP/E provides these service routines:
- GIMCPTS
- GIMDTS
- GIMGTPKG
- GIMUNZIP
- GIMXSID
- GIMXTRX
- GIMZIP

**GIMCPTS: SYSMOD compaction service routine**

The GIMCPTS service routine is used to compact or expand inline element data within SYSMODs. The inline element data can be compacted prior to receiving a SYSMOD or SYSMODs already received can be compacted.

SYSMODs with their inline element data compacted require less space, so the space requirements of the SMPPTS data set should be reduced. Of course, SMP/E APPLY and ACCEPT processing will expand any compacted inline element data prior to installing the element in a target or distribution library respectively.

GIMCPTS can also be used to expand the inline element data within a SYSMOD. This could be useful if you need to send the SYSMOD to another system where the SMP/E installed on the other system cannot handle compacted inline element data. It could also be useful for viewing inline data. Note that the SMP/E Query Dialogs can be used to view MCS entries from the SMPPTS data set and that SMP/E expands the inline element data before displaying it.

**Note:** GIMCPTS is a separate load module residing in the MIGLIB library that runs independently from the rest of SMP/E processing.

**Calling GIMCPTS**

Following are the JCL statements needed to call GIMCPTS:

```jcl
//JOBx JOB ...
//STEP1 EXEC PGM=GIMCPTS,PARM='options'
//SYSPRINT DD SYSOUT=* 
//SYSUT1 DD DSN=aaaaaaaa,DISP=SHR
//SYSUT2 DD DSN=bbbbbbbb,DISP=OLD
```

*Figure 85. JCL to call GIMCPTS*

**EXEC**

is the statement used to call GIMCPTS. The EXEC statement must specify **PGM=GIMCPTS**. The following options may be specified on the EXEC statement PARM operand:

**COMPACT**

The COMPACT option indicates to GIMCPTS the input data is to be
GIMCPTS

COMPACT is mutually exclusive with EXPAND. If neither COMPACT nor EXPAND is specified, the default is COMPACT.

EXPAND

The EXPAND option indicates to GIMCPTS the input data is to be expanded. EXPAND is mutually exclusive with COMPACT. If neither EXPAND or COMPACT is specified, the default is COMPACT.

LANGUAGE=xxx

where xxx can be one of the following:

ENU US English
JPN Japanese

The LANGUAGE option defines which language to use for GIMCPTS messages. LANGUAGE can also be specified as L. If LANGUAGE is not specified, the default is LANGUAGE=ENU.

SYSPRINT

is used by GIMCPTS for messages.

The record format (RECFM) of SYSPRINT must be FBA, and the record length (LRECL) may be either 81 or 121. GIMCPTS will format messages to 80 characters or 120 characters in length, based on the record length of the SYSPRINT data set.

SYSUT1

points to the sequential or partitioned input data. Sequential input data may be a sequential data set or a single member of a partitioned data set. Partitioned input data is an entire partitioned data set (PDS) whose members each contain input data. The actual input data must be the MCS of one or more SYSMODs. The inline element data within the SYSMODs will be either compacted or expanded, as determined by the COMPACT and EXPAND options.

The record format (RECFM) of SYSUT1 must be fixed (F), fixed-block (FB), or fixed-block standard (FBS), and the record length (LRECL) must be 80.

SYSUT2

points to the sequential or partitioned output destination. If the input is sequential, then SYSUT2 must also be sequential. If the input is partitioned, then SYSUT2 must also be partitioned. The output will contain the input SYSMODs with either compacted or expanded inline element data.

The record format (RECFM) of SYSUT2 must be fixed (F), fixed-block (FB), or fixed-block standard (FBS), the record length (LRECL) must be 80, and the BLKSIZE value must be a multiple of 80.

IBM recommends that SYSUT2 be allocated the same size as the original SMPPTs data set. The amount of compaction that can be achieved will vary depending on the content and packaging method of the SYSMOD in the originating SMPPTs data set. For example, if the originating SMPPTs data set contained only functions packaged in RELFILE format, no compaction would occur.

After GIMCPTS is run, you can release the unused space using ISPF (or other methods). You could also specify RLSE on the SPACE parameter of the SYSUT2 data set (if it is being allocated when GIMCPTS is being run) and z/OS will free unused space when it is closed.
Example

Suppose your system has an existing SMPPTS data set that is quite large and requires the space of an entire physical volume. You can use the SMP/E GIMCPTS compaction service routine to compact data within the members of the SMPPTS. In this example, the existing SMPPTS data set is compacted in place. However, a new data set could be allocated and used to receive the compacted data.

```plaintext
//JOBx   JOB ...
//COMPACT EXEC PGM=GIMCPTS,PARM='COMPACT,LANGUAGE=ENU'
//**
//SYSPRINT DD SYSOUT=*
//SYSUT1 DD DSN=SMP.ZOSR1.SMPPTS,DISP=SHR
//SYSUT2 DD DSN=SMP.ZOSR1.SMPPTS,DISP=OLD
```

Figure 86. Sample GIMCPTS jobstream

When this is executed, the GIMCPTS service routine will compact the inline element data within all SYSMOD members in the SMPPTS data set and rewrite them to the SMPPTS data set.

Note: You should only attempt to compact a partitioned data set in place if it is a PDSE. If the data set is a simple PDS however, it is likely the data set will get an out of space condition during the operation, unless a large amount of free space is available.

Return codes

To help you diagnose errors, GIMCPTS issues messages and return codes. The messages are documented in [SMP/E Messages, Codes, and Diagnosis](#). Here is a description of the return codes:

<table>
<thead>
<tr>
<th>Return code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The input data was processed successfully.</td>
</tr>
<tr>
<td>8</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• Syntax errors were found in an input SYSMOD.</td>
</tr>
<tr>
<td></td>
<td>• An unsupported compaction dictionary was used to compact data.</td>
</tr>
<tr>
<td></td>
<td>• Compact data is incomplete.</td>
</tr>
<tr>
<td>12</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• Data sets were missing.</td>
</tr>
<tr>
<td></td>
<td>• Data sets could not be opened.</td>
</tr>
<tr>
<td></td>
<td>• Data set organization of SYSUT1 and SYSUT2 is not sequential or</td>
</tr>
<tr>
<td></td>
<td>partitioned.</td>
</tr>
<tr>
<td></td>
<td>• Data set organizations of SYSUT1 and SYSUT2 did not match.</td>
</tr>
<tr>
<td></td>
<td>• SYSUT2 data set had incorrect RECFM or LRECL.</td>
</tr>
<tr>
<td></td>
<td>• SYSUT2 and SYSUT1 data sets are the same data set and they are</td>
</tr>
<tr>
<td></td>
<td>sequential or members of a PDS.</td>
</tr>
<tr>
<td></td>
<td>• Directory space for SYSUT2 was exceeded.</td>
</tr>
<tr>
<td></td>
<td>• Compression and expansion services are not supported.</td>
</tr>
<tr>
<td>16</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• An I/O error occurred.</td>
</tr>
<tr>
<td></td>
<td>• Syntax error was found on the EXEC statement parameters.</td>
</tr>
<tr>
<td>20</td>
<td>SYSPRINT data set is missing.</td>
</tr>
<tr>
<td>&gt; 20</td>
<td>Internal error. Report the error to the IBM Support Center.</td>
</tr>
</tbody>
</table>

Chapter 11. Service routines 449
GIMDTS: Data transformation service routine

Elements may have any of a variety of record formats, depending on how they are meant to be used. However, when elements are packaged inline in a SYSMOD, they must contain fixed-block 80 records. To help you package inline elements, SMP/E provides the GIMDTS service routine. GIMDTS is a background utility program that transforms data into fixed-block 80 records. For example, it can be used to format inline replacements for data elements. Although GIMDTS is packaged as part of SMP/E, it is a separate load module residing in SYS1.MIGLIB and runs independently from the rest of SMP/E processing.

The input for GIMDTS must meet these requirements:
- It must be a sequential data set or a member of a partitioned data set (PDS).
- It can contain either variable-length or fixed-length records.

The output from GIMDTS is in this format:
- It is a sequential data set or a member of a partitioned data set.
- It has these attributes:
  
  RECFM=FB
  LRECL=80
  BLKSIZE=a multiple of 80

After using GIMDTS to transform an element into the required format, you can package the transformed data inline in a SYSMOD, following the associated data element MCS. Later, when the element is installed, it is changed back to its original format.

The following sections describe:
- Statements used to call GIMDTS
- Processing done by GIMDTS
- Return codes issued by GIMDTS

Calling GIMDTS

Here are the JCL statements that are required to call GIMDTS:

```
//JOBx       JOB 'account','name',MSGLEVEL=(1,1)
//STEP1      EXEC PGM=GIMDTS
//SYSPRINT   DD SYSPUT=A
//SYSUT1     DD DSN=aaaaaaa,DISP=SHR
//SYSUT2     DD DSN=bbbbbbb,DISP=SHR
```

*Figure 87. Sample GIMDTS jobstream*

**EXEC**
is the statement used to call GIMDTS.

**SYSPRINT**
is used by GIMDTS for messages.

**SYSUT1**
points to the sequential data set or PDS member containing the data to be transformed.
GIMDTS

The record format (RECFM) must be F, FA, FM, FB, FBA, FBM, V, VA, VM, VB, VBA, VBM, VS, or VBS.

SYSUT2

points to the sequential data set or PDS member that will contain the transformed data. This data set must be on DASD.

The record format of this file should be fixed-block (RECFM=FB) with a BLKSIZE value that is a multiple of 80.

Processing

GIMDTS first checks for the required data sets, opens the SYSUT1 and SYSUT2 data sets, and makes sure these data sets have the correct attributes. If necessary, GIMDTS changes the RECFM value of the SYSUT2 data set to FB and the BLKSIZE value to 3200. Next, GIMDTS reads records from the SYSUT1 data set and transforms the data.

After processing the data, GIMDTS writes the output records to the SYSUT2 data set. GIMDTS also issues messages to SYSPRINT indicating the input data set, the output data set, and the return code. If any errors occur during processing, GIMDTS issues messages describing the errors and stops processing. If an abend occurs, normal system abend processing is done.

If SYSUT1 is already fixed-block with LRECL=80 and the data has no "++" characters in columns 1 and 2, then the data needs no transformation and is simply copied from SYSUT1 to SYSUT2.

Return codes

GIMDTS may issue the following return codes:

<table>
<thead>
<tr>
<th>Return code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>The input data was processed successfully.</td>
</tr>
<tr>
<td>04</td>
<td>The SYSUT2 data set was reblocked.</td>
</tr>
<tr>
<td>08</td>
<td>The input data was not processed successfully.</td>
</tr>
<tr>
<td>16</td>
<td>An I/O error occurred.</td>
</tr>
</tbody>
</table>

GIMGTPKG service routine

The GIMGTPKG service routine is used to get a GIMZIP package from a remote FTP server in a TCP/IP network and store the package on a local z/OS host. GIMGTPKG performs the functions of the SMP/E RECEIVE FROMNETWORK TRANSFERONLY command, but does so independently of SMP/E. It uses FTP to transport the files of a GIMZIP package from a remote FTP server to a local host, thus it will provide the following capabilities (which are also available with the RECEIVE FROMNETWORK command):

- Industry standard FTP protocol.
- Secure transmission using the capabilities of the z/OS FTP client.
- Ensured integrity of the transported files.

GIMGTPKG requires input to define the FTP server and package to be obtained and to control local host processing. GIMGTPKG depends on the z/OS FTP client for its FTP operations and on either ICSF or Java 2 Version 1 Release 4 to perform SHA-1 hash calculations.
GIMGTPKG

GIMGTPKG is an independent load module residing in the MIGLIB library. It has an addressing mode of 31 (AMODE=31), a residence mode of 24 (RMODE=24), and requires no special authorization (AC=0). GIMGTPKG runs independently from the rest of SMP/E processing.

Calling GIMGTPKG

Following are the JCL statements required to call GIMGTPKG.

```jcl
//job JOB ...
//step EXEC PGM=GIMGTPKG,PARM='options'
//SMPOUT DD SYSOUT=*  
//SYSPRINT DD SYSOUT=* 
//SMPNTS DD PATH='package_directory',PATHDISP=KEEP 
//SMPCPATH DD PATH='smpclasses_directory', PATHDISP=KEEP 
//SMPJHOME DD PATH='javaruntime_directory', PATHDISP=KEEP 
//SMPSRVR DD *  
  server tag and attributes
/* 
//SMPCNT DD *  
  client tag and attributes
/*
```

Figure 88. JCL to call GIMGTPKG

EXEC

is the statement used to call GIMGTPKG. The EXEC statement must specify PGM=GIMGTPKG. The following option may be specified on the EXEC statement PARM operand:

```
LANGUAGE=xxx
```

where xxx can be either ENU (US English) or JPN (Japanese). The LANGUAGE option defines which language to use for GIMGTPKG messages. LANGUAGE can also be specified as L. If LANGUAGE is not specified, the default is LANGUAGE=ENU.

SMPOUT

used for GIMGTPKG messages. SMPOUT may be sequential, a member of a partitioned data set, or a file in the UNIX file system. See "SMPOUT" on page 155 for information about allocating SMPOUT. GIMGTPKG will format messages to 80 characters in width, regardless of the LRECL for SMPOUT.

SYSPRINT

is used for FTP output. SYSPRINT may be sequential, a member of a partitioned data set, or a file in the UNIX file system. See "SYSPRINT" on page 167 for information about allocating SYSPRINT. GIMGTPKG will format output to 80-characters in width, regardless of the LRECL for SYSPRINT.

SMPNTS

specifies a directory in the UNIX file system where the package to be obtained is to be stored.

SMPCPATH

specifies a directory in the UNIX file system where the SMP/E Java classes reside. The SMPCPATH DD statement is optional and only required to calculate SHA-1 hash values if ICSF is not available.

SMPJHOME

specifies a directory in the UNIX file system where the Java runtime resides. The SMPJHOME DD statement is optional and only required to calculate SHA-1 hash values if ICSF is not available.

---

452  SMP/E V3R6.0 for z/OS V1R13.0 Reference
GIMGTPKG

SMPSRVR

provides the necessary information about the FTP server from which a package
is to be obtained and about the package itself. This package and FTP server
information is specified with the <SERVER> tag. For information about using
the <SERVER> tag, see SMP/E Commands.

SMPSRVR may be sequential, a member of a partitioned data set, or a file in
the UNIX file system. See “SMPSRVR” on page 160 for information about
allocating SMPSRVR.

SMPCLNT

provides information about the local FTP client host, such as information about
local firewall navigation. The local FTP client host information is specified with
the <CLIENT> tag. For information about using the <CLIENT> tag, see SMP/E
Commands.

SMPCLNT is an optional DD statement. SMPCLNT may be sequential, a
member of a partitioned data set, or a file in the UNIX file system. See
“SMPCLNT” on page 143 for information about allocating SMPCLNT.

Example of using GIMGTPKG

```
//job JOB ...
//step EXEC PGM=GIMGTPKG
//SMPOUT DD SYSOUT=*...
//SYSPRINT DD SYSOUT=*...
//SMPSRVR DD *
<SERVER host="host.sample.com"
   user="usr01"
   pw="n0peekng">
   <PACKAGE file="CBPROC/O12345/RIMTAPE/GIMPAF.XML"
             hash="1234567890123456789012345678901234567890"
             id="012345">
   </PACKAGE>
</SERVER>
/*
//SMPCLNT DD *
<CLIENT retry="3">
</CLIENT>
/*
```

Figure 89. GIMGTPKG example

In this example, a GIMZIP package residing on the host.sample.com FTP server is
transferred onto the local z/OS host and stored in the /u/ usr01/pkgs/ directory.
The <SERVER> tag and attributes in the SMPSRVR data set describe the FTP
server and the package to be obtained. The <CLIENT> tag and attributes in the
optional SMPCLNT data set indicate FTP operations should be retried a maximum
of three times if necessary in order to obtain full and accurate package files.

GIMGTPKG processing

GIMGTPKG checks the required and optional DD statements that were specified,
opens the data sets, and ensures they have the correct attributes.
The remainder of GIMGTPKG processing is much like that for the RECEIVE FROMNETWORK command with TRANSFERONLY (see the RECEIVE Command chapter in SMP/E Commands). Briefly, this processing includes the following activities:

- The SMPSRVR and SMPCLNT data sets are read and analyzed. For comparison to the RECEIVE FROMNETWORK command, SMPSRVR is analogous to the SERVER data set and SMPCLNT is analogous to the CLIENT data set.
- The package directory is a specific subdirectory of the SMPNTS directory and is the destination on the local host for the GIMZIP package files to be stored. The name of this subdirectory is the package-id value from the SMPSRVR data set. If the package directory does not already exist, then it is created.
- Each of the files in the GIMZIP package on the FTP server are retrieved and stored in the package directory created in the previous step. Each file is verified by computing an SHA-1 hash value for the file.
- After all files have been transferred successfully, then, if any archive files in the package have been segmented, the segments are combined to create complete archive files.

**Return codes**

GIMGTPKG may complete with any of the following return codes.

<table>
<thead>
<tr>
<th>Return code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>GIMGTPKG processing completed successfully.</td>
</tr>
<tr>
<td>12</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• Required data sets were missing or could not be opened.</td>
</tr>
<tr>
<td></td>
<td>• Required modules could not be loaded.</td>
</tr>
<tr>
<td></td>
<td>• Syntax error was found in the SMPSRVR or SMPCLNT tags.</td>
</tr>
<tr>
<td></td>
<td>• An FTP error was detected.</td>
</tr>
<tr>
<td></td>
<td>• Neither ICSF or Java 2 Version 1 Release 4 is available.</td>
</tr>
<tr>
<td>16</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• An I/O error occurred.</td>
</tr>
<tr>
<td></td>
<td>• Syntax error was found on the EXEC statement parameters.</td>
</tr>
<tr>
<td>20</td>
<td>The SMPOUT data set is missing.</td>
</tr>
</tbody>
</table>

**GIMUNZIP file extraction service routine**

The GIMUNZIP service routine is used to extract data sets, files, and directories from archive files in GIMZIP packages created by the GIMZIP service routine. These packages typically contain software and associated materials in the form of SYSMODs, RELFILE data sets, HOLDDATA, and other materials such as documentation, samples, and text files. These GIMZIP packages may be transported through a network, processed by the GIMUNZIP service routine, and then processed by the SMP/E RECEIVE command.

More specifically, the GIMUNZIP service routine extracts data sets, files, and directories from the archive files that compose the GIMZIP package. An archive file consists of a portable image of a sequential, partitioned, or VSAM data set, or a file or directory in a UNIX file system, and the information needed to create that data set, file, or directory from the portable image. The data set, file, or directory into which the archive file is to be extracted can already exist or GIMUNZIP can create a new one of the appropriate type. New sequential and partitioned data sets created by GIMUNZIP are always catalogued.
Notes:

1. GIMUNZIP is a separate load module residing in the MIGLIB library and runs independently from the rest of SMP/E processing.
2. GIMUNZIP optionally requires either the Integrated Cryptographic Services Facility (ICSF) One-Way Hash Generate callable service or Java Version 1 Release 4 to be available for its use in order to compute an SHA-1 hash value.

Calling GIMUNZIP

The following figure shows the job control statements for GIMUNZIP:

```
//JOBx JOB ...
//STEP1 EXEC PGM=GIMUNZIP,PARM='options'
//SMPDIR DD PATH='package_directory',PATHDISP=KEEP
//SMPWKDIR DD PATH='work_directory',PATHDISP=KEEP
//SMPCPATH DD PATH='smpclasses_directory',PATHDISP=KEEP
//SMPJHOME DD PATH='javaruntime_directory',PATHDISP=KEEP
//SMPOUT DD SYSOUT=* 
//SYSPRINT DD SYSOUT=* 
//SYSUT3 DD UNIT=SYSALLDA,SPACE=(CYL,(50,10))
//SYSUT4 DD UNIT=SYSALLDA,SPACE=(CYL,(25,5))
//SYSPRINT DD SYSOUT=*
package control statements
/*
```

Figure 90. JCL to call GIMUNZIP

**EXEC**

is the statement used to call GIMUNZIP. The EXEC statement must specify PGM=GIMUNZIP. The following options may be specified on the EXEC statement PARM operand:

**LANGUAGE=xxx**

where xxx can be one of the following:

- ENU US English
- JPN Japanese

The LANGUAGE option defines the language to use for GIMUNZIP messages. LANGUAGE can also be specified as L. If LANGUAGE is not specified, the default is LANGUAGE=ENU.

**HASH=xxx**

where xxx can be one of the following:

- YES indicates GIMUNZIP is to perform SHA-1 hash checking for the archive files specified in the SYSIN data set.

  **Note:** GIMUNZIP requires either the Integrated Cryptographic Services Facility (ICSF) One-Way Hash Generate callable service or Java 2 Version 1 Release 4 to be available in order to compute an SHA-1 hash value.

  NO indicates GIMUNZIP is not to perform SHA-1 hash checking for the archive files specified in the SYSIN data set.

  HASH can also be specified as H. If HASH is not specified, the default is HASH=NO.

**SYSIN**

specifies a control data set that contains the package control tags to control GIMUNZIP processing (see “GIMUNZIP package control tags” on page 456).
The control data set may be a sequential data set or a single member of a partitioned data set, and its attributes must be LRECL=80 and RECFM=F or FB.

**SMPDIR**

specifies a directory in a UNIX file system that contains a GIMZIP package. This directory is referred to as the package directory.

**SMPWKDIR**

specifies a directory in a UNIX file system that is used by GIMUNZIP for temporary work files. This is an optional DD statement. If the SMPWKDIR DD statement is not provided, GIMUNZIP will use the package directory specified on the SMPDIR DD statement for temporary work files.

**SMPCPATH**

specifies a directory in the UNIX file system where the SMP/E Java classes reside. The SMPCPATH DD statement is optional and only required to calculate SHA-1 hash values if ICSF is not available.

**SMPJHOME**

specifies a directory in the UNIX file system where the Java runtime resides. The SMPJHOME DD statement is optional and only required to calculate SHA-1 hash values if ICSF is not available.

**SMPOUT**

specifies an output data set that will contain SMP/E messages issued during GIMUNZIP processing. The data set attributes are LRECL=81 or LRECL=121 and RECFM=FBA.

**SYSUT3 and SYSUT4**

specifies work areas for the IEBCOPY utility to use for spill data sets on DASD.

**SYSPRINT**

specifies a data set to contain output and messages from the utilities called by GIMUNZIP.

**GIMUNZIP package control tags**

GIMUNZIP service routine processing is controlled by package control tags. The package control tags identify the archive files to be extracted by GIMUNZIP and are specified in the SYSIN data set. The package control tags follow XML syntax rules and their format is as follows:
GIMUNZIP package control tags

GIMUNZIP:

<table>
<thead>
<tr>
<th>ARCHDEF</th>
</tr>
</thead>
</table>

ARCHDEF:

<table>
<thead>
<tr>
<th>name=&quot;archive name&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>archid=&quot;archive id&quot;</td>
</tr>
<tr>
<td>volume=&quot;data set volume&quot;</td>
</tr>
</tbody>
</table>

| newname="data set name" |
| prefix="data set prefix" |
| replace="yes" |
| preserveid="yes" |

<ARCHDEF> Tag syntax

The <ARCHDEF> and corresponding </ARCHDEF> tags identify the beginning and end of a specific archive retrieval request. The following attributes may be found on the <ARCHDEF> tag:

name="archive name"

specifies the path name for an archive file to be extracted by GIMUNZIP. The path name may include a subdirectory name (provided by the subdir attribute when the archive was created by GIMZIP). The path name value is a relative value and it is relative to the package directory specified on the SMPDIR DD statement. The name attribute is mutually exclusive with the archid attribute.

Note: If the archive file has been segmented in the package, you should specify the name of the archive file, not the names of the archive segment files. GIMUNZIP will combine the associated archive segment files to extract the complete archive.

archid="archive id"

specifies a unique archive id associated with an archive file to be extracted by GIMUNZIP. GIMUNZIP will search the package attribute file looking for the archive file with this unique archid. The archid attribute is mutually exclusive with the archive name attribute.

volume="data set volume | *"

specifies the volume serial number of the volume on which GIMUNZIP is to request the allocation of the data set to be extracted from the archive file. The volume identifier must be from 1 to 6 alphanumeric characters or an asterisk (*).
GIMUNZIP

Notes:

1. If you are extracting a sequential, partitioned, or VSAM data set into an existing cataloged data set, you should not specify the *volume* attribute.

2. If you are extracting a sequential or partitioned data set:
   - Into a new data set to be allocated on a specific volume, or into an existing, uncataloged data set on a specific volume, you must specify `volume=data set volume`. If you specify `volume=*` it is ignored.
   - Into an existing, uncataloged data set on a specific volume, you must specify `volume=data set volume`. If you specify `volume=*` it is ignored.

   The data set is first allocated with a disposition of (OLD,KEEP) using the specified volume. If the data set is not on the volume, the data set is allocated again with a disposition of (OLD,KEEP) without specifying a volume, to see if there is an existing cataloged data set that should be used for the extraction. If an existing cataloged data set is allocated, the archive is extracted into the cataloged data set, even though the volume on which it is allocated is different from the volume specified.

3. If you are extracting a VSAM data set:
   - Into a new data set, you can specify `volume=data set volume`,
     Or,
     You can specify `volume=*`. GIMUNZIP uses the value on the VOLUMES parameter on the IDCAMS DEFINE command that GIMUNZIP will generate to create the new VSAM data set. Using the VOLUMES(*) parameter on the generated IDCAMS DEFINE command may allow SMS to assign a volume for the new VSAM data set if the automatic class selection (ACS) routines allow it.
   - Into an existing destination data set, a data set volume you specify on the *volume* attribute is ignored.

   Since VSAM data sets must be cataloged, the catalog is checked first to see if the VSAM destination data set already exists.

4. If you specify `volume=data set volume`, GIMUNZIP will use the specified volume to dynamically allocate work space to the SYSUT1 ddname during the processing of the archive. If you do not specify the *volume* attribute or you specify `volume=*`, then you must make available at least one volume of mounted storage or the allocation to the SYSUT1 ddname will fail.

5. If you are extracting a UNIX file or directory, you do not need to specify the *volume* attribute. It is ignored. However, if you specify a *volume* attribute, it must be syntactically correct.

`newname="name"`

specifies the data set name or absolute pathname to use for the data set or file to be extracted from the archive file. This name replaces the original name of the data set or file that is recorded in the archive's file attribute file.

The `newname` specified must identify the same type of data as the file being unzipped. That is, if the file being unzipped is a data set, `newname` must represent a data set. If the file being unzipped is a file or directory in the UNIX file system, `newname` must represent a file or directory in the UNIX file system.

The first character of an absolute pathname of a file or directory in the UNIX file system must be a slash (/). If the `newname` value does not begin with a slash, GIMUNZIP processing will assume that it is an MVS data set name and that it will conform to the MVS data set naming conventions. If the `newname` value does begin with a slash, it is assumed to be a file or directory in the
UNIX file system. When a file or directory in the UNIX file system is specified, the `newname` value can be from 1 to 1023 bytes long with 255 characters between delimiters (/). The value can be any character from X'40' through X'FF', except less than (<) and greater than (>) signs, ampersands (&), and double quotation marks ("). All data beyond column 72 is ignored, including blanks. The pathname of a file or directory in the UNIX file system is case sensitive and will not be converted to uppercase alphabetic during processing of the GIMUNZIP package control tags.

The `newname` specified may identify an existing data set, directory, or file. In this case, the archived file is unzipped into the existing data structure specified on the `newname` attribute. If the `newname` attribute is not specified for a file or directory in the UNIX file system, the original name for the file or directory is used. If neither the `prefix` nor the `newname` attributes are specified when a data set is being extracted, the original name for the data set is used.

`prefix="data set prefix"` specifies a data set prefix to use for the data set to be extracted from the archive file. This prefix replaces the high-level qualifier of the original name for the data set as recorded in the archive's file attribute file. The specified data set prefix can be up to 26 characters long and must conform to standard data set naming conventions.

The resolved name using the `prefix` attribute may be for an existing data set. In this case, the archived data is extracted into the existing data set specified using the `prefix` attribute.

A prefix value has no meaning when the file being unzipped is a file or directory in the UNIX file system. In this case, the prefix value is parsed but ignored.

If neither the `prefix` nor the `newname` attributes are specified, the original name for the data set is used.

`replace="YES | NO"` indicates whether data in an existing data set, file or directory should be replaced by data from the archive file. A value of `YES` indicates the data in the existing data set, file or directory should be replaced. A value of `NO` is equivalent to not specifying the replace attribute and indicates the data should not be replaced. The `replace` attribute is not meaningful when the data set, file or directory does not already exist.

- For an archive of a partitioned data set or directory in the UNIX file system, `replace="YES"` indicates that existing members in a partitioned data set and files in a directory will be replaced by members or files with the same name from the archive file. If `replace="NO"` is specified, or if the replace attribute is not specified at all, existing members and files will not be replaced.
- For an archive of a sequential data set or a file in the UNIX file system, `replace="YES"` indicates the existing data set or file and its attributes will be replaced with the data from the archive file. If `replace="NO"` is specified, or if the replace attribute is not specified at all, the existing sequential data set or file will not be replaced with the data from the archive file.

**Note:** GIMUNZIP will not check if a sequential data set or file is empty. If the data set or file exists, it will not be replaced unless `replace="YES"` is specified.

- For a VSAM data set (cluster), `replace="YES"` indicates that an existing VSAM cluster should be populated with the data from the archive file. If
replace="NO" is specified, or if the replace attribute is not specified at all, there will be no attempt to populate the existing VSAM cluster with the data from the archive.

Notes:
1. The existing VSAM cluster must be of the same type as the archived cluster (ESDS, KSDS, LDS, or RRDS).
2. The existing VSAM cluster must have characteristics that are compatible with the archived cluster (such as, record size, key size, and key offset). GIMUNZIP does not verify the compatibility of these characteristics.
3. To replace the contents of an existing cluster, the cluster is altered to a reusable state by using an IDCAMS ALTER command, if necessary, before the data from the VSAM archive is copied into the cluster by using an IDCAMS REPRO command. The REPRO command will use both the REPLACE and REUSE operands. Following the REPRO operation, the cluster is altered back to a nonreusable state if that was its state to begin with.

preserveid="YES | NO"

indicates whether the UID and GID of the extracted file or directory should:
- Remain as defined when the archive was created, or
- Inherit the UID and GID of the userid performing the extract operation (GIMUNZIP).

A value of YES indicates that the UID and GID from when the archive was created will be preserved. A value of NO is equivalent to not specifying the preserveid attribute and indicates that the UID and GID should be inherited from the userid performing the extract operation (GIMUNZIP).

Notes:
1. Preserving the UID and GID defined when the archive was created may cause GIMUNZIP to fail if the installer does not have the proper permissions.
2. The preserveid attribute applies only when extracting a UNIX file or directory from an archive. When extracting a data set, preserveid is parsed for syntax but otherwise ignored.

Syntax notes
1. GIMUNZIP ignores columns 73 through 80. If data is specified beyond column 72, GIMUNZIP ignores it, which may lead to the diagnosis of an error in a following tag.
2. Package control tags may contain comments. Comments start with <!-- (hex 4C5A6060) and end with --> (hex 60606E). The first --> encountered after the initial <!-- will end the comment. A comment may appear between a start-tag and its matching end-tag, but never within a tag.

Example of using GIMUNZIP
Suppose a GIMZIP package contains a set of archive files and you want to extract the data sets and files from those archive files. In addition, you wish to verify the integrity of the archive files and rename the destination data sets and files in the process. The following job stream can be used to perform such an operation.
GIMUNZIP processing

The GIMUNZIP service routine extracts data sets and files from the archive files that compose the GIMZIP package. An archive file consists of a portable image of a data set or file and the information needed to reload the data from the portable image (see "Archive files" on page 492 for details on archive file contents and construction).

GIMUNZIP uses the UNIX System Services `pax` command to expand the following component files from an archive file temporarily into a UNIX file system:

1. the portable image of the original data set or file, and
2. the file attribute file that contains the information necessary to reload the data from the archived data set or file.

GIMUNZIP then reads the file attribute file and uses the information recorded there, along with information specified on the `<ARCHDEF>` tag, such as `volume`, `newname`, or `prefix`, to allocate a new or existing data set or file. The portable image of the original data set or file is stored in the data set or file just allocated for that purpose.
When GIMUNZIP encounters a package that contains archive segments, GIMUNZIP reassembles the archive segments into the original archive form, optionally verifies the integrity of the archive, and recreates the original data.

If the HASH= YES option was specified, then GIMUNZIP performs SHA-1 hash checking for the archive files. Specifically, GIMUNZIP reads the package attribute file for the package. The package attribute file, GIMPAF.XML, is found in the package directory, and contains the known hash values for the archives. These hash values were recorded by GIMZIP when the package and archives were created. GIMUNZIP then uses either ICSF services or Java 2 programs to compute the current hash values for the archive files, and compares this value to the known hash values for the archive files.

If the computed hash value matches the known hash value, then the integrity of the archive file is ensured. However, if the computed hash value does not match the known hash value, GIMUNZIP processing stops. This condition indicates the archive file has been corrupted since it was produced by GIMZIP when the package was created.

Determining the required size of SMPWKDIR

SMPWKDIR is a directory in the UNIX file system used by GIMUNZIP for storing temporary work files. If this working directory is not identified by an SMPWKDIR DD statement, GIMUNZIP will use the package directory specified on the SMPDIR DD statement for storing temporary work files. In either case, the required size for the working directory varies depending on the type of information contained within an archive, and whether or not the archive is segmented. If an archive file is segmented, the file name of each segment will have a file type that identifies the segment position and the number of segments that make up the archive. For example, if an archive file is split into three segments, the segment files will have file types of 1of3, 2of3 and 3of3 in the package directory. Additionally, in the Package Attribute File, each segment will be identified by an <ARCHSEG> tag and matching </ARCHSEG> end tag. The required working directory size can be calculated as the maximum of the following:

• the archive size for the largest segmented UNIX file system data set archive. That is, the sum of the size of all the segments. All the segments are copied from SMPDIR into the working directory to create a complete archive file. Using the z/OS UNIX pax command, the archive is expanded into its end point location.
• twice the original data set size for the largest segmented PDSE, VSAM or sequential data set archive. All the segments are copied from SMPDIR into the working directory to create a complete archive file. Using the z/OS UNIX pax command, the archive is expanded into a file in the working directory. The end point data set is then loaded from the contents of the expanded file.
• the original data set size for the largest unsegmented PDSE, VSAM or sequential data set archive. Using the z/OS UNIX pax command, the archive in SMPDIR is expanded into a file in the working directory. The end point data set is then loaded from the contents of the expanded file.

Sample RECEIVE job for GIMUNZIP output

After GIMUNZIP has been run, the SMP/E RECEIVE command can be used to process the data sets and files extracted by GIMUNZIP. Here is a sample RECEIVE job that would process the SMPMCS and RELFILE data sets and files extracted by GIMUNZIP in the sample in Figure 2:
Return codes

GIMUNZIP may end with the following return codes:

<table>
<thead>
<tr>
<th>Return code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>The input data was processed successfully.</td>
</tr>
<tr>
<td>04</td>
<td>A call to a system service may not have completed successfully.</td>
</tr>
<tr>
<td>12</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• Required file attributes could not be obtained.</td>
</tr>
<tr>
<td></td>
<td>• Input files were not sequential, partitioned, or VSAM data sets,</td>
</tr>
<tr>
<td></td>
<td>or files and directories in the UNIX file system.</td>
</tr>
<tr>
<td></td>
<td>• Call to a required system service failed.</td>
</tr>
<tr>
<td></td>
<td>• Syntax errors in the SYSIN data set.</td>
</tr>
<tr>
<td></td>
<td>• Data sets could not be opened.</td>
</tr>
<tr>
<td></td>
<td>• SMPDIR was not allocated to a UNIX directory.</td>
</tr>
<tr>
<td></td>
<td>• Data sets, files, or directories were missing.</td>
</tr>
<tr>
<td></td>
<td>• Required modules could not be loaded.</td>
</tr>
<tr>
<td></td>
<td>• Existing data set, file, or directory is not compatible.</td>
</tr>
<tr>
<td></td>
<td>• Existing data set, file, or directory could not be replaced.</td>
</tr>
<tr>
<td>16</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• An I/O error occurred.</td>
</tr>
<tr>
<td></td>
<td>• A syntax error was found on the EXEC statement parameters.</td>
</tr>
<tr>
<td>20</td>
<td>SMPOUT data set is missing.</td>
</tr>
<tr>
<td>&gt; 20</td>
<td>Internal error. Report the error to the IBM Support Center.</td>
</tr>
</tbody>
</table>

GIMXSID software inventory data service routine

GIMXSID is an SMP/E service routine to be used as part of the ShopzSeries offering. GIMXSID creates a single data source required by ShopzSeries to place customized software product and service orders. The data source created by GIMXSID, the software inventory data, is a composite of three kinds of information as follows:

Feature list

a list of FEATUREs found in the SMPCSI data sets. The Feature List is used by ShopzSeries to perform product requisite checking and also to prime the order checklist when ordering a ServerPac.

FMID list

a list of the FMIDs found in the SMPCSI data sets. The FMID List is used by ShopzSeries to scope service orders to the PTFs applicable solely to the user’s desired configuration of target and global zones.
PTF bitmap

a bitmap representation of the PTFs found in the specified target zones and
global zones. The PTF Bitmap is used by ShopzSeries to produce service
packages that do not contain PTFs that are already present in the user's
configuration.

GIMXSID is an independent load module residing in the MIGLIB library. It has an
addressing mode of 31 (AMODE=31), a residence mode of 24 (RMODE=24), and is
not authorized (AC=0). GIMXSID runs independently from the rest of SMP/E
processing.

Calling GIMXSID

The JCL statements needed to call GIMXSID are:

```plaintext
//job    JOB ...
//step   EXEC PGM=GIMXSID,PARM='options'
//SMPOUT DD SYSOUT=*  
//SMPTOUT DD DSN=output.dataset,DISP=(NEW,CATLG),     
// SPACE=(TRK,(1,1)),UNIT=SYSALLDA
//SYSIN   DD *
/*
input control statements
*/
```

Figure 93. JCL to call GIMXSID

EXEC

is the statement used to call GIMXSID. The EXEC statement must specify
PGM=GIMXSID. The following options may be specified on the EXEC
statement PARM operand:

```
LANGUAGE=ENU | JPN
```

The LANGUAGE option defines which language to use for GIMXSID
messages. ENU is for US English, and JPN is for Japanese. LANGUAGE
can also be specified as L. If LANGUAGE is not specified, the default is
LANGUAGE=ENU.

```
WAIT=minutes_MIN
```

where minutes is a decimal number between 0 and 9999. The WAIT option
is used to control how long GIMXSID should wait if an SMPCSI data set is
unavailable because it is in use by another SMP/E task.

```
WAIT=minutes_MIN
```

causes GIMXSID to wait for a data set for the specified
number of minutes. If a data set is still not available after the specified
number of minutes, then GIMXSID stops. If WAIT is not specified, the
default is WAIT=60MIN.

```
WAIT may be specified as WAIT=minutes_MIN or WAIT=minutes. The MIN
keyword is optional.
```

SMPOUT

used for GIMXSID messages. The RECFM must be FBA and the LRECL must
be 81 or 121. GIMXSID formats messages to 80 characters in length, regardless
of the LRECL for SMPOUT.

SMPXTOUT

t he output specification for the software inventory data collected by GIMXSID.
The output can be a sequential data set, a member of a partitioned data set, or
a file in a UNIX file system. If a data set, the RECFM must be FB and the
GIMXSID

LRECL must be 12560. If a file in a UNIX file system, then FILEDATA=BINARY and PATHOPTS(OWRONLY) must be specified on the DD statement.

**SYSIN**

specifies an input data set that contains the GIMXSID control statements. The input data set may be sequential, or a member of a partitioned data set. The RECFM must be F or FB and the LRECL must be 80.

**GIMXSID control statements**

Following are the control statements used to direct the processing for GIMXSID. These control statements are specified in the SYSIN data set.

**CSI=dsname**

indicates the data set name for an SMPCSI data set containing a global zone. This is a required control statement. There can be multiple CSI statements to identify multiple SMPCSI data sets and global zones to be processed.

*dsname* can be up to 44 characters long and must conform to standard data set naming conventions

**TARGET=zone, zone, ...**

indicates the target zones controlled by the global zone identified on the preceding CSI statement that are to be processed. There can be multiple instances of the TARGET statement to allow many target zones to be specified. The *zone* values can be from 1 to 7 characters and must be separated by a comma and zero or more blank characters, or, one or more blank characters.

This is an optional control statement. If a TARGET statement is not specified following a CSI control statement, all target zones controlled by the global zone identified on the preceding CSI statement are processed.

**Syntax notes**

1. Control statements must each start on a new line.
2. Control statements can begin in any column of a line.
3. Control statements must end on the same line on which they begin (control statements cannot span records).
4. SMP/E ignores columns 73 through 80 of lines in the SYSIN data set.

**Example of using GIMXSID**

```
//job     JOB...
//BITMAP  EXEC PGM=GIMXSID
//SMPOUT  DD SYSOUT=*  
//SMPXTOUT DD DSN=userid.GIMXSID.OUTPUT,DISP=(NEW,CATLG),  
//          SPACE=(TRK,(1,1)),UNIT=SYSALLDA
//SYSIN    DD *
CSI=SMPE.ZOS.GLOBAL.CSI
TARGET=ZOS14, JES314
CSI=SMPE.DB2.GLOBAL.CSI
/*
```

*Figure 94. GIMXSID example*

In this example, the ZOS14 and JES314 target zones are processed from the global zone in the SMPE.ZOS.GLOBAL.CSI data set. In addition, all target zones from the global zone in the SMPE.DB2.GLOBAL.CSI data set are processed.*
Processing

There are several phases of GIMXSID processing described below in detail.

Initialization
GIMXSID first verifies that the SMPOUT data set, which is used for messages, is available. If SMPOUT is not allocated or cannot be opened, there is nowhere to write a message. Therefore, GIMXSID terminates with a return code of 20.

GIMXSID then checks for the required data sets, opens these data sets, and ensures they have the correct attributes.

Read and verify control statements
After the required data sets have been verified, GIMXSID reads the input control statements from the SYSIN data set. The CSI control statement is a required control statement and must precede any TARGET control statements. For each data set specified on a CSI control statement, GIMXSID ensures that the data set exists.

Verifying the target zones
For each SMPCSI data set specified on a CSI control statement, a list of target zones may be specified on one or more TARGET control statements. If target zones are specified, GIMXSID ensures that the specified zones exist and are in fact target zones.

Accessing resources
When reading SMPCSI data sets, it is possible one or more data sets are unavailable for use by GIMXSID because the data sets are being used by another SMP/E task. If so, GIMXSID waits for the data sets to become available based on the value of the WAIT option (see “Calling GIMXSID” on page 464).

If a global zone data set or a target zone data set is not available, then GIMXSID will wait until the data set is available, or until the number of minutes specified on the WAIT option have elapsed (or 60 minutes by default). During the WAIT time, GIMXSID attempts to read the data set every 30 seconds, just in case it becomes available. If the specified number of minutes elapses and one or more of the data sets are still not available, processing stops.

If WAIT=0MIN was specified, GIMXSID does not wait for SMPCSI data sets to become available. The initial attempt is made to access the data sets, and if they are not available, GIMXSID does not wait or make additional access attempts.

The number of minutes specified on the WAIT option (or defaulted) is the total time GIMXSID waits for all data sets. For example, if WAIT=60MIN was specified and the first SMPCSI data set causes a wait of 40 minutes, GIMXSID waits only a maximum of 20 additional minutes for all subsequent SMPCSI data sets.

Building the GIMXSID record
To identify itself as the creator of the software inventory data, GIMXSID creates a GIMXSID record and writes it to the output data set. This GIMXSID record is the first record in the output data set, but will be written only if there is software inventory data to produce (that is, if there are other records to be written). This GIMXSID record provides the name and service level of the program that produced the software inventory data, as well as the date and time it was produced. Table 57 on page 467 shows the format of the GIMXSID record.
Table 57. GIMXSID record format

<table>
<thead>
<tr>
<th>Field</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>60</td>
<td>The value is &quot;GIMXSID&quot; padded to the right with blanks, to indicate the record type, and to identify the name of the program used to create the software inventory data.</td>
</tr>
<tr>
<td>Version</td>
<td>2</td>
<td>Two character representation of the version level of GIMXSID.</td>
</tr>
<tr>
<td>Release</td>
<td>2</td>
<td>Two character representation of the release level of GIMXSID.</td>
</tr>
<tr>
<td>Modification</td>
<td>2</td>
<td>Two character representation of the modification level of GIMXSID.</td>
</tr>
<tr>
<td>PTF</td>
<td>3</td>
<td>Two character representation of the PTF level of GIMXSID, padded to the right with a blank.</td>
</tr>
<tr>
<td>Date</td>
<td>9</td>
<td>The date the software inventory data was created, padded to the right with a blank. The date value is of the form mm/dd/yy.</td>
</tr>
<tr>
<td>Time</td>
<td>8</td>
<td>The time the software inventory data was created. The time value is of the form hh:mm:ss.</td>
</tr>
<tr>
<td>Reserved</td>
<td>12474</td>
<td>Reserved field, padded with blanks.</td>
</tr>
</tbody>
</table>

Building the feature list

The Feature List segment of the software inventory data contains the combined Feature information from all specified SMPCI data sets. For each SMPCI data set specified on a CSI control statement:

- If target zones were specified on one or more TARGET control statements for an SMPCI data set, then the Feature List for that SMPCI contains all FEATURE subentries found in FUNCTION SYSMOD entries from all specified target zones.
- If no target zones were specified for an SMPCI data set, then the Feature List for that SMPCI contains the name of all FEATURE entries found in the global zone, in addition to all FEATURE subentries found in FUNCTION SYSMOD entries from all target zones in the SMPCI data set.

No regard is paid to the ERROR status of the FUNCTION SYSMOD entries; if a FUNCTION SYSMOD entry happens to be in ERROR, then it is assumed an overt attempt was made to install the FUNCTION and therefore its presence is desired. However, if a FUNCTION SYSMOD has been superseded or deleted, then it is ignored.

The individual Feature Lists from each specified SMPCI are then merged to create the total Feature List. That is, the total Feature List set is the union of the individual sets.

The total Feature List is represented in the output data set using the definition for FEATURE records. Multiple FEATURE records may be produced in order to contain the total FEATURE information.

Table 58. FEATURE record format

<table>
<thead>
<tr>
<th>Field</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>8</td>
<td>Value is &quot;FEATURE&quot; to indicate the record type, padded to the right with blanks.</td>
</tr>
<tr>
<td>Feature Name</td>
<td>52</td>
<td>Feature name, padded to the right with blanks.</td>
</tr>
</tbody>
</table>
Table 58. FEATURE record format (continued)

<table>
<thead>
<tr>
<th>Field</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product-id</td>
<td>9</td>
<td>The product identifier of the feature’s associated product, padded to the right with blanks. For IBM products, this field is an IBM program product number (5647-A01, for example). This field may be all blank if the FEATURE entry is not found in a global zone in an SMPCSI data set.</td>
</tr>
<tr>
<td>VRM</td>
<td>9</td>
<td>The version, release, and modification level of the feature’s associated product, padded to the right with blanks. This field is of the form vv.rr.mm where vv is the version, rr is the release, and mm is the modification level. This field may be all blank if the FEATURE entry is not found in a global zone in an SMPCSI data set.</td>
</tr>
<tr>
<td>reserved</td>
<td>5</td>
<td>This field is reserved for future use.</td>
</tr>
<tr>
<td>Description</td>
<td>65</td>
<td>A text description for the feature, padded to the right with blanks. This field may be all blank if the FEATURE entry is not found in a global zone in an SMPCSI data set.</td>
</tr>
<tr>
<td>FMID List</td>
<td>12412</td>
<td>List of function SYSMODs that make up this feature. This field is a list of 7-byte FMID names separated by one blank space, padded to the right with blanks. This field may be all blank if the FEATURE entry is not found in a global zone in an SMPCSI data set, or the FMID subentry is not found for the FEATURE entry.</td>
</tr>
</tbody>
</table>

A single FEATURE may be represented in the output data set with more than one FEATURE record. Although an unlikely case, it is necessary only to accommodate a FEATURE that contains more FMIDs than can fit in the FMID List field of a single FEATURE record.

**Building the FMID list**

The FMID List segment of the software inventory data contains the combined FMID information from all specified SMPCSI data sets. For each SMPCSI data set specified on a CSI control statement:

- If target zones were specified on one or more TARGET control statements for an SMPCSI data set, then the FMID List for that SMPCSI contains the names of all FUNCTION SYSMODs found in all specified target zones.
- If no target zones were specified for an SMPCSI data set, then the FMID List for that SMPCSI contains the names of all FUNCTION SYSMODs found in the global zone, in addition to the names of all FUNCTION SYSMODs found in all target zones for the SMPCSI data set.

No regard is paid to the ERROR status of the FUNCTION SYSMOD entries; if a FUNCTION SYSMOD entry happens to be in ERROR, then it is assumed an overt attempt was made to install the FUNCTION and therefore its presence should be noted. However, if a FUNCTION SYSMOD in a target zone has been superseded or deleted, then it is ignored.

The individual FMID Lists from each specified SMPCSI are then merged to create the total FMID List. That is, the total FMID List set is the union of the individual sets.
The total FMID List is represented in the output data set using the definition for PRODLIST records. Multiple PRODLIST records may be produced in order to contain the complete list of FMIDs.

Table 59. PRODLIST record format

<table>
<thead>
<tr>
<th>Field</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>60</td>
<td>Value is &quot;PRODLIST&quot; and padded to the right with blanks.</td>
</tr>
<tr>
<td>FMID List</td>
<td>12500</td>
<td>List of 7-byte FMID names separated by one blank space, padded to the right with blanks.</td>
</tr>
</tbody>
</table>

**Building the PTF bitmap**

The PTF Bitmap segment of the software inventory data contains a composite of PTF information from all specified SMPCSI data sets. The point of the PTF Bitmap is to identify all PTFs that are already "on-site" and therefore do not need to be included in a service order. That is, there is no need to deliver PTFs that are already applied in all target zones in which they could be applied, or are already in the global zones and could be applied to any target zone as needed.

For each SMPCSI data set specified on a CSI control statement, the individual zones to be processed when creating the PTF information are determined as follows:

- If target zones were specified on one or more TARGET control statements for an SMPCSI data set, then those target zones are the current set of zones to be processed.
- If no target zones were specified for an SMPCSI data set, then all target zones in the global zone, and the global zone itself, are the current set of zones to be processed.

For this discussion, a PTF is considered applicable to a target zone if the PTF's FMID is applied in that target zone. A PTF is considered applicable to a global zone if the PTF's FMID is received in the global zone. A PTF is considered applicable to an SMPCSI data set if the PTF is applicable to any of the zones in the current set, as determined above.

The PTF information for an individual SMPCSI data set contains all PTFs that are received in the global zone, or are applied in all target zones of the current set in which they are applicable. Such PTFs are deemed "present" in an SMPCSI data set. That is, such PTFs can be considered already present and therefore no need exists to deliver such PTFs on behalf of this SMPCSI, because they are either already in the global zone and can be applied to any target zone, or they are already applied in all target zones in which they could be applied.

A PTF is considered applied in a target zone whether or not it is superseded. A PTF is not considered applied in a target zone, or received in a global zone, if it's SYSMOD entry is in ERROR status. Further, it is not always possible to know the SYSMOD TYPE for a SYSMOD that has been superseded in a target zone.

Therefore, if the name of such a SYSMOD follows the IBM naming convention for PTFs, then it is assumed to be a PTF and its presence is noted. Specifically, if the first character of the name of such a SYSMOD is a character in the set "LMNOPQRSTU" and the last five characters of the name are numeric, then it is assumed to be a PTF.
The PTF information from each individual SMPCSI data set is combined to create the total PTF Bitmap. A PTF is included in the PTF Bitmap if and only if it is present in all SMPCSI data sets in which it is applicable. Stated another way, if a PTF is applicable to \( n \) SMPCSI data sets, but it is present in only \( n-1 \) or fewer SMPCSI data sets, then that PTF cannot be included in the PTF Bitmap.

The total PTF Bitmap is represented in the output data set using the definition for BITPTF records. Multiple BITPTF records may be produced in order to contain the total PTF Bitmap.

**Table 60. BITPTF record format**

<table>
<thead>
<tr>
<th>Field</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>6</td>
<td>Value is &quot;BITPTF&quot; to indicate the record type. BITPTF indicates the record contains a PTF identification bit-string.</td>
</tr>
<tr>
<td>PTF Prefix</td>
<td>54</td>
<td>A two-character PTF prefix value, padded to the right with blanks. The PTF prefix value is the first two alphabetic characters in a PTF name. For example, &quot;UR&quot; or &quot;UW&quot;.</td>
</tr>
<tr>
<td>Bit-String</td>
<td>12500</td>
<td>Bit-string where each bit represents a number from 00001 to 99999. The numbers from 00001 to 99999 correspond to the last five numeric characters in a PTF name. The position of a particular bit corresponds to a particular number from 00001 to 99999. This number, combined with the PTF prefix, represent a specific PTF name. For example, if the PTF prefix is &quot;UW&quot; and the 12345th bit is on, then the PTF with the name &quot;UW12345&quot; is represented.</td>
</tr>
</tbody>
</table>

**Zone and data set sharing considerations**

GIMXSID uses the system enqueue facility to control access to SMPCSI data sets. The following identifies the phases of GIMXSID processing and the zones and data sets GIMXSID may require for shared use during those phases.

1. **Initialization**
   - Global zones (as specified on CSI control statements) Read with shared enqueue.

2. **Processing**
   - Global zones (as specified on CSI control statements) Read with shared enqueue.
   - Target zones (as specified on TARGET control statements) Read with shared enqueue.

3. **Termination**
   - All resources are freed.

**Return codes**

GIMXSID may complete with any of the following return codes.

<table>
<thead>
<tr>
<th>Return code</th>
<th>Meaning</th>
</tr>
</thead>
</table>
**GIMXSID**

00  GIMXSID processing completed successfully.

12  One of the following:
    • Required data sets were missing or could not be opened.
    • Required modules could not be loaded.
    • Syntax error was found in the SYSIN control statements.
    • SMPCSI data sets were not available.
    • Specified zones were not defined or are not target zones.

16  One of the following:
    • An I/O error occurred.
    • Syntax error was found on the EXEC statement parameters.

20  The SMPOUT data set is missing.

---

**GIMXTRX service routine**

GIMXTRX is intended for use as part of an offering called ShopzSeries. It provides two basic functions:

**List target zone names (LSTTZN)**
Generate a list of target zone names associated with a given GLOBAL zone SMPCSI data set name.

**Create a BITMAP of an SMPCSI (BMPTZN)**
Generate a BITMAP representation of FUNCTION and PTF SYSMODs found in a given list of target zone names associated with a given GLOBAL zone SMPCSI data set name.

The input to GIMXTRX is the name of a data set that contains the input parameters. See "Input parameter data set contents" on page 472.

The results of each function is written to a data set created by GIMXTRX. See "Processing" on page 475.

Note: GIMXTRX is a separate load module residing in the MIGLIB library and runs independently from the rest of SMP/E, and is executable on a z/OS, OS/390, or MVS system; it will not run on VM.

---

**Calling GIMXTRX**

GIMXTRX has normal standard linkage and expects an input parameter that is a character string that is a data set name. Following are the JCL statements needed to call GIMXTRX:

```plaintext
//JOBx   JOB ...
//XTRX   EXEC  PGM=GIMXTRX,PARM='parmdsn'
//SYSTSPRT DD SYOUT=(*)
//SMPXTOUT DD DSN=outputsdsn,DISP=(NEW,CATLG),
//          UNIT=unit,SPACE=(TRK,(2,1))
```

*Figure 95. JCL to call GIMXTRX*

**EXEC**

is the statement used to call GIMXTRX. The EXEC statement must specify PGM=GIMXTRX. The following option must be specified on the EXEC statement PARM operand:

`parmdsn`

name of the data set that contains the input parameters required for
the GIMXTRX function. See "Input parameter data set contents" for a description of the data contained in the input parameter data set.

**SYSTSPRT**

is used by GIMXTRX for messages. The record format (RECFM) of SYSTSPRT must be FBA, and the record length (LRECL) must be 121. GIMXTRX will format messages to 120 characters in length.

**Note:** GIMXTRX messages are only issued in the U.S. English language.

**SMPXTOUT**

is the output data set. This DD statement is optional. If SMPXTOUT is specified, GIMXTRX writes the output to the identified data set. If SMPXTOUT is not specified, GIMXTRX allocates a new data set for the output. SMPXTOUT must be a sequential data set, the record format (RECFM) must be FB, and the record length (LRECL) must be 12560. If incorrect attributes are specified for SMPXTOUT, GIMXTRX will change the attributes to the required values.

### Input parameter data set contents

The input parameter data set is a fixed-block 80-byte record sequential data set. Each input parameter consists of a keyword and an associated value. The rules for the input parameter data set are as follows:

- There must be one keyword with its associated parameter data per 80-byte record.
- All keywords must begin in column one of the records.
- No blanks are allowed after the keyword and before the equal sign.
- No blanks are allowed after the equal sign and before the first parameter value.
- Any data read from the input parameter data set that is not a recognized keyword parameter is ignored and processing continues.

The keywords and parameters passed to the GIMXTRX program are as follows:
GIMXTRX input parameters

**SUFINC keyword:**

- **SUFINC=LSTTZN**
  - Generate a list of target zone names.
- **SUFINC=BMPTZN**
  - Create a BITMAP of a set of target zones.

**SUFGBL keyword:**

- **SUFGBL=globaldsn**

**SUFTGT keyword:**

- **SUFTGT=tzone name**

**SUFUNIT keyword:**

- **SUFTGT=unit**

**SUFVOL keyword:**

- **SUFVOL=volume**

**SUFINC**

This is the name of the function to be processed by the GIMXTRX program. The two allowable values are:

- **LSTTZN**
  - Generate a list of target zone names.
- **BMPTZN**
  - Create a BITMAP of a set of target zones.

This is a required parameter.

There can only be one function executed per invocation of GIMXTRX.

**SUFGBL**

This is the data set name containing the GLOBAL zone to be processed. This parameter is 44 characters long. This is a required parameter.

**SUFTGT**

This is the UNIT value used to allocate a new output data set. The value is up to 8 characters long. This parameter is optional. If no UNIT value is specified, GIMXTRX uses a default UNIT value of SYSALLDA when allocating a new output data set.

**SUFVOL**

This is the work volume used to allocate the output data set. The value is six characters long, and must be a valid MVS volume name. This is an optional parameter.
GIMXTRX

parameter. If no volume is specified, GIMXTRX will attempt to allocate the output data set without a volume serial.

SUFTGT
This is the list of target zone names that will be queried. Each target zone name is 7 characters long separated by one blank space. If a target zone name is less than 7 characters long it must be left-justified and padded with blanks. Each SUFTGT record can contain up to nine target zones. This is a required parameter when SUFFNC=BMPTZN. It is not required when SUFFNC=LSTTZN.

There can be multiple instances of the SUFTGT record to allow for situations where a single 80-byte record cannot contain the required input data.

Examples of using GIMXTRX
In the following examples, the input data set is created in the job stream. In the first example, the output of GIMXTRX is written to a new data set created by GIMXTRX.

```
//jobname JOB ...
/**
 * Create the input data set.
 */
//step1 EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=*
//SYSIN DD DUMMY
//SYSUT2 DD DSN=userid.dataset.INPUT,DISP=(NEW,CATLG),
//   DCB=(RECFM=FB,LRECL=80),
//   UNIT=unit,SPACE=(TRK,(1,0))
//SYSUT1 DD DATA,DLM=$$
SUFFNC=BMPTZN
SUFGBL=userid.GLOBAL.CSI
SUFTGT=ZOSTGT
$$
//step2 EXEC PGM=GIMXTRX,PARM='userid.dataset.INPUT'
/**
 * The output Bit-Map is written to a new data set
 * allocated with the name "userid.dataset.OUT"
 */
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
```

In the second example, the output of GIMXTRX is written to the data set specified on the SMPXTOUT DD statement.

```
//jobname JOB ...
//step1 EXEC PGM=IEBGENER
/**
 * Create the input data set.
 */
//SYSPRINT DD SYSOUT=*
//SYSIN DD DUMMY
//SYSUT2 DD DSN=userid.dataset.INPUT,DISP=(NEW,CATLG),
//   DCB=(RECFM=FB,LRECL=80),
//   UNIT=unit,SPACE=(TRK,(1,0))
//SYSUT1 DD DATA,DLM=$$
SUFFNC=BMPTZN
SUFGBL=userid.GLOBAL.CSI
SUFTGT=ZOSTGT
$$
//step2 EXEC PGM=GIMXTRX,PARM='userid.dataset.INPUT'
/**
 * The output Bit-Map is written to the data set specified
 * on the SMPXTOUT DD statement.
 */
```
Processing

GIMXTRX obtains the input parameter data set name from the PARM value passed to it on the EXEC statement.

If the SMPXTOUT DD statement is specified, GIMXTRX uses the data set specified for the output. If the SMPXTOUT DD statement is not specified, GIMXTRX allocates a new data set for the output. GIMXTRX constructs the name of the output data set using the input parameter data set name as a base. The last qualifier of the input parameter data set name is changed to 'OUT' and this becomes the name of the output data set. For example, if the input parameter data set name is TEST.GIMXTRX.DATA.IN, the output data set name will be TEST.GIMXTRX.DATA.OUT.

The output data set is allocated and cataloged as follows:

- **DSNAME=dsn.OUT**
- **DSORG=PS**
- **RECFM=FB**
- **LRECL=12560**
- **BLKSIZE=0**
- **SPACE=(TRK,(10,1),RLSE)**
- **UNIT=unit** (supplied as **SUFUNIT=** on input) or default of **SYSALLDA**
- **VOLUME=volume name** (supplied as **SUFVOL=** on input).

If the SMPXTOUT DD statement is not specified, GIMXTRX attempts to create the output data set, which therefore must not exist at the time that GIMXTRX is initiated. If there is no volume supplied as input, GIMXTRX will attempt the allocation without a volume serial. If there is no unit supplied as input, GIMXTRX will attempt the allocation with UNIT=SYSALLDA.

GIMXTRX then allocates and opens all of the required data sets and extracts the input parameters from the input parameter data set.

Once the input parameters have been read and the output data set created, GIMXTRX continues processing based on the function requested (LSTTZN or BMPTZN).

**List target zone names (LSTTZN)**

This function of GIMXTRX produces a list of target zone names associated with a given GLOBAL zone SMPCSI data set name. The results are written to the output data set.

**Inputs:** The inputs to the List Target Zone Names function are:

- **SUFFNC**
  Name of the function to be executed (LSTTZN)

- **SUFGBL**
  GLOBAL zone SMPCSI data set name

- **SUFUNIT**
  UNIT to be used for output data set (optional)

- **SUFVOL**
  VOLUME to be used for output data set (optional)
A sample of the parameters contained in the input parameter data set for the List Target Zone Names function is:

```
1 2 3 4 5 6 7 8
---------0---------0---------0---------0---------0---------0---------0
SUFFNC=LSTTZN
SUFGBL=TEST.GLOBAL.VSAM.CSI
SUFVOL=VOL111
```

*Figure 96. Sample input parameter data set for function LSTTZN*

Using the SMP/E GIMAPI, GIMXTRX obtains the names of all of the target zones associated with the specified global zone. GIMXTRX then writes the list of target zone names to the output data set in 12560-byte records defined as follows:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>header</td>
<td>60-byte header character string starting with the literal &quot;TARGETZN&quot; in upper case. The remaining 52 bytes are set to blanks.</td>
</tr>
<tr>
<td>Target zone name list</td>
<td>7-character target zone names each separated by one blank space, up to 12500 bytes in length. For target zone names that are less than 7 characters long, the field is left-justified and padded to the right with blanks. When the list is less than 12500 bytes long, the remaining bytes contain blanks.</td>
</tr>
</tbody>
</table>

*Example of a target zone name record:* The following example indicates an SMPCSI that has target zones MVST1, JESTGT1, CICTGA, DB2TG2.

```
1 2 3 4 5 6 7 8 9
---------0---------0---------0---------0---------0---------0---------0---------0
TARGETZN MVST1 JESTGT1 CICTGA DB2TG2
```

*Figure 97. TARGETZN record*

**Create a BITMAP of an SMPCSI (BMPTZN)**

This function of GIMXTRX produces a BITMAP representation of the PTF service present in a user's SMPCSI. This information is intended for shipment to IBM where it can be analyzed to create a custom-built service package to be delivered to the user over the internet.

The intent of the Create a BITMAP of an SMPCSI function is to identify and record in a set of BITMAP records:

- All FMIDs and PTFs applied in the target zones identified in the input data set (SUFTGT parameter).
- All PTFs received in the GLOBAL zone identified in the input data set (SUFGBL parameter) that are associated with the FMIDs applied in the specified target zones.

A PTF is considered applicable to a target zone if its FMID is applied in that target zone. If a PTF is applied in all selected target zones to which it is applicable, it is recorded in a BITMAP record. If a PTF is not applied in all of the selected target zones to which it is applicable, then it is not recorded in a BITMAP record. For example, if an FMID is applied in 4 of the selected target zones, but a PTF associated with that FMID is applied in only 3 of those zones, the PTF is not recorded in the BITMAP record.
All PTFs that are received in the GLOBAL zone are checked against the FMIDs applied in the selected target zones to determine if the PTFs are associated with the applied FMIDs. All PTFs received in the GLOBAL zone that are applicable to the FMIDs applied in the selected target zones are recorded in a BITMAP record.

**Inputs:** The inputs to the Create a BITMAP of an SMPCSI function are:

- **SUFFNC**
  Name of the function to be executed (BMPTZN)

- **SUFGBL**
  GLOBAL zone SMPCSI data set name

- **SUFUNIT**
  UNIT to be used for output data set (optional)

- **SUFVOL**
  VOLUME to be used for output data set (optional)

- **SUFTGT**
  Target zone names selected by the user

A sample of the input parameters contained in the input parameter data set for the create a BITMAP of an SMPCSI function is:

```
12345678
--------0--------0--------0--------0--------0--------0--------0--------0
SUFFNC=BMPTZN
SUFGBL=TEST.GLOBAL.VSAM.CSI
SUFVOL=VOL111
SUFTGT=TGT1 TGT2 TARGET3 TGT4 TARGET5 TGT6 TGT7
```

**Figure 98. Sample input parameter data set for function BMPTZN**

The output records created by GIMXTRX for the BMPTZN function represent the functions applied in the target zones selected by the user (PRODLIST record) and the PTFs that are either applied or received (BITMAP records). GIMXTRX then writes the PRODLIST and BITMAP records to the output data set in 12560-byte records defined as follows:

**PRODLIST record:** The PRODLIST record contains the list of FMIDs applied in the target zones identified by the user in the input data set. Following is the definition of the PRODLIST record that is placed in the output data set by GIMXTRX.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>header</td>
<td>60-byte header starting with the literal &quot;PRODLIST&quot; in upper case. The remaining 52 bytes are set to blank.</td>
</tr>
</tbody>
</table>

**FMID list**

- list of 7-character FMIDs each separated by one blank space. This field is 12500 bytes long. When the FMID list is less than 12500 bytes long, the remaining bytes will be made blanks.

The following example of a PRODLIST record indicates an SMPCSI that has FMIDs HBB6602, HMP1B00, JMP1B01 present.
The PRODLIST record will be placed before any BITMAP bitstring records in the output data set.

**BITMAP records:** The BITMAP records represent the PTFs applied in the target zones or received in the GLOBAL zone that are associated with the FMIDs listed in the PRODLIST record.

The BITMAP bitstring logic is based on the following PTF naming convention:
- PTF names are architected by IBM to consist of a 2 character prefix followed by 5 decimal digits providing 100000 possible PTF numbers per prefix (00000-99999).
- A BITMAP bitstring record simply contains 100000 bits and a 2-character PTF prefix in the record header.
- For example, if the prefix is "UR", then the first bit in the bitstring represents PTF UR00000, the second bit represents PTF UR00001, the third represents PTF UR00002 and so on. The last bit in the bitstring represents UR99999.
- The bits representing the URnnnnn PTF numbers received or applied on the customer system are turned on while the remaining bits are set to zero.
- One bitstring is required for each PTF prefix in the SMPCSI.

Following is the definition of the BITMAP records that are placed in the output data set by GIMXTRX.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>header</td>
<td>60-byte header starting with the literal &quot;BITPTF&quot; in upper case, followed by</td>
</tr>
<tr>
<td></td>
<td>the 2-character PTF prefix for each unique PTF prefix present in the zones</td>
</tr>
<tr>
<td></td>
<td>being queried. This field is in upper case. The remaining 52 bytes in the</td>
</tr>
<tr>
<td></td>
<td>header are set to blank.</td>
</tr>
<tr>
<td>bitstring</td>
<td>12500 bytes with 100000 bits representing PTF numbers present in the SMPCSI</td>
</tr>
<tr>
<td></td>
<td>associated with the two-character PTF prefix in this record. A bit is set</td>
</tr>
<tr>
<td></td>
<td>on to indicate a PTF number exists. All remaining bit positions are set to</td>
</tr>
<tr>
<td></td>
<td>zero.</td>
</tr>
</tbody>
</table>

The following example of a BITMAP bitstring record indicates an SMPCSI that has PTFs UR00001 and UR00005 present.

```
1 2 3 4 5 6 7 8
-----+-----+-----+-----+-----+-----+-----+-----+
PRODLIST     HBB6602 HMP1B00 JMP1B01

Figure 99. PRODLIST record
```

```
1 2 3 4 5 6 7 8
-----+-----+-----+-----+-----+-----+-----+-----+
BITPTFUR     bitstring...........

Figure 100. BITMAP record
```

The bitstring section of the BITMAP record is expanded as follows (where each mark of the scale line represents one bit position of the BITMAP record):
Return codes

To help you diagnose errors, GIMXTRX issues messages and return codes. The messages are documented in *SMP/E Messages, Codes, and Diagnosis*. All nonzero return codes indicate unsuccessful completion of GIMXTRX. Here is a description of the return codes:

**Return code**

**Meaning**

0  The requested GIMXTRX function was processes successfully.
4  A warning condition was encountered in a program called by GIMXTRX.
8  An error condition was encountered in a program called by GIMXTRX.
12 One of the following
  • A required parameter is missing or incorrect.
  • An allocation error occurred.
  • An I/O error occurred.
  • An error occurred while opening or closing a data set.
  • GIMXTRX could not load either GIMAPI, GIMSGENU, or GIMKXENU.
  • SMP/E could not obtain enough storage for processing.

If an error occurs with SYSTSPRT, there will be no message issued. Only a return code will be returned through standard linkage.

GIMXTRX does not provide an ESTAE routine for recovery when an ABEND occurs. Normal system ABEND processing occurs.

GIMZIP packaging service routine

The GIMZIP service routine creates portable packages of software and associated materials. Typically the packages will contain SYSMODs, RELFILE data sets, HOLDDATA, and associated materials such as documentation, samples, and text files. These GIMZIP packages may be transported through a network, processed by the GIMUNZIP service routine, and then processed by the SMP/E RECEIVE command.

More specifically, a GIMZIP package consists of a single package definition file, a set of archive files, and text files. The package definition file describes the total package and identifies the archive files and text files contained in the package. An archive file consists of:

1. a portable image of any of the following:
   • a sequential data set
   • a partitioned data set
   • a VSAM data set
GIMZIP

- a file in the UNIX file system
- a directory in the UNIX file system

2. and the information necessary to reload the data from the portable image.

A single GIMZIP package typically consists of several archive files.

Notes:
1. GIMZIP is a separate load module residing in the MIGLIB library and runs independently from the rest of SMP/E processing.
2. GIMZIP requires either the Integrated Cryptographic Services Facility (ICSF) One-Way Hash Generate callable service or Java 2 Version 1 Release 4 to be available for its use in order to compute an SHA-1 hash value.

Calling GIMZIP

The following figure shows the job control statements for GIMZIP:

```plaintext
//JOBx JOB ...
//STEP1 EXEC PGM=GIMZIP,PARM='option'
//SMPDIR DD PATH='package_directory',PATHDISP=KEEP
//SMWDIR DD PATH='work_directory',PATHDISP=KEEP
//SMPCPATH DD PATH='smptclasses_directory',PATHDISP=KEEP
//SMJHOME DD PATH='javuruntime_directory',PATHDISP=KEEP
//SMOUT DD SYSOUT=*
//SYSIN DD *

* package control tags

Figure 102. JCL to call GIMZIP
```

EXEC

is the statement used to start GIMZIP. The EXEC statement must specify
PGM=GIMZIP. The following options may be specified on the EXEC statement
PARM operand:

**LANGUAGE=xxx**

where xxx can be one of the following:

ENU US English
JPN Japanese

The LANGUAGE option defines which language to use for GIMZIP
messages. LANGUAGE can also be specified as L. If LANGUAGE is not
specified, the default is LANGUAGE=ENU.

**SEGMENT=nnnnM**

where nnnn is a decimal number between 1 and 9999. The SEGMENT
option indicates the archive files produced by GIMZIP are to be divided
into archive segments no larger than nnnn megabytes. If SEGMENT is not
specified, the default is to perform no archive segmenting.

**SYSIN**

specifies a control data set that contains the package control tags to direct
GIMZIP processing (see "GIMZIP package control tags" on page 481). The
control data set may be a sequential data set or a single member of a
partitioned data set and its attributes must be LRECL=80 and RECFM=F or FB.
SMPDIR
specifies a directory in a UNIX file system that is to contain the package created by GIMZIP. All files and archives created by GIMZIP are stored in this directory. This directory is referred to as the package directory.

The package directory must be empty at the start of GIMZIP processing. If any files, including a previous package, already exist in the package directory, you must move them to another directory, delete them, or specify a different directory on the SMPDIR DD statement for the package directory.

SMPWKDIR
specifies a directory in a UNIX file system that is used by GIMZIP for temporary work files. This is an optional DD statement. If the SMPWKDIR DD statement is not provided, GIMZIP will use the package directory specified on the SMPDIR DD statement for temporary work files.

SMPCPATH
specifies a directory in the UNIX file system where the SMP/E Java classes reside. The SMPCPATH DD statement is optional and only required to calculate SHA-1 hash values if ICSF is not available.

SMPJHOME
specifies a directory in the UNIX file system where the Java runtime resides. The SMPJHOME DD statement is optional and only required to calculate SHA-1 hash values if ICSF is not available.

SMPOUT
specifies an output data set that will contain SMP/E messages issued during GIMZIP processing. The data set attributes are LRECL=81 or LRECL=121 and RECFM=FBA.

SYSUT2
specifies a work area for the IEBCOPY utility to use for the sequential unload data sets placed on DASD as the result of an IEBCOPY unload operation. This data set must be large enough to accommodate the largest partitioned data set being processed by the current execution of GIMZIP.

SYSUT3 and SYSUT4
specifies work areas for the IEBCOPY utility to use for spill data sets on DASD.

SYSPRINT
specifies a data set to contain output and messages from the utilities called by GIMZIP.

GIMZIP package control tags
GIMZIP service routine processing is controlled by package control tags. The package control tags define the GIMZIP package to be created and are specified in the SYSIN data set. The package control tags follow XML syntax rules and their format is as follows:
GIMZIP package control tags

**GIMZIP**:  
```xml
<GIMZIP
    description="package description"

  FILEDEF
    name="file name"
      archid="archive id"
    subdir="subdirectory name"
      description="file description"
    type="README"
      volume="file volume"

  </FILEDEF>

</GIMZIP>
```

**FILEDEF**:  
```xml
<FILEDEF
    name="file name"
      archid="archive id"
      subdir="subdirectory name"
      description="file description"
      type="README"
      volume="file volume"

    </FILEDEF>
```

**<GIMZIP> Tag syntax**
The `<GIMZIP>` and corresponding `</GIMZIP>` tags identify the beginning and end of a GIMZIP package build request. The following attribute may be found on the `<GIMZIP>` tag:

* description="package description"
  
  specifies a text description for the package. The description can be up to 500 characters long.

**<FILEDEF> Tag syntax**
The `<FILEDEF>` and corresponding `</FILEDEF>` tags identify the beginning and end of a file definition. Each file described by a `<FILEDEF>` tag is input to GIMZIP and is to be a component of the GIMZIP package. The following attributes may be found on the `<FILEDEF>` tag:

* name="file name"
  
  specifies the name of a data set or file that is to be a component of the GIMZIP package. The name attribute can specify either a sequential, partitioned, or VSAM data set, or the absolute pathname for a file or directory in the UNIX file system.

  The first character of the absolute pathname of a file or directory in the UNIX file system must be a slash (/). If the name does not begin with a slash, GIMZIP processing will assume that it is an MVS data set name and that it will conform to the MVS data set naming conventions. When a file or directory in the UNIX file system is specified, the name can be from 1 to 1023 bytes long with 255 characters between delimiters (/). The value can contain any character from X'40' through X'FE', except `<`, `>`, `&`, and the double quotation mark (``). All data beyond column 72 is ignored, including blanks. The pathname of a file or directory in the UNIX file system is case sensitive and will not be converted to uppercase alphabetic during GIMZIP processing. Data set names are not case sensitive and will be converted to uppercase alphabetic during GIMZIP processing.
Notes:

1. When a VSAM data set is being specified, the true cluster name must be used. Do not reference a VSAM data set by a path name. Although an alternate index may be defined to the cluster, the alternate index does not become part of the archive. If an alternate index is desired at the destination site after the archive is unzipped, then the alternate index must be defined and built at the destination site.

2. GIMZIP processing uses the UNIX System Services `pax` command for creating archive files. Although GIMZIP will attempt to process UNIX file and directory names of up to 1023 bytes in length, this processing is subject to the limitations of the `pax` command. See z/OS UNIX System Services Command Reference, SA22-7802 for information about the `pax` command.

**archid**="archive id"
specifies a unique archive id value to be associated with the archive file created for the data set, file, or directory identified by the associated name attribute. The value of this attribute will most likely be something like a ddname or the low level qualifier of an MVS library name or some generic specification for a file or directory in the UNIX file system.

The value can be from 1 to 243 characters in length. The value can contain any character from 'X'40' through X'FE', except '<', '>', '&', '/' and the double quotation mark ("). The archid value is case sensitive and will not be converted to uppercase alphabetic by GIMZIP processing.

The same archid value may not be used in more than one file definition group in the same package.

**description**="file description"
specifies a text description for the file. The description can be up to 500 characters long.

**subdir**="subdirectory name"
specifies the subdirectory in which to store the file specified by the corresponding name attribute. The subdirectory name can be from 1 to 500 bytes long with 255 characters between delimiters (/).

This attribute can only be specified if the file type attribute is either not specified or is README.

The subdirectory name is a relative pathname for the desired subdirectory, so the first character of the subdirectory name cannot be a slash (/). However, the subdirectory name may be specified with or without an ending slash.

"SMP", all capitalized, cannot be specified as the first three characters of the subdir attribute value. Other case variations are allowed.

**type**="file type"
specifies a file type indicator that describes the contents of the file, identifies how the file is to be processed by GIMZIP, and also how the file will be processed by the SMP/E RECEIVE command. Any of the following values may be specified:

- **README** indicates that the associated file is a sequential text data set or a file in the UNIX file system that is intended to be viewable text after it is placed in the package. These files are not placed in archives by GIMZIP and are not compressed. They are only copied into the package by GIMZIP. The file name specified on
the associated **name** attribute must identify a file in the UNIX file system or a sequential data set that does not have a record format of VS (variable spanned).

**SMPHOLD** indicates that the associated data set or file contains SMP/E HOLDDATA statements. These files are placed in archives by GIMZIP and compressed. The name specified on the associated **name** attribute must identify a sequential data set with a logical record length of eighty (80) bytes and a record format of FB (fixed block), or it must identify a file in the UNIX file system.

**SMPPTFIN** indicates that the associated data set or file contains SMP/E modification control statements (MCS). These files are placed in archives by GIMZIP and compressed. The name specified on the associated name attribute must identify a sequential data set with a logical record length of eighty (80) bytes and a record format of FB (fixed block), or it must identify a file in the UNIX file system.

**SMPRELF** indicates the data set specified on the **name** attribute is an SMP/E RELFILE data set associated with a SYSMOD contained in the SMPPTFIN file. Data sets with the SMPRELF file type must be either a partitioned data set or an IEBCOPY unloaded sequential image of a partitioned data set. These data sets are placed in archives by GIMZIP and compressed.

The **type** attribute must not be specified when a VSAM data set or a UNIX file system directory is specified on the **name** attribute. If the **type** attribute is not specified, then the data set specified on the **name** attribute can be either a sequential, partitioned, or VSAM data set, or a file or directory in the UNIX file system.

**volume**="**file volume**"

specifies the volume serial number of the volume containing the data set specified on the **name** attribute. The volume identifier must be from 1 to 6 alphanumeric characters and should be specified only if the **name** attribute identifies an uncataloged data set.

If the associated name attribute identifies a file or directory in the UNIX file system (the name begins with a slash), the volume attribute will be checked for syntax, but it will be ignored otherwise.

**Note:** The volume is used to allocate data sets. However, VSAM data sets must be cataloged. Information needed for the archival of VSAM data sets is gathered from the catalog and related areas, such as the VVDS - VSAM volume data set.

**Syntax notes**

1. GIMZIP ignores columns 73 through 80. If data is specified beyond column 72, GIMZIP ignores it, which may lead to the diagnosis of an error in a following tag.

2. Package control tags may contain comments. Comments start with <!-- (hex 4C5A6060) and end with --> (hex 60606E). The first --> encountered after the initial <!-- will end the comment. A comment may appear between a start-tag and its matching end-tag, but never within a tag.
Example of using GIMZIP

Suppose a GIMZIP package is to be created containing the following data sets, files, and directories:

- /SAMPLE/ORDER123/SMPMCS - an SMP/E MCS file
- SAMPLE.IBM.FMID001.F1 - An SMP/E PDS relative file
- SAMPLE.IBM.FMID001.F2 - An SMP/E PDSE relative file
- SAMPLE.ORDER123.DOCLIB - A PDS containing documents
- SAMPLE.ORDER123.RIMLIB - A PDS containing related installation materials
- SAMPLE.ORDER123.MVS.GLOBAL.CSI - A VSAM cluster
- /SAMPLE/ORDER123/RootHFS/ - the root directory

The following job stream can be used to create such a GIMZIP package:
GIMZIP

<FILEDEF name="/SAMPLE/ORDER123/readme.html"
archid="readme.html"
description="This is the README file for the package."
subdir="README"
type="README">
</FILEDEF>

<FILEDEF name="/SAMPLE/ORDER123/SMPMCS"
description="This is the SMPMCS file for ORDER123."
archid="SMPMCS"
type="SMPPTFIN">
</FILEDEF>

<FILEDEF name="SAMPLE.IBM.FMID001.F1"
archid="FMID001.F1"
description="This is SMP/E RELFILE 1 for FMID001."
type="SMPRELF">
</FILEDEF>

<FILEDEF name="SAMPLE.IBM.FMID001.F2"
archid="FMID001.F2"
description="This is SMP/E RELFILE 2 for FMID001."
type="SMPRELF">
</FILEDEF>

<FILEDEF name="SAMPLE.ORDER123.DOCLIB"
subdir="DOCLIB"
description="This is the Document Library for the package.">
</FILEDEF>

<FILEDEF name="SAMPLE.ORDER123.RIMLIB"
subdir="RIMLIB"
description="This is the Related Installation Materials Library for the package."
</FILEDEF>

<FILEDEF name="SAMPLE.ORDER123.MVS.GLOBAL.CSI"
archid="GLOBAL"
description="This is a sample VSAM cluster.">
</FILEDEF>

<FILEDEF name="/SAMPLE/ORDER123/RootHFS/
description="This is the entire root directory.">
</FILEDEF>

</GIMZIP>

 Figure 103. GIMZIP example
Note: Blank lines and spaces have been added to the package control tags for clarity, but are not required.

**GIMZIP processing**
The GIMZIP service routine processes the following data structures as input to create GIMZIP packages:
- sequential data sets
- partitioned (PDS and PDSE) data sets
- VSAM data sets
- UNIX files
- UNIX directory files

The packages produced are stored in the package directory in a UNIX file system. The package directory is specified by the SMPDIR DD statement.

GIMZIP collects various attributes for the input data sets and files specified in the package control tags. These attributes, along with portable images of the input data sets, are temporarily stored in a UNIX file system. These images are then archived and compressed to create archive files. Each GIMZIP package will contain one or more archive files.

For each package there are two additional files:
- The package attribute file, which describes the contents of the package itself. The package attribute file identifies the archives for a package and contains a hash value for each archive within the package. The hash value for each archive is used for data integrity purposes and can be checked when the original data set is recreated from the archive using the GIMUNZIP service routine.
- An extensible stylesheet language (XSL) file, which describes how to format the information found in the package attribute file. The XSL file is used for rendering (displaying on a browser) the package attribute file.

**Package attribute file**
Each package contains a package attribute file. The name of this file is `GIMPAF.XML` and it is stored in the package directory identified by the SMPDIR DD statement. The file contains package definition control tags that describe the contents of the package and how the package was created. The package definition control tags follow XML syntax rules.

The format of the package definition control tags is as follows:
Package definition control tags

**PKGDEF:**

```
<PKGDEF
date="yyyy.ddd"
description="package description"
files="filenum"
gmt="hh:mm:ss"
>ARCHDEF
>level="vv.rr.mm.pp"
originalsize="original package size"
size="package size"
</PKGDEF>
```

**ARCHDEF:**

```
<ARCHDEF
name="archive name"
archid="archive id"
description="archive description"
>hash="archive hash value"
originalsize="original file size"
size="archive size"
</ARCHDEF>
```

**ARCHSEG:**

```
<ARCHSEG
name="segment name"
hash="segment hash value"
size="segment size"
</ARCHSEG>
```

**PKGHASH:**

```
<?PKGHASH
hash="package hash value"?>
```

**<PKGDEF> Tag syntax**

The `<PKGDEF>` and corresponding `<PKGHASH>` tags identify the beginning and end of a GIMZIP package definition. The following attributes may be found on the `<PKGDEF>` tag:

- `date="yyyy.ddd"`
  - Indicates the year and Julian day on which the package was created by the GIMZIP service routine.
description="package description"
  specifies a text description for the package. The value GIMZIP places here, is
  the same as the description value, if specified, on the input <GIMZIP> tag.

files="filenum"
  indicates the total number of files that compose the package. This number
  includes all archives, and any and all control files, such as the Package
  Attribute File.

gmt="hh:mm:ss"
  indicates the time at which the package was created by the GIMZIP service
  routine. The time value is expressed as Greenwich Mean Time (GMT).

level="vv.rr.mm.pp"
  indicates the service level of the GIMZIP service routine used to create this
  package. The individual values are as follows:
    vv    version
    rr    release
    mm    modification level
    pp    PTF level

originalsize="original package size"
  indicates the sum of all the values of originalsize attributes on ARCHDEF tags
  in this package definition. It is used to determine the amount of auxiliary
  storage required to retrieve all the archives in this package.

size="package size"
  indicates the total size of the package, in bytes. This size is the sum of the sizes
  of all files that compose the package.

<ARCHDEF> Tag syntax
The <ARCHDEF> and corresponding </ARCHDEF> tags identify the beginning
and end of a specific archive file definition. The following attributes may be found
on the <ARCHDEF> tag:

name="archive name"
  specifies the relative path name for an archive file in the GIMZIP package. The
  path name value is relative to the package directory specified on the SMPDIR
  DD statement. See "Archive files" on page 492 for information about the
  proper naming convention for archive files.

archid="archive id"
  specifies a unique archive id associated with the archive specified in the
  archive name attribute. There can be only one archid attribute in an archive
definition group. The value is taken from the archid value found on the file
definition tag in the GIMZIP package control tags (if one was specified). This is
  an optional attribute.

description="archive description"
  specifies a text description for the archive. The value GIMZIP produces is
  copied from the description value, if specified, on the corresponding input
  <FILEDEF> tag.

hash="archive hash"
  indicates an SHA-1 hash value for the archive file.

originalsize="original file size"
  indicates the total size, in bytes, of the original data set, directory, or file stored
  in the archive file.

size="archive size"
  indicates the total size of the archive file, in bytes.
type="file type"
specifies an archive type indicator that describes the contents of the archive and how the file will be processed by the SMP/E RECEIVE command. The value is copied from the file type value, if specified, on the corresponding <FILEDEF> statement, and may be any of the following values:

- README indicates the archive contains text data.
- SMPHOLD indicates the archive contains SMP/E HOLDDATA statements.
- SMPPTFIN indicates the archive contains SMP/E modification control statements (MCS).
- SMPRELF indicates the archive contains an SMP/E RELFILE data set associated with a SYSMOD contained in the SMPPTFIN file.

<ARCHSEG> Tag syntax
The <ARCHSEG> tag and matching </ARCHSEG> end tag identify a segment of an archive file. The following attributes may be found on the <ARCHSEG> tag:

name="segment name"
specifies the path name of a segment file for an archive file in the GIMZIP package. The path name value is a relative value and it is relative to the package directory specified on the SMPDIR DD statement for GIMZIP. This attribute and corresponding attribute value is always produced when archives are to be segmented.

size="segment size"
indicates the total size of the segment file, in bytes. While GIMZIP always produces a byte count when archives are to be segmented, this attribute and attribute value is optional and is not used for GIMUNZIP or RECEIVE processing.

hash="segment hash"
indicates an SHA-1 hash value for the segment file. This attribute and corresponding attribute value is always produced when archives are to be segmented.

<?PKGHASH> Tag syntax
hash="package hash"
indicates an SHA-1 hash value for the information in the package attribute file between the starting <PKGDEF> tag and the ending </PKGDEF> tag.

Syntax notes
1. Data in columns 73 through 80 is ignored. If data is specified beyond column 72, it is ignored and an error in a following tag may be indicated.
2. Package definition control tags may contain comments. Comments start with <!-- (hex 4C5A6060) and end with -- (hex 60606E). The first --> encountered after the initial <!-- will end the comment. A comment may appear between a start-tag and its matching end-tag, but never within a tag.

Package attribute file example
The package attribute file for the package produced by GIMZIP in Figure 103 on page 486 would look like the following:
Figure 104. Package attribute file (GIMPAF.XML) example (Part 1 of 2)
**Note:** Spaces have been added to the example for clarity, but are not necessarily produced by GIMZIP, nor are they required. Some of the values, such as size, original size, and hash, are for illustration purposes only, and may not be accurate.

### Displaying the package attribute file

The package attribute file (GIMPAF.XML) can be rendered on an internet browser that supports XML and XSL (for example, Microsoft Internet Explorer 5). The package directory containing the package attribute file must be accessible to the browser through your z/OS web server. The accessibility of the package directory is determined by your specific web server configuration.

For example, suppose /u/userid/public/ is defined by your z/OS web server to be your publicly accessible directory. Further suppose, to view the contents of this directory, you must specify an URL of the form `http://host name/~userid`. Then, in order to view the package attribute file, your package directory must be contained within the /u/userid/public/ directory, or a symbolic link that points to the package directory must be in the /u/userid/public/ directory in a UNIX file system. Furthermore, your z/OS web server must be configured to recognize .XML and .XSL as text MIME types and, if appropriate, convert EBCDIC text to ASCII text.

The package attribute file can then be viewed on an internet browser by entering an URL similar to the following:

```
http://host name/~userid/GIMPAF.XML
```

### Archive files

The actual software and associated materials for a package are stored in archive files. GIMZIP creates an archive file for each input data set or file specified by a `<FILEDEF>` tag. An archive file consists of a portable image of the original data set or file and attributes of the original data set or file needed to reload the data from the portable image.

The attributes of the original data set or file are recorded in a file attribute file in the UNIX file system called GIMFAF.XML. For data sets, the portable image of the original data set is stored temporarily as a file in the UNIX file system (this step is not required for UNIX files and directories). The name of this temporary file is the
type attribute value for the archive (SMPPTFIN, SMPHOLD, SMPRELF) or the name MVSFILE if the archive has no type attribute specified. An archive file is then created by using the UNIX System Services `pax` command to combine the GIMFAF.XML file and the original data into an archive. Although a UNIX file is not first stored as a temporary file with a name matching its type attribute value, its name within the archive will be its type attribute value, or the name MVSFILE, if the archive has no type attribute specified. The GIMUNZIP service routine and the RECEIVE command both expect an archive file to contain the GIMFAF.XML component file, and the following, depending on the original content:

For data sets:
  a file named SMPPTFIN, SMPHOLD, SMPRELF, or MVSFILE

For UNIX files:
  a file named SMPPTFIN, SMPHOLD, or MVSFILE

For UNIX directories:
  the directory and its contents are stored in the archive using the original names.

Data sets and files with a type of README are an exception to the previously described archive processing. README data sets and files have no file attribute file and are not archived using `pax`, but rather are stored unchanged in a file in the package directory. Likewise, README data sets and files are not subject to archive segmentation.

The absolute names for archive and README files in the package directory use the following format:

/package_directory/subdir/Snnnn.original__name.pax.Z

The absolute names for archive segment files in the package directory use the following format:

/package_directory/subdir/date_timeofday.nofm

/package_directory
  indicates the package directory specified on the SMPDIR DD statement.

/subdir
  indicates the subdirectory into which the archive file is stored. GIMZIP creates this subdirectory based on the filetype or the subdir attributes specified on the `<FILEDEF>` tag for the data set. If neither attribute is specified, then no subdirectory is used. If the file type is README, the subdirectory is created only if the `subdir` attribute has been specified.

/Snnnn
  specifies the sequence indicator for the archive file. The sequence indicator is necessary so the correct order can be determined for the SMPHOLD and SMPPTFIN files when eventually processed by the SMP/E RECEIVE command. GIMZIP processes the input data sets or files in the sequence in which they are specified in the package control tags and the indicator is assigned accordingly. All archive files, except those specified with a file type of SMPRELF, are assigned a sequence indicator. If the file type is README and an archid tag was specified, then no sequence number will be assigned.

/original_name
  specifies the original name of the data set, file, or directory, as indicated on the `<FILEDEF>` tag. This could also be the archid for UNIX directories and README data sets or files, or the date and time for UNIX directories, if the archid is not specified.
GIMZIP

pax.Z

is the file extension for all archive files, except those with a file type of

README. The pax.Z extension indicates a file that has been processed by the
UNIX System Services pax command with the compress option.

date_timeofday

the date and time of day value, unique for each archive, but the same for each
segment file of a particular archive.

n  segment number

m  the total number of segments for the archive.

File attribute file

The File Attribute File (FAF) is included in the archive file along with the data set,
file or directory when it is archived. The FAF contains control tags that:

- Describe the attributes of the original data set, file or directory
- Provide information needed by GIMUNZIP to process the archive

The GIMUNZIP service routine uses the information in an FAF to reload the data
from an archive file. The FAF for a file or directory in the UNIX file system is
created in the temporary working directory, just as it is for data sets. The FAF
control tags follow XML syntax rules, and their format is as follows:
File attribute file control tags

### FILELIST:

```xml
<FILELIST>
  FILEATTR for MVS data set
  FILEATTR for UNIX file system
  FILEATTR for VSAM data set
</FILELIST>
```

#### FILEATTR for MVS data set:

```xml
<FILEATTR
  name="MVS data set name"
  description="file description"
  type="SMPHOLD"
  level="vv.rr.mm.pp"
  dsntype="LIBRARY"
  recfm="recfm"
  lrecl="length"
  blksize="size"
  blksizen="size"
  primary="prispace"
  secondary="secspace"
  dirblocks="blocks"
  allocunits="BLOCK"
  length="block length"
  record length="record length"
  avgrec="U"
</FILEATTR>
```

#### FILEATTR for UNIX file system:

```xml
<FILEATTR
  name="file or directory name"
  description="file"
  type="SMPHOLD"
  level="vv.rr.mm.pp"
  dsorg="UFS"
  dsntype="FILE"
</FILEATTR>
```

#### FILEATTR for VSAM data set:

```xml
<FILEATTR
  name="VSAM cluster name"
  description="file description"
  level="vv.rr.mm.pp"
  dsntype="ESDS"
  lrecl="LRECL of REPROed data from cluster"
  primary="prispace"
  secondary="secspace"
  allocunits="BLOCK"
  length="block length"
  record length="record length"
  avgrec="U"
</FILEATTR>
```

```
<DEFINECLUSTER component="DATA">
  <OPERAND>
    IDCAMS DEFINE operand</OPERAND>
</DEFINECLUSTER>
```
<FILELIST> Tag syntax: The <FILELIST> and corresponding </FILELIST> tags identify the beginning and end of a group of file definitions.

<FILEATTR> Tag syntax: The <FILEATTR> and corresponding </FILEATTR> tags identify the beginning and end of a file attribute definition for an archive file. The following attributes may be found on the <FILEATTR> tag:

**name**="file name | pathname"
- Specifies one of the following:
  - The name of the original MVS data set or VSAM cluster in this archive
  - The pathname for the original UNIX file or directory in this archive

**Notes:**
1. The first character of the absolute pathname of a file or directory in the UNIX file system must be a slash (/).
2. If the value of the name attribute does not begin with a slash, it is assumed to be an MVS data set name (sequential, partitioned, and VSAM data sets) and must conform to the MVS data set naming conventions.
3. When a file or directory in the UNIX file system is specified, the name can be from 1 to 1023 bytes long with 255 characters between delimiters (/). The value can be any character from X'40' through X'FE' except '<', '>', '&', and double quotation mark (''). All data beyond column 72 is ignored, including blanks. The pathname of a file or directory in the UNIX file system is case sensitive and will not be converted to uppercase alphabetic during GIMUNZIP processing. (Data set names are not case sensitive. They will be converted to uppercase alphabetic during GIMUNZIP processing.)

**description**="file description"
- Specifies a text description for the file. The value GIMZIP produces is copied from the description value, if specified, on the corresponding input <FILEDEF> tag. The description can be up to 500 characters long.

**type**="file type"
- Specifies a file type indicator that describes the contents of the file, identifies how the file is to be processed by GIMZIP, and also how the file will be processed by the SMP/E RECEIVE command. Any of the following values may be specified:

**SMPHOLD**
- Indicates that the data set or file contains SMP/E HOLDDATA statements.

**SMPPPTFIN**
- Indicates that the data set or file contains SMP/E modification control statements (MCS).

**SMPPREF**
- Indicates that the data set contains an SMP/E RELFILE data set associated with a SYSMOD contained in the SMPPPTFIN file.

If the type attribute is not specified, then the data set specified on the name attribute can be either sequential or partitioned.

**level**="vv.rr.mm.pp"
- Indicates the service level of the GIMZIP service routine used to create this file attribute file. The individual values are as follows:
  - **vv** version
  - **rr** release
dsntype="type"
    specifies the type of partitioned data set, VSAM cluster, or UNIX file or
directory for the original data.
    DIR         indicates a directory in the UNIX file system.
    FILE        indicates a file in the UNIX file system.
    ESDS        indicates a VSAM cluster that is an entry sequence data set.
    KSDS        indicates a VSAM cluster that is a keyed sequence data set.
    LDS         indicates a VSAM cluster that is a linear data set.
    RRDS        indicates a VSAM cluster that is a relative record data set.
    LIBRARY     indicates a PDSE partitioned data set.
    PDS         indicates a PDS partitioned data set.

dsorg="organization"
    specifies the organization of the original data set, file, or directory.
    PO          indicates a partitioned data set
    PS          indicates a sequential data set
    UFS         indicates a file or directory in the UNIX file system
    VSAM        indicates a VSAM cluster

recfm="format"
    specifies:

    • When dsorg="PS" or dsorg="PO", the record format of the sequential or
      partitioned MVS data set in an archive
    • When dsorg="VSAM", the record format of the data set used to contain the
      REPROed VSAM data

    The value can be from 1 to 3 alphabetic characters.

lrecl="length"
    specifies the logical record length of the original data set. The value can be
    from 1 to 5 decimal digits.

lreclu="length"
    specifies:

    • When dsorg="PO", the logical record length of the IEBCOPY unloaded
      format of the original data set
    • When dsorg="VSAM", the logical record length of the data set used to
      contain the REPROed VSAM data from the cluster

    In either case, the value can be from 1 to 5 decimal digits.

blksize="size"
    specifies the block size of the original data set. The value can be from 1 to 5
    decimal digits in the range of 00000 through 32760.

blksizeu="size"
    specifies:

    • When dsorg="PO", the block size of the IEBCOPY unloaded format of the
      original data set
    • When dsorg="VSAM", the block size of the data set used to contain the
      REPROed VSAM data from the cluster

    In either case, the value can be from 1 to 5 decimal digits in the range of 00000
    through 32760.
GIMZIP
primary="prispace"
specifies:
v When dsorg="PS" or dsorg="PO", the primary space needed to reload the
sequential or partitioned MVS data set in an archive
v When dsorg="VSAM", the primary space needed to create a temporary data
set to contain the REPROed VSAM data
The value must be non-zero and can be from 1 to 8 decimal digits.
secondary="secspace"
specifies:
v When dsorg="PS" or dsorg="PO", the secondary space needed to reload the
sequential or partitioned MVS data set in an archive
v When dsorg="VSAM", the secondary space needed to create a temporary
data set to contain the REPROed VSAM data
The value must be non-zero and can be from 1 to 8 decimal digits.
dirblocks="blocks"
specifies the number of directory blocks in the original data set. The value
must be non-zero and can be from 1 to 5 decimal digits.
allocunits="units"
specifies:
v When dsorg="PS" or dsorg="PO", the units to be used when allocating the
data set used to reload the sequential or partitioned MVS data set in an
archive
v When dsorg="VSAM", the units to be used to create a temporary data set to
contain the REPROed VSAM data
The value can be one of the following:
BLOCK
indicates space is to be allocated in blocks. The size of the blocks is
specified on the length attribute.
CYLINDER
indicates space is to be allocated in cylinders.
RECORD
indicates space is to be allocated in records. The average length of
these records is specified on the length attribute.
TRACK
indicates space is to be allocated in tracks.
length="average record length | block length"
specifies either the block length, or the average record length, used to allocate
the primary and secondary space for reloading the data from the archive. If
allocunits="BLOCK" is specified, then this value represents the block length. If
allocunits="RECORD" is specified, then this value represents the average
record length. The value of the length attribute must be from 1 to 5 decimal
digits and must be non-zero.
avgrec="multiplier"
indicates the multiplier for the primary and secondary space allocation values
only when allocunits="RECORD".
U
indicates that the primary and secondary space quantities represent the
number of records in units (multiplier of 1).

498

SMP/E V3R6.0 for z/OS V1R13.0 Reference


**GIMZIP**

**K** indicates that the primary and secondary space quantities represent the number of records in thousands (multiplier of 1024 or 1K).

**M** indicates that the primary and secondary space quantities represent the number of records in millions (multiplier of 1048576 or 1M).

- When `dsorg=PS` or `dsorg=PO`, this value represents the multiplier to be used for allocating the sequential or partitioned data set in an archive.
- When `dsorg=VSAM`, this value represents the multiplier to be used for allocating the temporary data set to contain the REPROed VSAM data.

**<DEFINECLUSTER> Tag syntax:** The `<DEFINECLUSTER>` tag and corresponding `</DEFINECLUSTER>` tag identify the beginning and end of information that is needed to define a new VSAM cluster during GIMUNZIP processing. The `<DEFINECLUSTER>` tag may be specified for the data component of a VSAM cluster. The `<DEFINECLUSTER>` tag may also be specified for the index component of a VSAM cluster if the cluster is a KSDS. The following attribute is found on the `<DEFINECLUSTER>` tag:

`component=`

specifies the name of the component type of a VSAM cluster, and may be one of the following values:

"DATA"

indicates that the following `<OPERAND>` tags are providing information to define the data component of a VSAM cluster.

"INDEX"

indicates that the following `<OPERAND>` tags are providing information to define the index component of a VSAM KSDS cluster.

Following the `<DEFINECLUSTER>` tag is the `<OPERAND>` tag.

**<OPERAND> Tag syntax:** The `<OPERAND>` tag and corresponding `</OPERAND>` tag identify the beginning and end of a valid IDCAMS DEFINE command operand for GIMUNZIP processing when a new VSAM cluster must be created in order to extract a VSAM archive. The `<OPERAND>` and `</OPERAND>` tags can only be specified within a `<DEFINECLUSTER>` tag and corresponding `</DEFINECLUSTER>` tag.

A maximum of 50 characters is allowed between the `<OPERAND>` tag and its corresponding `</OPERAND>` end tag. Multiple `<OPERAND>` tags may be specified to provide IDCAMS DEFINE command information.

**Note:** A portable package of software can be built either using GIMZIP, or manually (without using GIMZIP). If a packager is building a GIMZIP-like package manually and always intends that the users of the package will redefine a VSAM cluster for usage by GIMUNZIP, the `<DEFINECLUSTER>` tag and its imbedded `<OPERAND>` tags may be omitted from the FAF.

**Syntax notes:**

1. Data in columns 73 through 80 is ignored. If data is specified beyond column 72, it is ignored and an error in a following tag may be indicated.

2. File definition control tags may contain comments. Comments start with `<!-` (hex 4C5A6060) and end with `-->` (hex 60606E). The first `-->` encountered after the initial `<!-` will end the comment. A comment may appear between a start-tag and its matching end-tag, but never within a tag.
3. The space allocation for an archived data set is stored in the File Attribute File in terms of AVGREC regardless of how the data set was originally allocated. GIMZIP converts the current allocation information for a data set (as obtained from its DSCBs) into record type allocation information using AVGREC (usually set to U, although K could be used for extremely large data sets) and an average record length equal to the data set block size. The data set allocated using this information when an archive is retrieved by GIMUNZIP will approximately equal in size (number of bytes) the original data set, but will not use the same allocation units as the original data set that was archived by GIMZIP.

**File attribute file examples:** The following is an example of a file attribute file for a sequential data set containing modification control statements.

```xml
<?xml version="1.0" ?>
<FILELIST>
  <FILEATTR name="SAMPLE.FMID001.SMPMCS"
    description="This is the SMPMCS file for FMID001."
    level="02.07.00.35"
    type="SMPPTFIN"
    dsorg="PS"
    recfm="FB"
    lrecl="80"
    blksize="6160"
    allocunits="RECORD"
    length="6160"
    avgregc=U"
    primary="24000">
  </FILEATTR>
</FILELIST>
```

*Figure 105. File attribute file (GIMFAF.XML) example for a sequential data set*

The file attribute file included with the archive for file `/CPACTST/DO000029/SMPMCS` from the example in Figure 104 on page 491 would look something like this:

```xml
<FILELIST>
  <FILEATTR name="/CPACTST/DO000029/SMPMCS"
    description="This is the SMPMCS file for DO000029"
    level="03.03.00.00"
    type="SMPPTFIN"
    dsntype="FILE"
    dsorg="UFS">
  </FILEATTR>
</FILELIST>
```

*Figure 106. File attribute file example for a UNIX file*

The file attribute file included with the archive for VSAM cluster `/CPACTST.DO000029.MVS.GLOBAL.CSI` from the example in Figure 104 on page 491 would look something like this:
Return codes

GIMZIP may end with the following return codes:

<table>
<thead>
<tr>
<th>Return code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>The input data was processed successfully.</td>
</tr>
<tr>
<td>04</td>
<td>A call to a system service may not have completed successfully.</td>
</tr>
<tr>
<td>12</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>Required file attributes could not be obtained.</td>
</tr>
<tr>
<td></td>
<td>Input data sets were not sequential, partitioned, or VSAM data sets, nor files and directories in the UNIX file system.</td>
</tr>
<tr>
<td></td>
<td>Call to a required system service failed.</td>
</tr>
<tr>
<td></td>
<td>Syntax errors in the SYSIN data set.</td>
</tr>
<tr>
<td></td>
<td>Data sets could not be opened.</td>
</tr>
<tr>
<td></td>
<td>SMPDIR was not allocated to a UNIX directory.</td>
</tr>
<tr>
<td></td>
<td>SMPDIR directory was not empty.</td>
</tr>
<tr>
<td></td>
<td>Data sets, directories, or files were missing.</td>
</tr>
<tr>
<td></td>
<td>Required modules could not be loaded.</td>
</tr>
<tr>
<td>16</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>An I/O error occurred.</td>
</tr>
<tr>
<td></td>
<td>A syntax error was found on the EXEC statement parameters.</td>
</tr>
<tr>
<td>20</td>
<td>SMPOUT data set is missing.</td>
</tr>
</tbody>
</table>

Note: Blank lines and spaces have been added to the file attribute file for clarity, but are not necessarily produced by the GIMZIP service routine, nor are they required.
> 20   Internal error. Report the error to the IBM Support Center.
Chapter 12. GIMIAP: Copy utility invocation program

The SMP/E GENERATE command creates a job stream that builds a set of target libraries from a set of distribution libraries. For data elements or hierarchical file system elements, the GENERATE command builds a job (DEIINST for data elements or HFSINST for hierarchical file system elements) to invoke the appropriate copy utility for those elements. Each job step in the job installs multiple elements from multiple distribution libraries into a single target library. This is done by invoking the SMP/E program GIMIAP, which calls the appropriate copy utility to do the actual installation.

Although GIMIAP is to be used only by SMP/E, you may need to understand the control statements passed to the program. For example, you may need to diagnose errors detected by GIMIAP (such as syntax errors or missing information). To help you with this task, this chapter describes:

- The control statements used to invoke GIMIAP
- The return codes issued by GIMIAP
- The JCL statements used in the job that invokes GIMIAP

Control statements used to invoke GIMIAP

The GIMIAP program runs as a background job and is driven by control statements that identify the following:

- The copy utility to be invoked
- The distribution and target libraries to be used for each invocation of GIMIAP
- The elements to be installed
- The parameters to be passed to the copy utility

These control statements are created by SMP/E during GENERATE processing and are for use during GIMIAP processing. Each invocation of GIMIAP can install many elements through multiple invocations of the copy utility. Each invocation of the copy utility by GIMIAP installs a single element. These are the control statements used for input to a GIMIAP step:

- INVOKE
- COPY
- SELECT

The INVOKE control statement

The INVOKE control statement identifies the copy utility that GIMIAP should invoke.

The INVOKE control statement is always produced. There is one INVOKE control statement for each job step (that is, one INVOKE control statement for each invocation of GIMIAP). The INVOKE control statement must be the first control statement and must be a single-card image (one 80-byte record).
The following operands are required on the INVOKE statement:

**COPY**

Specifies the name of the copy utility to be used when copying data elements that do not need reformatting and are being installed into a partitioned data set. The value of COPY is either the value of the NAME operand in the UTILITY entry in effect at GENERATE time or the default of IEBCOPY.

**HFSCOPY**

Specifies the name of the copy utility to be used when copying hierarchical file system elements. The value of HFSCOPY is either the value of the NAME operand in the UTILITY entry in effect at GENERATE time or the default of BPXCOPY.

**LIST**

Applies only when copying data elements. It specifies whether member names should be listed during copy processing. This value is derived from the copy utility entry in effect when the GENERATE command was issued.

- **YES**: Indicates that member names should be listed during copy processing. This is the default.
- **NO**: Indicates that the list of member names should be suppressed during copy processing.

**PRINT**

Specifies the ddname that is to be used for print output generated by the specified copy utility. The value of PRINT is either the value of the PRINT operand in the UTILITY entry in effect at GENERATE time or the default of SYSPRINT.

**Note**: If SYSTSPRT is specified as the PRINT value for the copy utility, it is ignored and the default of SYSPRINT is used instead.

**RC**

Specifies the highest acceptable return code from the specified copy utility. The value of RC is either the value of the RC operand in the UTILITY entry in effect at GENERATE time or the default of 0.

**REPLACE**

Applies only when copying data elements. It specifies whether existing members are to be replaced when copying an element. If REPLACE is specified on the GENERATE command, then it is specified on the INVOKE control statement for data elements.

**COPY control statement**

A COPY control statement identifies the input and output libraries to be used for the SELECT control statements that follow it. Each COPY statement identifies a distribution library that has elements to be copied to a single target library. Multiple COPY statements can identify the same target library.
A COPY control statement must follow an INVOKE or SELECT control statement. Also, each COPY control statement must be followed by at least one SELECT control statement.

**COPY control statement**

COPY
FROMLIB(ddname)
TOLIB(ddname)

The following operands are required on the COPY statement:

**FROMLIB**

is the ddname used by the copy utility as its input file for the source of the element (that is, the distribution library). The name can be 1 to 8 characters and must be composed of uppercase alphabetic, numeric, or national ($, #, @) characters.

**TOLIB**

is the ddname used by the copy utility as its output file identifying where the element is to be installed (that is, the target library). The name can be 1 to 8 characters and must be composed of uppercase alphabetic, numeric, or national ($, #, @) characters.

**The SELECT control statement**

The SELECT control statement identifies the element to be installed and the operands to be passed to the copy utility or SMP/E to enable its installation. The SELECT control statement must follow a COPY control statement and may span multiple 80-byte records.

**SELECT control statement**

SELECT
\(type(name)\)

- ALIAS(alias)
- EPARM(option)
- SHSCRIPT({/directory/file\},PRE,POST)

**type**

is the type of the element to be processed. Any valid data element type or hierarchical file system element type, including those with national language identifiers, may be specified. An element type name is formed by stripping the ++ from the beginning of the name of the MCS for that element. For example, the element type for an element defined by a ++CLIST MCS is CLIST, while the element type for a ++HFSRMS MCS is HFSRMS. See "Data element MCS" on page 10 for a list of data element MCS and "Hierarchical file system element MCS" on page 26 for a list of hierarchical file system element MCS.

**name**

is the name of element to be processed.
ALIAS

specifies the list of alias names for the data element. This operand applies only to data elements.

EPARM

is the parameter list to be passed for this invocation of the HFSCOPY utility. This operand applies only to hierarchical file system elements. The parameter list consists of an LL value and the actual execution parameters.

- LL represents a halfword hexadecimal length of the character string (the HFSCOPY utility execution parameters) immediately following it as part of the EPARM value. The length of the character string described does not include the opening parenthesis preceding the LL value nor the closing parenthesis following the last option specified.

No blanks are allowed between the opening parenthesis and the LL value. The opening parenthesis and the LL value must be in the same record. Because the LL value is nondisplayable (ready-to-use) hexadecimal, it may appear as blanks or odd characters. This is valid data and should not be removed or modified.

- The option values are the execution parameters to be used by the HFSCOPY utility for this invocation of the utility. SMP/E always supplies execution parameters to the HFSCOPY utility, and the parameters are separated by commas with no intervening blanks.

If the HFSCOPY UTILITY entry that is in effect supplies execution parameters, these values precede the SMP/E-generated information. For example, suppose the UTILITY entry has the following values:

NAME

MYHFSCPY

PRINT

MYPRINT

PARM A-PARM-FOR-MYHFSCPY

The character string for the execution parameters would be generated as:

LLA-PARM-FOR-MYHFSCPY,ELEMENT(hfse1m1),TYPE(TEXT),LINK('linknm01'))

If a PARM value of user_info is specified in an element's MCS, the character string for the execution parameters would be generated as:

LLA-PARM-FOR-MYHFSCPY,user_info,ELEMENT(hfse1m1),TYPE(TEXT),LINK('linknm01'))

These are the parameters that are generated by SMP/E, using information in the hierarchical file system element entry:

ELEMENT(element_name)

the name of the element to be installed in a UNIX file system. The element_name is an unquoted character string 1 to 8 bytes long. It is composed of uppercase alphabetic, numeric, or national ($, #, @) characters.

LINK('linkname1','linkname2','linkname3'...)

the alternate names by which this element can be known in the target library within a UNIX file system. A linkname can be up to 1023 characters long and can contain special characters other than just uppercase alphabetic, numeric, or national ($, #, and @) characters.

SMP/E always puts apostrophes around each linkname and separates multiple values with commas with no intervening blanks.

SYMLINK('linkname1','linkname2','linkname3'...)

a list of symbolic links, which are file names that can be used as alternate names for referring to this element in a UNIX file system. Each
linkname listed here is associated with a pathname listed in the SYMPATH operand. A symbolic link can be up to 1023 characters long and can contain characters in the range X'40' through X'FE'.

SMP/E always puts apostrophes around each symbolic link and separates multiple values with commas with no intervening blanks.

SYMPATH('pathname1','pathname2','pathname3'...) a list of pathnames that are associated with symbolic links identified by the SYMLINK operand. A symbolic pathname can be up to 1023 characters long and can contain characters in the range X'40' through X'FE'.

SMP/E always puts apostrophes around each symbolic pathname and separates multiple values with commas with no intervening blanks.

SYMPATH appears if SYMLINK appears, otherwise it is omitted.

For information about how the pathnames and linknames are associated, see the description of the SYMPATH operand and "Example 3: Packaging a SYSMOD with a symbolic link" on page 36 in "Hierarchical file system element MCS" on page 26.

TYPE(TEXT|BINARY) the installation format for the element in a UNIX file system. SMP/E generates this parameter from information stored in the element entry in the target zone. If no setting exists for TEXT mode or BINARY mode, SMP/E does not pass this parameter to the HFS copy utility.

SHSCRIPT indicates that a UNIX shell script is to be invoked to complete the installation of the selected element. This operand applies only to hierarchical file system elements whose ELEMENT entry contains a SHSCRIPT subentry.

/directory/file is the full path specification for the shell script to be invoked. The /directory/file is derived from the SHELLSCR entry that matches the name in the hierarchical file system element's SHSCRIPT subentry. The SYSLIB subentry of the shell script identifies the ddname of the directory and the shell script name itself is the file.

PRE indicates the shell script is to be invoked before the selected element is copied to the directory in a UNIX file system. This value is obtained from the SHSCRIPT subentry of the selected element.

POST indicates the shell script is to be invoked after the selected element is copied to the directory in a UNIX file system. This value is obtained from the SHSCRIPT subentry of the selected element.

Return codes
To help you diagnose errors, GIMIAP issues messages and return codes. The messages are documented in SMP/E Messages, Codes, and Diagnosis. Here is a description of the return codes:

<table>
<thead>
<tr>
<th>Return code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>GIMIAP processing was successful. The message issued by GIMIAP states that the invocation of GIMIAP was successful.</td>
</tr>
</tbody>
</table>
GIMIAP

Return code  Meaning
04  GIMIAP processing was successful. The message issued by GIMIAP states that the invocation of GIMIAP was successful, but one or more of the following has occurred:
  • An ALIAS was not installed during element installation.
  • The COPY or HFSCOPY utility issued a non-zero return code that is less than or equal to the acceptable return code as defined in the RC operand of the INVOKE statement.
08  SMP/E processing errors occurred or a return code from the copy utility was higher than the acceptable return code specified on the INVOKE control statement. At least one element was not installed correctly, although an attempt was made to install all elements to the target library. Review the copy utility print output and SMPOUT output to determine the error. Correct the problem and rerun the job step.
12  GIMIAP encountered an invalid control statement. As a result, the elements in that job step were not installed. Once a control statement error is encountered, no more elements are processed for that job step. The statement must be corrected and the job step must be rerun. However, subsequent job steps will be processed. The message issued by GIMIAP indicates how many 80-byte records had been processed when the syntax error was encountered.
16  A severe error was encountered. As a result, the elements in that job step were not installed. After correcting the problem, the job step must be rerun. However, subsequent job steps will be processed. The message issued by GIMIAP indicates the cause of the problem, such as an I/O error on SYSIN, an open failure on SYSIN or another ddname, a syntax error on the EXEC parm, a syntax error on the control cards, or the absence of the copy utility.
20  A terminating error was encountered. The SMPOUT data set is not allocated or cannot be opened. GIMIAP is terminated.

JCL statements used in the DEIINST or HFSINST job

JCL statements are created for the DEIINST or HFSINST job during the GENERATE process. Along with the GIMIAP control statements for each job step, the JCL is composed of:
  • A JOB statement. The JOB statement describes current installation-dependent parameters. The jobname is “DEIINST” when GIMIAP is to install data elements or “HFSINST” for hierarchical file system elements.
  • One or more EXEC statements. The EXEC statement specifies PGM=GIMIAP,PARM='option'.
    The step name is the name of the target library.
    The following is an example of the EXEC statement to call GIMIAP:

```
//jobname  JOB ...
//stepname  EXEC  PGM=GIMIAP,PARM='option'
```

Figure 108. JCL to call GIMIAP

EXEC

is the statement used to call GIMIAP. The EXEC statement must specify PGM=GIMIAP. The following option may be specified on the EXEC statement PARM operand:

```
LANGUAGE=xxx
```

where xxx can be one of the following:

<table>
<thead>
<tr>
<th>Language</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENU</td>
<td>US English</td>
</tr>
<tr>
<td>JPN</td>
<td>Japanese</td>
</tr>
</tbody>
</table>
The LANGUAGE option defines which language to use for GIMIAP messages. LANGUAGE can also be specified as \texttt{L}. The LANGUAGE value will be the same as was used on GIMSMP while the GENERATE command was executed. If LANGUAGE is not specified, the default is LANGUAGE=ENU.

- **DD statements.** The DD statements define the data sets to be used by GIMIAP processing. The following ddnames are required:

  - \texttt{dlib}: The file identified by a FROMLIB operand on a COPY control statement.
  - \texttt{syslib}: The file identified by a TOLIB operand on a COPY control statement.
  - \texttt{print}: This ddname is used by the copy utility for messages and processing information. If no DDDEF is found for the print file during GENERATE processing, the default of SYSOUT=* is generated. The ddname is the same as is specified for the PRINT operand of the INVOKE statement being used for the current invocation of GIMIAP.
  - \texttt{SYSIN}: Defines the input control stream for GIMIAP.
  - \texttt{SMPOUT}: The file to be used by the GIMIAP for message writing.
  - \texttt{SYSPRINT}: Contains output when GIMIAP is processing data elements or hierarchical file system elements with shell scripts.

The following DD statements are required when processing data elements:

  - \texttt{SYSUT1}: The file identified for containing the input statements to the copy program.
  - \texttt{SYSUT3}: A work file to be used by the copy program.
  - \texttt{SYSUT4}: A work file to be used by the copy program.

The following DD statement is required when processing file system elements that require the invocation of a UNIX shell script, and the shell script uses Java commands:

  - \texttt{SMPJHOME}: Directory in the UNIX file system that contains the Java runtime.

During GIMIAP processing (invoked from the JCL produced by the GENERATE command), SMP/E can invoke a UNIX shell script to perform installation activities on behalf of a ++HFS file. If the UNIX shell script issues a Java command, the SMPJHOME DD statement is required.

An SMPJHOME DD statement is created in the HFSINST job by the GENERATE command, if an SMPJHOME DDDEF entry is defined at the time the GENERATE command is run.

Figure 109 on page 510 illustrates the JCL statements that could be generated to invoke GIMIAP processing for data elements and Figure 110 on page 511 illustrates the JCL statements that could be generated to invoke GIMIAP processing for hierarchical file system elements.
GIMIAP

//DEIINST JOB 'accounting info', MSGLEVEL=(1,1)

//tgtlib01 EXEC PGM=GIMIAP, PARM='LANGUAGE=ENU' EXECDEI
//distlib1 DD distribution library DD info.
//distlib2 DD distribution library DD info.
//tgtlib01 DD target library 1 DD info
tgtlib01

/* ----- work data sets ----------------
//SYSUT1 DD UNIT=SYSDA, SPACE=(TRK,(x,y)), DISP=(NEW,DELETE) SYSUT1
//SYSUT3 DD UNIT=SYSDA, SPACE=(TRK,(x,y)), DISP=(NEW,DELETE) SYSUT3
//SYSUT4 DD UNIT=SYSDA, SPACE=(TRK,(x,y)), DISP=(NEW,DELETE) SYSUT4
//SYSPRINT DD SYSOUT** SMPOUT
//SYSIN DD *

INVOKE COPY(IEBCOPY) RC(0) PRINT(SYSPRINT) REPLACE.
COPY FROMLIB(distlib1) TOLIB(tgtlib01).
SELECT CLIST(elem1).
SELECT CLIST(elem2) ALIAS(elemA, elemB).
COPY FROMLIB(distlib2) TOLIB(tgtlib01).
SELECT CLIST(elem3) ALIAS(elemC).
SELECT CLIST(elem4).

//tgtlib02 EXEC PGM=GIMIAP, PARM='LANGUAGE=ENU' EXECDEI
//distlib3 DD distribution library DD info.
//distlib4 DD distribution library DD info.
//tgtlib02 DD target library 2 DD info.
tgtlib02

/* ----- work data sets ----------------
//SYSUT1 DD UNIT=SYSDA, SPACE=(TRK,(x,y)), DISP=(NEW,DELETE) SYSUT1
//SYSUT3 DD UNIT=SYSDA, SPACE=(TRK,(x,y)), DISP=(NEW,DELETE) SYSUT3
//SYSUT4 DD UNIT=SYSDA, SPACE=(TRK,(x,y)), DISP=(NEW,DELETE) SYSUT4
//SYSPRINT DD SYSOUT** SMPOUT
//SYSIN DD *

INVOKE COPY(IEBCOPY) RC(0) PRINT(SYSPRINT) REPLACE.
COPY FROMLIB(distlib3) TOLIB(tgtlib02).
SELECT HELPENU(elem3).
SELECT HELPENU(elem4) ALIAS(elemD, elemE).
COPY FROMLIB(distlib4) TOLIB(tgtlib02).
SELECT HELPENU(elem5) ALIAS(elemF).
SELECT PNGENU(elem6).

Notes:

1. The job name “DEIINST” is generated during the job creation process. A single job installs the data elements.

2. The job step name is the target library. Each job step installs multiple data elements from multiple distribution libraries into a single target library.

Figure 109. Sample DEIINST job for GIMIAP
Notes:

1. The job name “HFSINST” is generated during the job creation process. A single job installs all hierarchical file system elements.

2. The job step name on the EXEC statement is the target library within a UNIX file system. Each job step installs multiple hierarchical file system elements from multiple distribution libraries into a single target library within a UNIX file system.
## Appendix A. SMP/E naming conventions

This appendix describes the naming conventions used by SMP/E. Table 61 summarizes the naming conventions you need to follow when using SMP/E. After this table are details on the naming conventions IBM follows for:

- HOLD reason IDs and classes
- Source IDs
- SYSMODs

### Table 61. Summary of SMP/E naming conventions

<table>
<thead>
<tr>
<th>Entry or value</th>
<th>Number of characters</th>
<th>Other requirements</th>
</tr>
</thead>
</table>
| DDDEF entry            | 1–8 alphabetic (A–Z), national (@, #, or $), or numeric (0–9) characters | • The first character must be alphabetic or national  
  • Must match ddname of data set |
| DLIB zone or DLIBZONE entry | 1–7 alphanumeric (A–Z, 0–9) or national ($, #, @) characters | The first character must be alphabetic. |
| Element entry          | 1–8 alphanumeric (A–Z, 0–9) characters | Naming convention for first character:  
  A–I Used by IBM  
  J–Z Available for users |
| FMID see SYSMOD ID     |                      |                    |
| FMIDSET entry          | 1–8 alphanumeric (A–Z, 0–9) characters |                    |
| Global zone or GLOBALZONE entry | 1–8 alphanumeric (A–Z, 0–9) characters | Must be GLOBAL |
| Hold class             | 1–7 alphanumeric (A–Z, 0–9) characters | Naming convention for first character:  
  A Reserved for IBM use  
  V Reserved for user-assigned values  
  For a description of the hold classes used by IBM, see “Class values” on page 516. |
| Hold reason ID: error  | 1–7 alphanumeric (A–Z, 0–9) characters | Number of the APAR used to report an error in a PTF  
  For details on the conventions IBM follows, see “Error reason IDs” on page 514. |
| Hold reason ID: system | 1–7 alphanumeric (A–Z, 0–9) characters | Naming convention for first character:  
  A–U Reserved for IBM use  
  V–Z Reserved for user-assigned values  
  For a description of the system reason IDs used by IBM, see “System reason IDs” on page 514. |
| Hold reason ID: user   | 1–7 alphanumeric (A–Z, 0–9) characters | To avoid conflicts with IBM reason IDs, follow the conventions for system reason IDs.  
  For details on the conventions IBM follows, see “User reason IDs” on page 516. |
| OPTIONS entry          | 1–8 alphanumeric (A–Z, 0–9) characters |                    |
Naming conventions

Table 61. Summary of SMP/E naming conventions (continued)

<table>
<thead>
<tr>
<th>Entry or value</th>
<th>Number of characters</th>
<th>Other requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source ID</td>
<td>1–64 characters</td>
<td>For a description of the source IDs used by IBM, see &quot;Naming conventions for source IDs&quot; on page 516.</td>
</tr>
<tr>
<td></td>
<td>in length, consisting of any nonblank character (X'41' through X'FE') except single quotation mark ('), asterisk (*), percent (%), comma (,), left parenthesis ((), and right parenthesis ()).</td>
<td></td>
</tr>
<tr>
<td>SYSMOD ID</td>
<td>7 alphanumeric</td>
<td>Must start with a letter. Naming convention for first character: A–K Used by IBM for functions L–T Available for users U Used by IBM for PTFs V–Z Used by IBM for APARs</td>
</tr>
<tr>
<td></td>
<td>(A–Z, 0–9) characters</td>
<td></td>
</tr>
<tr>
<td>Target zone or</td>
<td>1–7 alphanumeric</td>
<td>The first character must be alphabetic.</td>
</tr>
<tr>
<td>TARGETZONE entry</td>
<td>(A–Z, 0–9) or national ($, #, @) characters</td>
<td></td>
</tr>
<tr>
<td>UTILITY entry</td>
<td>1–8 alphanumeric</td>
<td>Must match the associated name specified in the appropriate OPTIONS entry Should match the name of the associated utility program</td>
</tr>
<tr>
<td></td>
<td>(A–Z, 0–9) characters</td>
<td></td>
</tr>
<tr>
<td>ZONESET entry</td>
<td>1–8 alphanumeric</td>
<td>Avoid using the same name as any of the target or DLIB zones defined in the global zone containing the ZONESET entry</td>
</tr>
<tr>
<td></td>
<td>(A–Z, 0–9) characters</td>
<td></td>
</tr>
</tbody>
</table>

Naming conventions for HOLD reason IDs and HOLD classes

The ++HOLD statement prevents SMP/E from installing a SYSMOD until some special action is taken. The type of action is indicated by the reason ID or class specified on the ++HOLD statement. A reason ID or class value can contain from 1 to 7 alphanumeric characters. To prevent conflicts between IBM- and user-specified values, there are naming conventions for the three types of HOLD reason IDs (error, system, and user), as well as for HOLD classes.

Error reason IDs

The reason ID for an error HOLD is the number of the APAR used to report an error in a PTF. Therefore, error reason IDs follow the naming conventions for APARs. These are described under “PTF, APAR, and USERMOD SYSMOD IDs” on page 518.

System reason IDs

The reason ID for a system HOLD is generally a brief indication of the kind of processing the SYSMOD requires. These are the values currently used by IBM:

<table>
<thead>
<tr>
<th>ID</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTION</td>
<td>The SYSMOD needs special handling before or during APPLY processing, ACCEPT processing, or both.</td>
</tr>
<tr>
<td>AO</td>
<td>The SYSMOD may require action to change automated operations procedures and associated data sets and user exits in products or in customer applications. The PTF cover letter describes any</td>
</tr>
</tbody>
</table>
Naming conventions

changes (such as to operator message text, operator command syntax, or expected actions for operator messages and commands) that can affect automation routines.

**DB2BIND**  
A DB2 application REBIND is required for the SYSMOD to become effective.

**DDDEF**  
Data set changes or additions as required.

**DELETE**  
The SYSMOD contains a ++DELETE MCS, which deletes a load module from the system.

**DEP**  
The SYSMOD has a software dependency.

**DOC**  
The SYSMOD has a documentation change that should be read before the SYSMOD is installed.

**DOWNLD**  
Code that is shipped with maintenance that needs to be downloaded.

**DYNACT**  
The changes supplied by the SYSMOD may be activated dynamically without requiring an IPL. The HOLD statement describes the instructions required for dynamic activation. If those instructions are not followed, then an IPL is required for the SYSMOD to take effect.

**EC**  
The SYSMOD needs a related engineering change.

**ENH**  
The SYSMOD contains an enhancement, new option or function. The HOLD statement provides information to the user regarding the implementation and use of the enhancement.

**EXIT**  
The SYSMOD contains changes that may affect a user exit. For example, the interface for an exit may be changed, an exit may need to be reassembled, or a sample exit may be changed.

**EXRF**  
The SYSMOD must be installed in both the active and the alternative Extended Recovery Facility (XRF) systems at the same time to maintain system compatibility. (If you are not running XRF, you should bypass this reason ID.)

**FULLGEN**  
The SYSMOD needs a complete system or subsystem generation to take effect.

**IOGEN**  
The SYSMOD needs a system or subsystem I/O generation to take effect.

**IPL**  
The SYSMOD requires an IPL to become effective. For example, the SYSMOD may contain changes to LPA or NUCLEUS, the changes may require a CLPA, or a failure to perform an IPL might lead to catastrophic results, such as could be caused by activation of a partial fix.

**Note:** If you plan to perform an IPL with CLPA after the SYSMOD has been applied, then no further investigation of the HOLD is required; simply bypass the IPL reason ID. However, if you are not planning to perform an IPL with CLPA, then the details of the HOLD statement must be investigated to determine what kind of actions are required to activate the SYSMOD.
Naming conventions

**MSGSKEL**  This SYSMOD contains message changes that must be compiled for translated versions of the message changes to become operational on extended TSO consoles.

If you want to use translated versions of the messages, you must run the message compiler once for the library containing the English message outlines, and once for each additional language you want to be available on your system. For details, see [z/OS MVS Planning: Operations](#).

If you want to use only the English version of the messages, you do not need to run the message compiler. You should bypass this reason ID.

**MULTSYS**  Identifies fixes that need to be applied to multiple systems, in one of three cases: preconditioning, coexistence, or exploitation.

**RESTART**  To become effective, the SYSMOD requires a special subsystem restart operation. The HOLD statement contains information regarding the required restart actions.

**User reason IDs**

The reason ID for a user HOLD is whatever you think is appropriate to describe why the SYSMOD should be held. Because IBM does not use these reason IDs, there are no restrictions on the values you can use. However, to prevent possible confusion with other IBM reason IDs, follow the naming conventions for system reason IDs.

**Class values**

A class value indicates an alternative way to release a held SYSMOD. These are the values currently used by IBM:

<table>
<thead>
<tr>
<th>Class</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERREL</td>
<td>The SYSMOD is held for an error reason ID but should be installed anyway. IBM has determined that the problem the SYSMOD resolves is significantly more critical than the error reflected by the holding APAR.</td>
</tr>
<tr>
<td>HIPER</td>
<td>The SYSMOD is held with a hold class of HIPER (High Impact)</td>
</tr>
<tr>
<td>PE</td>
<td>The SYSMOD is held with a hold class of “PTF in Error”.</td>
</tr>
<tr>
<td>UCLREL</td>
<td>UCLIN needed for the SYSMOD has been handled by IBM and no longer requires your attention.</td>
</tr>
<tr>
<td>YR2000</td>
<td>Identifies PTFs that provide Year 2000 function, or fix a Year 2000-related problem.</td>
</tr>
</tbody>
</table>

**Naming conventions for source IDs**

With the SOURCEID operand, you can associate a name (a source ID) with each SYSMOD processed by a single RECEIVE command, and then later process those SYSMODs as a group by using that source ID. The following values are currently used by IBM.

**RSU{yyymm}**  This is the indicator used to identify a Recommended Service Upgrade. For more information about RSU, see [SMP/E User’s Guide](#).

**PUT{yyymm}**  Each PTF is assigned a source ID in the format PUT{yyymm}.
Naming conventions

- *yy* is the year the PTF was made available as preventive service.
- *mm* is the month the PTF was made available as preventive service.

The PUT\*yy\*mm source ID represents a monthly accumulation of PTFs that have been COR-closed (made available for corrective service) within a given month and assigned a RETAIN VOLID indicating this. The RETAIN VOLID reflects the monthly PUT\*yy\*mm level no later than the 15th of the following month. For example, PUT0706 represents PTFs that were COR-closed in June 2008 and assigned a RETAIN VOLID of 0706 between July 1 and July 15.

**SMCCOR**

This indicates a PTF that is approved for distribution, but has not been assigned a monthly PUT\*yy\*mm source ID at the time your order was created. SMCCOR PTFs are not necessary for installation; however, they are made available for corrective service. If you plan to install any of the PTFs with this source ID, refer to the CORPE PSP bucket or latest available PUT bucket for the most current HOLD information. These PTFs are provided in case you experience the problem they fix and need to install them as corrective service.

**SMCREC**

This indicates a PTF that is approved for distribution and is recommended for installation, but has not been assigned a monthly PUT\*yy\*mm source ID at the time your order was created. These PTFs should be installed on your system. They were assigned a source ID of SMCREC for one of the following reasons:

- The PTF is needed to install a product.
- The PTF is needed to support new hardware.
- The PTF resolves an error for another PTF (PE-PTF).

To further enhance your ability to selectively install PUT\*yy\*mm and SMCREC PTFs, the additional SOURCEIDs listed below are assigned to the applicable PTFs. (There will be PUT\*yy\*mm and SMCREC PTFs that do not fit into the categories below and, therefore, are not assigned multiple source IDs.)

**HIPER**

Identifies PTFs resolving a high-impact APAR

**SPE**

Identifies PTFs that are small programming enhancements

**PRP**

Identifies PTFs that resolve PTFs in error

**OS390R\*n**

This indicates a PTF that is at the integration tested level for OS/390 Release \*n*.

**YR2000**

This identifies PTFs that provide Year 2000 function, or fix a Year 2000-related problem.

Naming conventions for SYSMODs

The specific naming conventions for a SYSMOD ID depend on the type of SYSMOD: function, PTF, APAR, or USERMOD. The following sections define IBM’s naming conventions for SYSMOD IDs. This information is provided to help you develop a naming scheme for your own SYSMODs and avoid conflicts with IBM-written SYSMODs. However, these conventions are not requirements for user-written SYSMODs, and SMP/E does not check whether a SYSMOD ID follows these conventions.
Function SYSMOD IDs

The IBM convention for the SYSMOD ID of a function SYSMOD is \textit{tcccrrr}, where:

- \textit{t} is the type of function. These are the values used by IBM:
  - A: Licensed vendor or business partner base function
  - B: Licensed vendor or business partner dependent function
  - C, D: Reserved for future use
  - E: Unlicensed base function
  - F: Unlicensed dependent function
  - G: Reserved for future use
  - H: Licensed base function
  - I: Reserved for future use
  - J: Licensed dependent function
  - K: Reserved for future use

Names of user-written functions can start with any letters other than these.

- \textit{ccc} For functions provided by IBM, this is the product code. It must be three alphanumeric characters (no vowels).

- \textit{rrr} For functions provided by IBM, this is the release number. It identifies a specific function within a product. These three characters identify a specific release of a product function and must be alphanumeric.

Note: The former convention for the SYSMOD ID of a function was \textit{tccrrr}.

PTF, APAR, and USERMOD SYSMOD IDs

The IBM convention for the SYSMOD ID of a service SYSMOD (APAR, APAR fix, PTF) or USERMOD is \textit{tannnnn}, where:

- \textit{t} identifies the type of SYSMOD. It is a single alphanumeric character. These are the values used by IBM:
  - A–K: Used by IBM for various levels of an APAR fix
  - L–T: Available for users
  - U: Used by IBM for PTFs
  - V–Z: Used by IBM for various levels of an APAR fix

- \textit{a} is any alphabetic character (A–Z). Any valid character can be used for user SYSMODs.

- \textit{nnnnn} is an additional identifier for the SYSMOD. For PTFs and APAR fixes supplied by IBM, it is a number from 00001 to 99999. Any valid characters can be used for user SYSMODs.
Appendix B. Accessibility

Publications for this product are offered in Adobe Portable Document Format (PDF) and should be compliant with accessibility standards. If you experience difficulties when using PDF files, you may view the information through the z/OS Internet Library website or the z/OS Information Center. If you continue to experience problems, send an email to mhvrdfs@us.ibm.com or write to:

IBM Corporation
Attention: MHVRDFS Reader Comments
Department H6MA, Building 707
2455 South Road
Poughkeepsie, NY 12601-5400
USA

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use software products successfully. The major accessibility features in z/OS enable users to:

- Use assistive technologies such as screen readers and screen magnifier software
- Operate specific or equivalent features using only the keyboard
- Customize display attributes such as color, contrast, and font size

Using assistive technologies

Assistive technology products, such as screen readers, function with the user interfaces found in z/OS. Consult the assistive technology documentation for specific information when using such products to access z/OS interfaces.

Keyboard navigation of the user interface

Users can access z/OS user interfaces using TSO/E or ISPF. Refer to z/OS TSO/E Primer, z/OS TSO/E User’s Guide, and z/OS ISPF User’s Guide Vol I for information about accessing TSO/E and ISPF interfaces. These guides describe how to use TSO/E and ISPF, including the use of keyboard shortcuts or function keys (PF keys). Each guide includes the default settings for the PF keys and explains how to modify their functions.

z/OS information

z/OS information is accessible using screen readers with the BookServer or Library Server versions of z/OS books in the Internet library at:

http://www.ibm.com/systems/z/os/zos/bkserv/

One exception is command syntax that is published in railroad track format, which is accessible using screen readers with the Information Center, as described in "Dotted decimal syntax diagrams."

Dotted decimal syntax diagrams

Syntax diagrams are provided in dotted decimal format for users accessing the Information Center using a screen reader. In dotted decimal format, each syntax element is written on a separate line. If two or more syntax elements are always
present together (or always absent together), they can appear on the same line, because they can be considered as a single compound syntax element.

Each line starts with a dotted decimal number; for example, 3 or 3.1 or 3.1.1. To hear these numbers correctly, make sure that your screen reader is set to read out punctuation. All the syntax elements that have the same dotted decimal number (for example, all the syntax elements that have the number 3.1) are mutually exclusive alternatives. If you hear the lines 3.1 USERID and 3.1 SYSTEMID, you know that your syntax can include either USERID or SYSTEMID, but not both.

The dotted decimal numbering level denotes the level of nesting. For example, if a syntax element with dotted decimal number 3 is followed by a series of syntax elements with dotted decimal number 3.1, all the syntax elements numbered 3.1 are subordinate to the syntax element numbered 3.

Certain words and symbols are used next to the dotted decimal numbers to add information about the syntax elements. Occasionally, these words and symbols might occur at the beginning of the element itself. For ease of identification, if the word or symbol is a part of the syntax element, it is preceded by the backslash (\) character. The * symbol can be used next to a dotted decimal number to indicate that the syntax element repeats. For example, syntax element *FILE with dotted decimal number 3 is given the format 3 \* FILE. Format 3* FILE indicates that syntax element FILE repeats. Format 3* \* FILE indicates that syntax element * FILE repeats.

Characters such as commas, which are used to separate a string of syntax elements, are shown in the syntax just before the items they separate. These characters can appear on the same line as each item, or on a separate line with the same dotted decimal number as the relevant items. The line can also show another symbol giving information about the syntax elements. For example, the lines 5.1*, 5.1 LASTRUN, and 5.1 DELETE mean that if you use more than one of the LASTRUN and DELETE syntax elements, the elements must be separated by a comma. If no separator is given, assume that you use a blank to separate each syntax element.

If a syntax element is preceded by the % symbol, this indicates a reference that is defined elsewhere. The string following the % symbol is the name of a syntax fragment rather than a literal. For example, the line 2.1 %OP1 means that you should refer to separate syntax fragment OP1.

The following words and symbols are used next to the dotted decimal numbers:

- ? means an optional syntax element. A dotted decimal number followed by the ? symbol indicates that all the syntax elements with a corresponding dotted decimal number, and any subordinate syntax elements, are optional. If there is only one syntax element with a dotted decimal number, the ? symbol is displayed on the same line as the syntax element, (for example 5? NOTIFY). If there is more than one syntax element with a dotted decimal number, the ? symbol is displayed on a line by itself, followed by the syntax elements that are optional. For example, if you hear the lines 5 ?, 5 NOTIFY, and 5 UPDATE, you know that syntax elements NOTIFY and UPDATE are optional; that is, you can choose one or none of them. The ? symbol is equivalent to a bypass line in a railroad diagram.

- ! means a default syntax element. A dotted decimal number followed by the ! symbol and a syntax element indicate that the syntax element is the default option for all syntax elements that share the same dotted decimal number. Only
one of the syntax elements that share the same dotted decimal number can specify a ! symbol. For example, if you hear the lines 2? FILE, 2.1! (KEEP), and 2.1 (DELETE), you know that (KEEP) is the default option for the FILE keyword. In this example, if you include the FILE keyword but do not specify an option, default option KEEP will be applied. A default option also applies to the next higher dotted decimal number. In this example, if the FILE keyword is omitted, default FILE(KEEP) is used. However, if you hear the lines 2? FILE, 2.1, 2.1.1! (KEEP), and 2.1.1 (DELETE), the default option KEEP only applies to the next higher dotted decimal number, 2.1 (which does not have an associated keyword), and does not apply to 2? FILE. Nothing is used if the keyword FILE is omitted.

- * means a syntax element that can be repeated 0 or more times. A dotted decimal number followed by the * symbol indicates that this syntax element can be used zero or more times; that is, it is optional and can be repeated. For example, if you hear the line 5.1* data area, you know that you can include one data area, more than one data area, or no data area. If you hear the lines 3*, 3 HOST, and 3 STATE, you know that you can include HOST, STATE, both together, or nothing.

Notes:
1. If a dotted decimal number has an asterisk (*) next to it and there is only one item with that dotted decimal number, you can repeat that same item more than once.
2. If a dotted decimal number has an asterisk next to it and several items have that dotted decimal number, you can use more than one item from the list, but you cannot use the items more than once each. In the previous example, you could write HOST STATE, but you could not write HOST HOST.
3. The * symbol is equivalent to a loop-back line in a railroad syntax diagram.

- + means a syntax element that must be included one or more times. A dotted decimal number followed by the + symbol indicates that this syntax element must be included one or more times; that is, it must be included at least once and can be repeated. For example, if you hear the line 6.1+ data area, you must include at least one data area. If you hear the lines 2+, 2 HOST, and 2 STATE, you know that you must include HOST, STATE, or both. Similar to the * symbol, the + symbol can only repeat a particular item if it is the only item with that dotted decimal number. The + symbol, like the * symbol, is equivalent to a loop-back line in a railroad syntax diagram.
Notices

This information was developed for products and services offered in the USA.

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

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
USA

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

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

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

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

© Copyright IBM Corp. 1986, 2011
Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Mail Station P300
2455 South Road
Poughkeepsie, 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 information 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.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

**Programming interface information**

This publication is intended to help you understand the output from SMP/E processing.

This publication also documents intended Programming Interfaces that allow the customer to write programs to obtain the services of IBM SMP/E for z/OS, V3R6. This Programming Interface information is identified where it occurs by an introductory statement to a chapter or section.

This publication also documents information that is *not* intended to be used as a Programming Interface of IBM SMP/E for z/OS, V3R6. This information is identified where it occurs by an introductory statement to a chapter or section.

**Policy for unsupported hardware**

Various z/OS elements, such as DFSMS, HCD, 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.

**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" at [http://www.ibm.com/legal/copytrade.shtml](http://www.ibm.com/legal/copytrade.shtml)
Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.
Index

Special characters

?PKGHash
  tag syntax 490
++APAR MCS
  examples 7
  operands
    DESCRIPTION 6, 23
    FILES 6
    REWORK 7
    RFDSNPFX 7
    SYSMOD ID 7
  overview 6
  syntax 6
++ASMIN
  ASSEM subentry 183
++ASSIGN MCS
  coding considerations 8
  examples 9
  operands
    SOURCEID 8
    TO 8
  overview 8
  syntax 8
++DELETE MCS
  coding considerations 18
  examples 19
  operands
    ALIAS 17
    name 18
    SYSLIB 18
  overview 17
  syntax 17
++ENDASMIN
  ASSEM subentry 183
++ENDLMODIN
  LMOD subentry 254
++FEATURE MCS
  coding considerations 22
  example 22
  operands 21
    DESCRIPTION 21
    FMID 21
    PRODUCT 21
    REWORK 21
  overview 21
  syntax 21
++FUNCTION MCS
  coding considerations 24
  examples 25
  operands
    FESN 23
    FILES 23
    REWORK 24
    RFDSNPFX 24
    SYSMOD ID 24
  overview 23
  syntax 23
++HOLD MCS
  coding considerations 42
  examples 43
++HOLD MCS (continued)
  operands
    CATEGORY 38
    CLASS 38
    COMMENT 39
    DATE 39
    ERROR 39
    FIXCAT 39
    FMID 40
    REASON 40
    RESOLVER 42
    SYSMOD ID 42
    SYSTEM 39
    USER 39
  overview 37
  syntax 37
++IF MCS
  coding considerations 45
  examples 46
  operands
    FMID 45
    REQ 45
    THEN 45
  overview 45
  syntax 45
++JAR MCS
  coding considerations 54
  examples 54
  operands
    DELETE 48
    DISTLIB 49
    JARPARM 50
    LINK 50
    name 51
    PARM 51
    RELFILE 51
    RMD 51
    SSCRIPT 51
    SYMLINK 52
    SYMPATH 53
    SYSLIB 53
    TXLIB 53
    UMID 53
    VERSION 54
  overview 47
  syntax 47
++JCLIN MCS
  coding considerations 62
  examples 62
  operands
    ASM 60
    CALLLIBS 60
    COPY 60
    LKED 61
    OPCODE 61
    RELFILE 61
    TXLIB 62
    UPDATE 62
  overview 60
  syntax 60
++JARUPD MCS
  coding considerations 69
  example 69
  operands
    ASSEM 65
    DELETE 66
    DISTLIB 66
    DISTMOD 66
    DISTSRC 66
    MALIAS 67
    name 67
    PREFIX 67
    RELFILE 68
    RMD 68
    SSI 68
    SYSLIB 68
    TXLIB 68
    UMID 68
    VERSION 68
  overview 65
  syntax 65
++MACUPD MCS
  coding considerations 73
  example 74
  operands
    ASSEM 72
  DISTLIB 72
  DISTMOD 73
  DISTSRC 73
  MALIAS 73
  name 73
  PREFIX 73
  SYSLIB 73
  OVERVIEW 72
  syntax 72
++MOD MCS
  coding considerations 81
  examples 81
  operands
    CSECT 77
    DALLAS 77
    DELETE 77
    DISTLIB 77
    LEPARM 78
    LKLIB 80

© Copyright IBM Corp. 1986, 2011
AMODE=31
  LMOD subentry 251
  MOD entry 277
AMODE=64
  LMOD subentry 251
  MOD entry 277
AMODE=ANY
  LMOD subentry 251
  MOD entry 277
AMODE=MIN
  LMOD subentry 251
  MOD entry 277
AMS
  OPTIONS subentry 292
AMS utility
  default values 292, 350
  UTILITY entry for 292
AO reason ID 41, 102, 515
APAR fixes
  defining 6
  MCS statement for 6
  naming conventions 518
SYMSMOD subentry
  for distribution zone 322
  global zone 336
  target zone 322
APARS
ORDER subentry 303
APPPDATE
  SYMSMOD subentry
  target zone 325
APPID
  SYMSMOD subentry
  global zone 336
  application programming interface
  for SMP/E CSI 359
  for SMP/E exit routines 439
  for SMP/E shell scripts 413
  library change file records 419
APPLY
  SYMSMOD subentry
  target zone 322
APPLY command
  LEPARM processing 82
APPTIME
  SYMSMOD subentry
  target zone 325
ARCHDEF
  tag syntax 457, 489
archid
  ARCHDEF attribute 457
  FILEDEF attribute 483
archive files
  contents of 492
  creating 492
  extracting data from 454
  in GIMZIP package 480
  introduction to 479
ARCHSEG
  tag syntax 490
ASM
  +JCLIN MCS operand 60
  OPTIONS subentry 292
  UTILITY entry for 350
ASMA90 utility
  See assembler utility
ASSEM
  ++MAC MCS operand 65
  ++MACUPD MCS operand 72
SYMSMOD subentry
  distribution zone 322
  target zone 322
ASSEM entry
  listing 184
  subentries
    ASSEMBLER INPUT 183
    LASTUPD 183
    LASTUPDTYPE 183
  summary 183
  UCLIN for 186
  unloading 185
ASSEMBLE
  MOD entry 275
  assembler
    data structures in 401
    sample GIMAPI program 403
  ASSEMBLER INPUT
    ASSEM subentry 183
  assembler OPCODEs
    in SMPPPARM 137
  assembler utility
    ++JCLIN MCS operand 60
    default values 292, 350, 352
  OPTIONS entry 292
  specifying on JCLIN 60
  UTILITY entry for 292, 350
B
BACKUP entries
  listing 188
  subentries 188
  summary 188
  UCLIN for 189
BINARY
  hierarchical file system element
  entry 229
  hierarchical file system element MCS operand 28
  binder
  See also link-edit utility
  UTILITY entry for 350
BLOCK
  DDDEF entry 197
  BMPPTZN option of GIMXTRX 476
BYPASS
  SYMSMOD subentry
  distribution zone 323
  target zone 323
C
  data structures in 396
  sample GIMAPI program 403
CALL
  effect of CALLLIBS subentry on 351
  LMOD subentry 251
  CALLLIBS
    LMOD subentry 249
    specifying on JCLIN 60
  CASE
    LMOD subentry 251
  CATALOG
    DDDEF entry 197
  CATEGORY
    ++HOLD MCS operand 38
  CBPDO
    Recommended service upgrade 516
    ccc value of FMID 518
  CHANGEFILE
    OPTIONS subentry 292
  CHANGEFILE(YES) subentry
    required for library change file records 419
  CIFREQ
    SYMSMOD subentry
    distribution zone 323
    target zone 323
  CLASS
    ++HOLD MCS operand 38
    naming conventions 516
    values
      ERREL 38, 516
      HIPER 38, 516
      PE 38, 516
      UCLIN 38, 516
      YR2000 38, 516
  CLIENT, defining 141
  CLIST data element
    installing in variable block library 10
  COMMENT
    ++HOLD MCS operand 39
  COMP
    OPTIONS subentry 293
    UTILITY entry 350
  COMPACT
    OPTIONS subentry 293
  COMPACT option of GIMCPTS 448
  compacting SMPPTS data set
    with GIMCPTS 447
  COMPAT
    LMOD subentry 252
    MOD entry 278
  COMPAT option of GIMSMP 443
  compress utility
    default values 293, 350
  OPTIONS entry 293
  UTILITY entry for 293, 350
  CONCAT
    DDDEF entry 198
    concatenating data sets
    DDDEF entry 198
    not allowed for SMPPTS spill data sets 158
  SYSLIB data sets
    ++MAC 69
    ++MACUPD 74
    order 167
    SMPMTS 154
    use 167
    conditional requisites
    defining 45, 55
  CONTENT
    ALL value
    ORDER subentry 302
CONTENT (continued)
ORDER subentry 302
COPY
++JCLIN MCS operand 60
INVOKE control statement operand 504
LMOD subentry 249
OPTIONS subentry 293
UTILITY entry for 350
COPY control statement for GIMIAP 504
copy utility
default values 293, 350
OPTIONS entry 293
specifying on JCLIN 60
UTILITY entry for 293, 350
COREquisite SYSMODs
++IF MCS 45
++JAR MCS 53
defining 119
CSECT
++MOD MCS operand 77
MOD entry 275
CSI
application programming interface
for 359
ASSEM entry
distribution zone 183
target zone 183
data element entry
distribution zone 191
target zone 191
DDDEF entry
distribution zone 197
global zone 197
target zone 197
defining 145
DLIB entry
distribution zone 211
target zone 211
DLIBZONE entry 215
FEATURE entry 219
FINDEX entry 221
GIMAPI for 359
GLOBALZONE entry 224
HOLD DATA entry 239
LMOD entry
distribution zone 249
target zone 249
PRODUCT entry 306
PROGRAM entry
distribution zone 309
target zone 309
CSI option
of GIMSSMP 443
CYLINDERS
DDDEF entry 197
D
DALIASS
++MOD MCS operand 77
++ZAP MCS operand 124
MOD entry 276
data element entry
listing 192
subentries 191
data element entry (continued)
summary 191
UCLIN for 195
unloading 194
data element MCS
coding considerations 16
examples 16
operands
ALIAS 14
DELETE 14
DISTLIB 14
FROMMD 14, 29, 49, 61, 66, 78, 108
name 15
RELFILE 15
RMID 15
SYSLIB 15
TXLIB 15
VERSION 15
overview 10
syntax 13
data elements
adding 10, 16
defining 10
deleting
data element MCS 14
listing 192
renaming 16, 105
replacing
data element MCS 10
SYSMOD subentry
distribution zone 323
global zone 337
target zone 323
unloading 194
data set organization 171
data sets
CLIENT 141
defining 141
distribution library 141
FILE 142
link library 142
LKLIB 142
ORDERSERVER 142
OUTFILE 143
SERVER 143
SMCPLNT 143
SMIPNRTL 144
SMPCPATH 144
SMPSCL 145
SMPCI 145
SMPI 146
SMPI 146
SMPIDEBUG 147
SMPI 147
SMPIO 148
SMPIHOLD 148
SMPI 149
SMPPRINT 149
SMPI 149
SMPIHOME 150
SMPI 150
SMPI 151
SMPI 151
SMPI 152
SMPI 152
SMPI 153
SMPI 153
SMPI 154
SMPI 154
data sets (continued)
SMPOUT 155
SMPPARM 155
SMPTFIN 156
SMPPSTS 157
SMPPSTS sp1 158
SMPPUNCH 158
SMPRPT 159
SMPSDS 159
SMPSNAP 160
SMPSVR 160
SMPPSTS 160
SMPTLIB 161
SMPTLOAD 162
SMPTKDIR 163
SMPPWRK1 163
SMPPWRK2 164
SMPPWRK3 164
SMPPWRK4 165
SMPPWRK5 165
SYSIN 166
SYSLIB 166
SYSPRT 167
SYSPUNCH 168
SYSUT1 168
SYSUT4 168
target library 169
text library 169
TXLIB 169
zonedname 169
data structures
for QUERY command 361
in assembler 401
in C 396
in PL/I 398
data transformation service routine
(GIMDTS) 450
DATACLASS
DDDEF entry 198
DATASET
DDDEF entry 198
DATE
++HOLD MCS operand 39
++RELEASE MCS operand 101
DATE option
of GIMSSMP 443
DB2BIND reason ID 41, 102, 515
DC
LMOD subentry 252
MOD entry 278
DDDEF entry
listing 202
subentries
BLOCK 197
CATALOG 197
CONCAT 198
CYLINDERS 197
DATACLAS 198
DATASET 198
DELETE 197
DIR 199
DSNTYPE 199
DSPREFIX 199
KEEP 197
MGMTCLAS 200
MOD 200
NEW 200

530 SMP/E V3R6.0 for z/OS V1R13.0 Reference
DDDEF entry (continued)
  subentries (continued)
     OLD 200
     PATH 200
     PROTECT 201
     SHR 200
     SPACE 201
     STORCLAS 201
     SYOUT 201
     TRACKS 197
     UNIT 202
     VOLUME 202
     WAIT 202

DELETENote
  summary 197

UCLIN for
  distribution zone 205
  global zone 205
  target zone 205

DDDEF reason ID 41, 102, 515
  default utilities used by SMP/E 350

defaults
  OPTIONS entry
    distribution zone 215
    global zone 224
    on the SET command 292

OPTIONS subentry
  target zone 346

SMPTLIB
  data set space 294
  disposition 162

subentries
  NOPURGE 296
  NOREJECT 296
  ORDERRET 297
  PAGELEN 297
  PEMA 297
  RETRYDDN 298
  SAVEMTS 298
  SAVESTS 298
  SUPPHOLD 299

utility programs
  access method services 292, 350
  assembler 292, 350, 352
  compress 293, 350
  copy 293, 350
  link-edit utility 295, 350, 352
  retry 298, 350
  superzap 299, 350
  update 299, 350

DEFINECLUSTER
  tag syntax 499
  defining data sets
    data set descriptions 141
    DDDEF entry 197

DEIINST job, built by GENERATE 503
DELBY
  SYMSMOD subentry
    distribution zone 323
    target zone 323

DELETE
  ++JAR MCS operand 48
  ++MAC MCS operand 66
  ++MOD MCS operand 77
  ++PROGRAM MCS operand 92
  ++SRC MCS operand 107
  ++VER MCS operand 118

DELETE (continued)
  data element MCS operand 14
  DDDEF entry 197
  hierarchical file system element MCS operand 29
  MCS statement 17
  SYMSMOD subentry
    distribution zone 323
    global zone 337
    target zone 323

DELETE reason ID 41, 102, 515
  deleting elements
    data elements 14
    hierarchical file system elements 29
    JAR elements 48
    macros 66
    modules 77
    modules, MCS for 84
    source 107

  deleting functions
    ++VER MCS 118
    dummy function SYMSMOD 122
    example of 122

  deleting load modules
    ++DELETE MCS 17

DELLMOD
  SYMSMOD subentry
    distribution zone 323, 337
    target zone 323, 337

DEP reason ID 41, 102, 515
  description
    FILEDEF attribute 483
    GIMZIP attribute 482

DESCRIPTION
  ++APAR MCS operand 6, 23
  ++FEATURE MCS operand 21
  ++PRODUCT MCS operand 89
  ++USERMOD MCS operand 115
  FEATURE entry 219
  PRODUCT entry 306
  SYMSMOD subentry
    distribution zone 323, 337
    target zone 323, 337

DIR
  DDDEF entry 199
  directories
    SMPDIR 147
    SMPTNTS 154
    SMPWDIR 163

  directory information for pathname 200
  disability 519

DISTLIB (continued)
  hierarchical file system element MCS operand 29
  JAR entry 242
  MAC entry 267
  MOD entry 276
  PROGRAM entry 309
  SRC entry 314

DISTLIB (continued)
  ++MAC MCS operand 66
  ++MACUPD MCS operand 73
  ++SRC MCS operand 108
  ++SRCUPD MCS operand 112

DISTLIB (continued)
  distribution libraries
  defining 141
  distribution zone
    ASSEM entry 183
    data element entry 191
    DDDEF entry 197
    DLIB entry 211
    DLIBZONE entry 215
  hierarchical file system element
    entry 229
    JAR entry 242
    LMOD entry 249
    MAC entry 267
    MOD entry 275
    PROGRAM entry 309
    SRC entry 314
    SYMSMOD entry 322
    zone description provided by
    user 216

DISTSRC
  ++MAC MCS operand 66
  ++MACUPD MCS operand 73

DLIB entry
  listing 212
  subentries
    LASTUPD 211
    LASTUPDTYPE 212
    SYSLIB 212
    UCL syntax 211

  summary 211
  UCLIN for 213
  unloading 213

DLIBZONE entry
  listing 216
  subentries
    ACCJCL 215
    name 215
    OPTIONS 215
    RELATED 215
    SREL 216
    UCL syntax 215
    UPLEVEL 216, 347
    ZDESC 216

  summary 215
  UCLIN for 217

DLMOD
  SYMSMOD subentry
    distribution zone 323, 337
    target zone 323, 337

  DOC reason ID 41, 102, 515
  DOWNLD reason ID 41, 102, 515
  DOWNLDATE
    ORDER subentry 303
ENTRY

ORDER subentry 303
DSNTYPE
DDDEF entry 199
DSPREFIX
conflict with RFDSNPFX value 293
DDDEF entry (global zone only) 199
OPTIONS subentry 293
SYSSMOD subentry
global zone 341
DSSPACE
OPTIONS subentry 293
dummy data sets
defined in DDDEF entry 199
not allowed for SMPPPTS 157
not allowed for SMPPPTS spill data sets 158
dummy function SYSSMOD to delete another function 122
dummy volumes
restriction for SMPTLIB data sets 161
DYNACT reason ID 41, 102, 515
dynamiac allocation
CSI parameter on EXEC statement for GIMSMMP 443
DDDEF entry 197
specifying data sets in GIMDDALC 129

EC

EC reason ID 41, 102, 515

elements 84
deleting
data elements, MCS for 14
hierarchical file system elements, MCS for 29
JAR elements, MCS for 48
macros, MCS for 66
modules, MCS for 77
source, MCS for 107

tomoving
++MOVE MCS 85
renaming
data elements, MCS for 16
hierarchical file system elements, MCS for 35
macros, MCS for 71
modules, MCS for 83
source, MCS for 110
ELEMMOV
SYSSMOD subentry
distribution zone 324, 337
target zone 324, 337

EMOVE

SYSSMOD subentry
distribution zone 324, 337
target zone 324, 337
ENH reason ID 41, 102, 515
Enhanced HOLDDATA
in ++HOLD MCS 39

ENTRY

ASSEM entry 183
BACKUP entries 188
CSI 173

EXEC statement

for GIMCPPTS 447
for GIMDTS 450
for GIMGTPKG 452

ENTRY

EXEC statement (continued)

for GIMUNZIP 455
for GIMXMLD 464
for GIMTR 471
for GIMZIP 480
for invoking SMP/E 443
EXIT reason ID 41, 102, 515

exit routines
specifying in GIMEXITS
summary 133
exit routines for SMP/E 439
parameter list mapping 439
RECEIVE exit (Exit 1) 439
Retry Exit (Exit 2) 441
summary 439

EXPAND option
of GIMCPTS 448

EXEC statement

++ZAP input 125
expanding SMPPPTS data set with GIMCPTS 447
EXPR reason ID 41, 102, 515

EXRTYDD
OPTIONS subentry 294
extensible stylesheet language (XSL) file browser for 492
definition of 487
external HOLDDATA 37, 239

FEATURE

SYSSMOD subentry 324, 337
FEATURE entry
listing 219

FEATURE subentries
description 219
FMID 219
name 219
PRODUCT 219
RECDATE 219
RECTIME 219
REWORK 219
UCLDATE 219
UCLTIME 219
summary 219

FILEATTR

tag syntax 496

FILEDATA operand

for CLIENT 141, 144, 149, 150, 160, 166
for ORDERSERVER 142

FILEDATA operand (continued)

FILEDATA operand (continued)
FILEDATA operand (continued)
for SERVER 143
for SMPCLUINT 144
for SMPDEBUG 147
for SMPLIST 151
for SMPOUT 155
for SMPPTFIN 156
for SMPRPT 159
FILEDEF
  tag syntax 482
FILELIST
  tag syntax 496
FILES
  ++APAR MCS operand 6
  ++FUNCTION MCS operand 23
  ++PTF MCS operand 97
  ++USERMOD MCS operand 115
FILL
  LMOD subentry 252
  MOD entry 278
  filter parameter
  of QUERY command 363
FIXCAT
  ++HOLD MCS operand 39
  ++RELEASE MCS operand 100
  OPTIONS subentry 294
  reason IDs 42, 101
FMID
  ++FEATURE MCS operand 21
  ++IF MCS operand 45
  ++MOVE MCS operand 85
  ++RELEASE MCS operand 101
  ++VER MCS operand 118
  data element entry 191
  FEATURE entry 219
  FMIDSET entry 221
  GLOBALZONE subentry 224
function SYSMODs
defining 23
functions
defining 23
G
GENASM
  MAC entry 267
GIMAPI
  data structures for 359
  FREE command of 393
  OVERVIEW of 359
  QUERY command of 359
  sample assembler program 403
  sample C program 403
  sample PL/I program 403
  used by GIMXTRX 476
  VERSION command of 394
GIMCPTS 447
GIMDDALC control statements
defined in SMPPARM 129
GIMDDALC member
  sample 132
  use of 129
GIMDTS
  inline data elements 16, 34
  inline program elements 94
  SUMMARY of 450
GIMEXITS control statements
defined in SMPPARM 133
GIMEXITS member
  in SMPPARM 133
  sample 135
  specifying exit routines in
  control statements for 133
  RECEIVE exit routine 439
  Retry exit routine 441
  use of 134
GIMFAF.XML 492
GIMGTPKG
  summary 451
GIMIAP 503
GIMMPUXP 439
GIMOPCDE
  sample member supplied 137, 156
  overriding values in, example of 139
  GIMOPCDE members
  operands 138
GIMPAFXML
  browser for 492
  definition of 487
GIMMSMP 443
GIMUNZIP
  calling 480
  example of using 485
  package control tags 481
  processing 487
  sample job stream 485
  summary 479
  SYSSIN data set for 166
  tag syntax 482
  global zone
  DDDEF entry 197
  FEATURE entry 219
  FMIDSET entry 221
  GLOBALZONE entry 224
  HOLDDATA entry 239
  OPTIONS entry 292
  ORDER entry 302
  PRODUCT entry 306
  SYSMOD entry 336
  upgrade level for 216, 224
  UTILTY entry 350
  zone description provided by
  user 225
  ZONESET entry 357
GLOBALZONE entry
  listing 226
  subentries
    FMID 224
    OPTIONS 224
    SREL 224
    UCL syntax 224
    UPGLEVEL 224
    ZONEDESCRIPTION 225
    ZONEINDEX 225
    summary 224
    UCLIN for 226
H
HASH option
  of GIMUNZIP 455
HEWLH096 utility
  See link-edit utility
HFSCOPY
  INVOKE control statement
  operand 504
  OPTIONS subentry 295
HFSINST job, built by GENERATE 503
hierarchical file system
  copy utility
    default values 350
  copy utility
    default values 350
  MCS statement 26
  placing pre-built program object
  in 34
  syntax 27
hierarchical file system element entry
  hierarchical file system element
    MCS 26
    listing 232
    subentries
      BINARY 229
      DISTLIB 229
      FMID 229
      LASTUPD 230
      LASTUPDTYPE 230
      LINK 230
      PARM 230
      RMID 231
      SHSCRIPT 231
      SYMLINK 231
      SYMPATH 232
      SYSLIB 232
      TEXT 229
      UCL syntax 229
    summary 229
    unloading 235

hierarchical file system element MCS
coding considerations 34
elements 34
operands
  BINARY 28
  DELETE 29
  DISTLIB 29
  LINK 30
  name 31
  PARM 31
  RELFILE 31
  RMID 31
  SHSCRIPT 31
  SYMLINK 32
  SYMPATH 33
  SYSLIB 33
  TEXT 33
  TXMLIB 34
  VERSION 34
overview 26

hierarchical file system elements 26
adding 26, 35
defining 26
deleting 29
example of symbolic link 36

hierarchical file system element
MCS 29
renaming 35, 105
replacing
  hierarchical file system element
    MCS 26
  SYSMOD subentry
distribution zone 324
global zone 338
target zone 324
HIPER class value 38, 516
HOBSET
LMOD subentry 252
MOD entry 278
HOLD
  See HOLD reason IDs
HOLD reason IDs
class values
  +HOLD MCS 38
  ERREL 38, 516
  HIPER 38, 516

HOLD reason IDs (continued)
class values (continued)
  naming conventions 516
    PE 38, 516
    UCLIN 38, 516
    YR2000 38, 516
error reason IDs
  +HOLD MCS 40
  +RELEASE MCS 101
fix category reason IDs
  +HOLD MCS 42
  fixcat reason IDs
    +HOLD MCS 101
  naming conventions 514
system reason IDs
  +HOLD MCS 40
  +RELEASE MCS 101
ACTION 41, 101, 514
AO 41, 102, 515
DB2BIND 41, 102, 515
DDDEF 41, 102, 515
DELETE 41, 102, 515
DEP 41, 102, 515
DOC 41, 102, 515
EC 41, 102, 515
ENH 41, 102, 515
EXIT 41, 102, 515
EXRF 41, 102, 515
FULLGEN 41, 102, 515
IPL 42, 103, 516
MSGSKEL 42, 103, 516
MULTSYS 42, 103, 516
RESTART 42, 103, 516
user reason IDs
  +HOLD MCS 42
  +RELEASE MCS 103
  naming conventions 516
HOLDDATA
  See also HOLD reason IDs
external 37, 239
internal 37, 103, 239
HOLDDATA entry
  listing 239
  saving after ACCEPT processing (NOPURGE) 296
  summary 239
HOLDErrOR
  SYSMOD subentry
global zone 338
target zone 325
HOLDFIXCAT
  SYSMOD subentry
global zone 338
HOLDSYSTEM
  SYSMOD subentry
global zone 338
HOLDUSER
  SYSMOD subentry
global zone 338

HOLD reason IDs (continued)
class values
  +HOLD MCS 38
  ERREL 38, 516
  HIPER 38, 516

I

ICSF
  use by GIMUNZIP 455
  use by GIMZIP 480
IDCAMS utility
  See AMS utility
IEBCOPY utility
  See copy utility
IEBUPDTE utility
  See update utility
IEHIOSUP utility
  OPTIONS entry 295
  UTILITY entry for 295
IEWBLINK utility
  See link-edit utility
IEWL utility
  See link-edit utility
IF
  MCS statement 45
IFREQ
  SYSMOD subentry
distribution zone 324
target zone 324
IMASPZAP utility
  See superzap utility
INFILE
  defining 142
  inline elements 16, 34, 54, 69, 81, 94, 109
  inline JCLIN
  packaging 62
INSTALLDATE
  SYSMOD subentry
distribution zone 325
target zone 325
INSTALLTIME
  SYSMOD subentry
distribution zone 325
target zone 325
  internal HOLDDATA 37, 103, 239
  INVOKE control statement
    for GIMIAPI 503
  IOGEN reason ID 41, 102, 515
  IOSUP
    OPTIONS subentry 295
  IPL reason ID 42, 103, 516

J

JAR
  SYSMOD subentry
distribution zone 325
global zone 338
target zone 325
  ++JAR MCS 47, 48
    adding 47
    defining 47
    deleting 48
    replacing 47
  JAR entry
    ++JAR MCS 47
    listing 245
  subentries
    DISTLIB 242
    FMID 242
    LASTUPD 242

534  SMP/E V3R6.0 for z/OS V1R13.0 Reference
JAR entry (continued)
subentries (continued)
  LASTUPDTYPE 243
  LINK 243
  PARM 243
  RMID 243
  SHSCRIPT 244
  SYMLINK 244
  SYMPATH 244
  SYSLIB 245
  UIMID 245
  summary 242
  unloading 248
JARPARM
  ++JAR MCS operand 50
  ++JARUPD MCS operand 56
JARUPD
SYSMOD subentry
distribution zone 325
  global zone 339
target zone 325
JCLIN
GIMOPCDE member for assembler
OPCODEs in SMPPARM 137
GIMOPCDE members
summary 137
syntax 138
packaged inline 62
SYSMOD subentry
distribution zone 325
global zone 339
target zone 325
JARPARM
++JAR MCS operand 50
++JARUPD MCS operand 56
JARUPD
SYSMOD subentry
distribution zone 325
  global zone 339
target zone 325
LC_ALL variable
  used for shell scripts 414
LEPARAM
  ++MOD MCS operand 78
  example of use 82
library change file records
  file records 419
  record types 419
SMPDATA1 data set for 146
SMPDATA2 data set for 146
spill processing for 146
LINK
  ++JAR MCS operand 50
  ++JARUPD MCS operand 56
  hierarchical file system element MCS operand 30
JAR subentry 243
LINK command
  hierarchical file system element entry 230
link edits
  multi-tasking of 353
link-edit utility
default values 295, 350, 352
effect of LEPARM 82
OPTIONS entry 295
specifying on JCLIN 61
UTILITY entry for 295, 350
LIST
  INVOKE control statement operand 504
UTILITY entry 351
LKED
  ++JCLIN MCS operand 61
  OPTIONS subentry 295
  UTILlITY entry for 350
LKED ATTRIBUTES
  LMOD subentry 250
  MOD entry 277
LKED CONTROL
  LMOD subentry 254
LKLIB
  ++MOD MCS operand 80
  ++PROGRAM MCS operand 93
  defining 142
LLA
effect on ++ZAP processing 126
LMOD
  ++MOD MCS operand 80
  ++MOVE MCS operand 86
MOD entry 280
LMOD entry
listing 256
  subentries
    CALLLIBS 249
    COPY 249
    LASTUPD 250
    LASTUPDTYPE 250
    LKED ATTRIBUTES 250
    LKED CONTROL 254
    MODELL 254
    RC 254
    SIDEDECKLIB 255
    SYSLIB 255
    UTIN 255
    XZMOD 256
    XZMODP 256
  summary 249
  UCLIN for 263
  unloading 260
load modules
  aliases for 81
  deleting 17
  moving 85
  renaming 105
  updating 126
  local shared resources (LSR) 145, 170
LSTTZN option
  of GIMXTRX 475
K
KEEP
  DDDEF entry 197
keyboard 519
L
language abbreviations used on MCS for
data elements 12
LANGUAGE option
  of GIMCPTS 448
  of GIMTPKG 452
  of GIMSMP 444
  of GIMUNZIP 455
  of GIMXSID 464
  of GIMZIP 480
LASTSUP
SYSMOD subentry
distribution zone 325
target zone 325
LASTUPD
  ASSEM subentry 183
data element entry 192
DLIB entry 212
  hierarchical file system element entry 230
JAR subentry 243
  LMOD subentry 250
  MAC entry 268
  MOD entry 277
  PROGRAM entry 309
  SRC entry 315
  SYSMOD subentry
distribution zone 325
target zone 325
LC_ALL variable
  used for shell scripts 414
LEPARAM
  ++MOD MCS operand 78
  example of use 82
library change file records
  file records 419
  record types 419
SMPDATA1 data set for 146
SMPDATA2 data set for 146
spill processing for 146
LINK
  ++JAR MCS operand 50
  ++JARUPD MCS operand 56
  hierarchical file system element MCS operand 30
JAR subentry 243
LINK command
  hierarchical file system element entry 230
link edits
  multi-tasking of 353
link-edit utility
default values 295, 350, 352
effect of LEPARM 82
OPTIONS entry 295
specifying on JCLIN 61
UTILITY entry for 295, 350
LIST
  INVOKE control statement operand 504
UTILITY entry 351
LKED
  ++JCLIN MCS operand 61
  OPTIONS subentry 295
  UTILlITY entry for 350
LKED ATTRIBUTES
  LMOD subentry 250
  MOD entry 277
LKED CONTROL
  LMOD subentry 254
LKLIB
  ++MOD MCS operand 80
  ++PROGRAM MCS operand 93
  defining 142
LLA
effect on ++ZAP processing 126
LMOD
  ++MOD MCS operand 80
  ++MOVE MCS operand 86
MOD entry 280
LMOD entry
listing 256
  subentries
    CALLLIBS 249
    COPY 249
    LASTUPD 250
    LASTUPDTYPE 250
    LKED ATTRIBUTES 250
    LKED CONTROL 254
    MODELL 254
    RC 254
    SIDEDECKLIB 255
    SYSLIB 255
    UTIN 255
    XZMOD 256
    XZMODP 256
  summary 249
  UCLIN for 263
  unloading 260
load modules
  aliases for 81
  deleting 17
  moving 85
  renaming 105
  updating 126
  local shared resources (LSR) 145, 170
LSTTZN option
  of GIMXTRX 475
M
MAC
  ++MOVE MCS operand 86
  MCS statement 65
  OPCODE statement 138
SYSMOD subentry
distribution zone 326
global zone 339
target zone 326
MAC entry
  ++MAC MCS 65
  listing 269
  subentries
    DISTLIB 267
    FMID 267
    GENASAM 267
    LASTUPD 268
    LASTUPDTYPE 268
    MALIAS 268
    RMID 268
    SYSLIB 268
    UCLIN syntax 267
    UIMID 268
  summary 267
  UCLIN for 271
  unloading 270
macros
  ++MAC MCS 66
  ++MACUPD MCS 72
  adding 65
  defining 65
deleting
  ++MAC MCS 66

Index 535
macros (continued)
  MCS statement 66, 72
  moving 85
  renaming 71, 105
  replacing
    ++MAC MCS 65
  updating
    ++MACUPD MCS 72
MACUPD
  MCS statement 72
  SYSMOD subentry
    distribution zone 326
    global zone 339
    target zone 326
mainframe
  education xi
MALIAS
  ++MAC MCS operand 67
  ++MACUPD MCS operand 73
  MAC entry 268
master CSI
  specified on EXEC statement 443
  use 145
MAXBLK
  LMOD subentry 252
MOD entry 278
MCS entry
  listing 273
  subentries 273
  summary 273
MCS statements
  ++APAR 6
  ++ASSIGN 8
  ++DELETE 17
  ++FUNCTION 23
  ++HOLD 37
  ++IF 45
  ++JAR 47
  ++JARUPD 55
  ++JCLIN 60
  ++MAC 65
  ++MACUPD 73
  ++MOD 76
  ++MOVE 85
  ++NULL 88
  ++PRODUCT 89
  ++PROGRAM 92
  ++PTF 97
  ++RELEASE 100
  ++RENAME 105
  ++SRC 107
  ++SRCUPD 112
  ++USERMOD 115
  ++VER 118
  ++ZAP 124
  data element 10
  general syntax rules 2
  hierarchical file system 26
  overview 5
MGMTCLAS
  DDDEF entry 200
MOD
  ++MOVE MCS operand 86
  DDDEF entry 200
  MCS statement 76
MOD (continued)
  SYSMOD subentry
    distribution zone 326
    global zone 339
    target zone 326
MOD entry
  ++MOD MCS 76
  listing 281
  subentries
    ASSEMBLE 275
    CSECT 275
    DALIAS 276
    DISTLIB 276
    FMID 276
    LASTUPD 276
    LASTUPDTYPE 277
    LIKED ATTRIBUTES 277
    LMOD 280
    RMID 280
    RMIDASM 280
    TALIAS 280
    UMID 280
    XZLMOD 280
    XZLMODP 281
  summary 275
  UCLIN for 287
  unloading 285
MODEL
  LMOD subentry 254
  modules
    ++MOD MCS 76
    ++ZAP MCS 124
  adding 76
  adding to an existing load module 82
  defining 76
  deleting
    ++MOD MCS 77
    ++ZAP MCS 124
  moving 85
  renaming 83, 105
  replacing
    ++MOD MCS 76
    ++PROGRAM MCS 92
  updating
    ++ZAP MCS 124
  moving elements
    ++MOVE MCS 85
  moving load modules
    ++MOVE MCS 85
MSGFILTER
  OPTIONS subentry 296
  MSGSKEI reason ID 42, 103, 516
  MSGWIDTH
  OPTIONS subentry 295
  MTSMAC entry
  saving through the OPTIONS subentry 298
  subentries 290
  summary 290
  UCLIN for 290
  multi-tasking of link edits 353
  multilingual support
  language abbreviations used on MCS for data elements 12
  MULTSYS reason ID 42, 103, 516
  N
name
  ++DELETE MCS operand 18
  ++FEATURE MCS operand 21
  ++FEATURE MCS operand 21
  ++IF MCS operand 45
  ++IF MCS operand 45
  ++JAR MCS operand 51
  ++JARUPD MCS operand 57
  ++MAC MCS operand 67
  ++MACUPD MCS operand 73
  ++MOD MCS operand 80
  ++MOVE MCS operand 86
  ++PROGRAM MCS operand 94
  ++SRC MCS operand 108
  ++SRCUPD MCS operand 112
  ++ZAP MCS operand 124
  ARCHDEF attribute 457
  data element MCS operand 15
  FEATURE entry 219
  FILEDEF attribute 482
  hierarchical file system element MCS operand 31
NAME
  UTILITY entry 351
  naming conventions
    CLASS 516
    HOLD reason IDs 514
    reason IDs 514
    SOURCEIDs 516
    summary 513
    SYSMOD IDs
      APARs 518
      function 518
      PTFs 518
      summary 517
      USERMODs 518
    zones 215, 346
  national language identifiers used on
  MCS for data elements 12
NCAL
  effect of CALLLIBS subentry on 351
  LMOD subentry 252
NE
  LMOD subentry 252
  MOD entry 278
  negative prerequisite SYSMODs
  defining with the NPRE operand 119
NEW
  DDDEF entry 200
  newname
  ARCHDEF attribute 458
  NOCALL
  LMOD subentry 252
  MOD entry 278
  NOPURGE
  default value 296
  OPTIONS subentry 296
  NOREJECT
  default value 296
  OPTIONS subentry 296
  notices 523
  NPRE
  ++VER MCS operand 119
  SYSMOD subentry
    distribution zone 326
    global zone 339
    target zone 326
NULL
  MCS statement 88
NULLFILE used for dummy data sets not allowed for SMPPTS 157 not allowed for SMPPTS spill data sets 158

OPTIONS subentry defining default target zone 346
ORDER entry listing 303
subentries
APARS 303
CONTENT 302
DOWNLDATE 303
DOWNLTIME 303
ORDERDATE 303
ORDERID 303
ORDERENTRY 303
ORDERENTRY 303
ORDERENTRY 303
ORDERENTRY 303
ORDERENTRY 303
ORDERENTRY 303
PKGID 303
PTFS 303
STATUS 302
USERID 303
ZONES 303
summary 302
UCLIN for 304
ORDERDATE
ORDER entry 303
ORDERID
ORDER entry 303
ORDERRET
default value 297
OPTIONS subentry 296
ORDERSERVER
ORDER entry 303
ORDERSERVER, defining 142
ORDERENTRY
ORDER entry 303
organization of SMP/E data sets 171
OS21 element entry
UCLIN for 286
OS90Rn SOURCEID 517
OUTFILE
defining 143
OVLY
Lmod subentry 253
MOD entry 278

packing SYMODs (continued)
relative files (RELFILES) (continued)
FILES operand on hierarchical file
system element MCS 31
reference material 23
RELFILE operand on
++JCLIN 61, 62
RELFILE operand on ++MOD 80
RELFILE operand on ++PTF 97
RELFILE operand on ++SRC 108
RFSDSNPFX operand on
++APAR 7
RFSDSNPFX operand on
++FUNCTION 24
RFSDSNPFX operand on
++PTF 98
RFSDSNPFX operand on
++USERMOD 116
TXLIB 34, 53, 58, 62, 68, 81, 109
PAGELEN
default value 297
OPTIONS subentry 297
PARM
++JAR MCS operand 51
++JARUPD MCS operand 57
hierarchical file system element
entry 230
hierarchical file system element MCS
operand 31
JAR subentry 243
UTILITY entry 352
PATH
DDDEF entry 200
PATH variable
used for shell scripts 414
pathname, DDDEF entry for 200
PATHOPTS operand
for CLIENT 141, 144, 149, 150, 160,
166
for ORDERENTRY 142
for SERVER 143
for SMPCLNT 144
for SMPDEBUG 147
for SMPLIST 151
for SMPOUT 155
for SMPPTFIN 156
for SMPPTPUNCH 159
for SMPPRINT 159
pax command
use by GIMUNZIP 461
use by GIMZIP 492
PDSE
specification in DDDEF entry 199
PE class value 38, 516
PEMAX
default value 297
OPTIONS subentry 297
performance
local shared resources (LSR) feature of
VSAM 145, 170
PKGDEF
tag syntax 488
PKGID
ORDER subentry 303
PL/1
data structures in 398
sample GIMAPI program 403

Index 537
RESTIME
SYSMOD subentry
target zone 327
RESTORE
SYSMOD subentry
target zone 327
restrictions
dummy volumes for SMPTLIB data sets 161
retry
exit routine 441
RETRY
excluding data sets 294
including data sets 298
OPTIONS subentry 298
UTILITY entry for 350
retry utility
default values 298, 350
defaults 298, 350
OPTIONS subentry 298
UTILITY entry for 298, 350
RETRYDDN
OPTIONS subentry 298
return codes
for GIMIAP 507
utilities
maximum acceptable value, specifying 354
summary 354
REUS
LMOD subentry 253
MOD entry 279
REUS(NONE)
LMOD subentry 253
MOD entry 279
REWORK
++APAR MCS operand 7
++FEATURE MCS operand 21
++FUNCTION MCS operand 24
++PRODUCT MCS operand 91
++PTF MCS operand 98
++USERMOD MCS operand 116
FEATURE entry 219
PRODUCT entry 307
SYSMOD subentry
distribution zone 328
global zone 340
target zone 328
RFDSNPFX
++APAR MCS operand 7
++FEATURE MCS operand 21
++FUNCTION MCS operand 24
++PRODUCT MCS operand 91
++PTF MCS operand 98
++USERMOD MCS operand 116
conflict with DSPREFIX value 293
RLMOD
SYSMOD subentry
distribution zone 328, 340
target zone 328, 340
RMID (continued)
hierarchical file system element entry 231
hierarchical file system element MCS operand 31
JAR subentry 243
MAC entry 268
MOD entry 280
PROGRAM entry 310
SRC entry 315
RMIDASM
MOD entry 280
RMODE=24
LMOD subentry 253
MOD entry 279
RMODE=31
LMOD subentry 253, 279
RMODE=ANY
LMOD subentry 253
MOD entry 279
RMODE=SPLIT
LMOD subentry 253
MOD entry 279
S
sample GIMAPI program 403
SAVEMTS
default value 298
OPTIONS subentry 298
S wealthiest
default value 298
OPTIONS subentry 298
SCTR
LMOD subentry 253
MOD entry 279
SELECT control statement
for GIMIAP 505
SERVER, defining 143
service routines 447
GIMDTS 450
GIMGTPKG 451
GIMUNZIP 454
GIMXSID 463
GIMXTRX 471
GIMZIP 479
SETSSI statement
++ZAP input 125
shell scripts for SMP/E
description 413
example 415
returning control from 415
ShopzSeries
GIMXSID service routine 463
GIMXTRX service routine 471
shortcut keys 519
SHR
DDDEF entry 200
SHSCRIPT
++JAR MCS operand 51
hierarchical file system element entry 231
hierarchical file system element MCS operand 31
JAR subentry 244
SELECT control statement
operand 505
SIDEDECKLIB
LMOD subentry 253
SMCCOR SOURCEID
naming conventions 517
SMCREC SOUCEID
naming conventions 517
SMODTYPE
additional subentry type for GIMAPI 367
value for superseded only
SYSMODs 381, 384
SMP_Action variable
used for shell scripts 414
SMP_Directory variable
used for shell scripts 413
SMP_File variable
used for shell scripts 414
SMP_Phase variable
used for shell scripts 414
SMP/E data set entries 171
SMP/E data sets, descriptions of 141
SMP/E MCS statements 5
SMP/E service routines 447
SMPCNTRL option
of GIMGTPKG 453
SMPCNTRL, defining 143
SMPCPATH option
go of GIMGTPKG 452
of GIMUNZIP 456
SMPCPATH, defining 144
SMPCSI
hierarchical file system element entry
distribution zone 229
target zone 229
JAR entry
distribution zone 242
target zone 242
MAC entry
distribution zone 267
target zone 267
MOD entry
distribution zone 275
target zone 275
OPTIONS subentry 292
ORDER subentry 302
SRC entry
distribution zone 314
target zone 314
SYSMOD entry
distribution zone 322
global zone 336
target zone 322
TARGETZONE entry 346
UTILITY entry 350
ZONESET entry 357
SMPDATA1
defining 146
SMPDATA2
defining 146
SMPDEBUG
defining 147
SMPDIR
defining 147
SMPDIR option
of GIMUNZIP 456
defining 148
for definition side deck library 148
defining 148
defining 149
defining 149
defining 150
defining 148
of GIMGTPKG 452
of GIMUNZIP 456
defining 150
defining 151
defining 151
defining 152
defining 153
MTSMAC entry 290
saving MTSMAC entries after ACCEPT processing (SAVEMTS) 298
defining 166
defining 154
defining 154
defining 152
defining 154
of GIMGTPKG 452
defining 154
defining 155
defining 154
of GIMGTPKG 452
of GIMUNZIP 456
of GIMMSID 464
defining 155
members of 129
syntax rules 2
defining 156
compacting 447
defining 157
expanding 447
MCS entry 273
saving MCS entries after ACCEPT processing (NOPURGE) 296
saving MCS entries after RESTORE processing (NOREJECT) 296
defining 158
defining 158
defining 159
BACKUP entries 188
SMPSCDS (continued)
defining 159
SMPSNAP
defining 160
SMPSSVR option
of GIMGTPKG 453
SMPVR, defining 160
SMPSTS
ACCEPT processing 320
APPLY processing 320
defining 160
saving STSSRC entries after ACCEPT processing (SAVESTS) 298
STSSRC entry 320
SMPLIB
ACCEPT processing 296
conflict between DSPREFIX value and RFDSNPFX value 293
defaults
disposition 162
space 294
defining 161
DSPREFIX value for data set name
DDDEF entry 199
OPTIONS subentry 293
DSSPACE value for 293
saving after ACCEPT processing (NOPURGE) 296
SMPLOAD
defining 162
SMPWKDIR
defining 163
SMPWKDIR option
of GIMUNZIP 456
of GIMZZIP 481
SMPWRK1
defining 163
SMPWRK2
defining 164
SMPWRK3
defining 164
SMPWRK4
defining 165
SMPWRK6
defining 165
SMPXTOUT option
of GIMMSID 464
of GIMXTRX 472
SMS
DATACLAS specification in DDDEF entry 198
MGMTCLAS specification in DDDEF entry 200
PDSE (LIBRARY) specification in DDDEF entry 199
STORCLAS specification in DDDEF entry 201
UNIT specification not needed in DDDEF entry 202
VOLUME specification not needed in DDDEF entry 202
software inventory data
GIMMSID service routine 463
source
++SRC MCS 107
++SRCUPD MCS 112
adding 107, 110
source (continued)
defining 107
deleting
++SRC MCS 107
moving 85
renaming 105, 110
replacing
++SRC MCS 107
upgrading
++SRCUPD MCS 112
SOURCEID
++ASSIGN MCS operand 8
assigning 8
naming conventions 516
OS390Rn 517
PUTyymm 517
RSUyymm 516
SMCCOR 517
SMCREC 517
SYSMOD subentry
distribution zone 328
global zone 340
target zone 328
YR2000 517
SPACE
DDDEF entry 201
SRC
++MOVE MCS operand 86
MCS statement 107
SYSMOD subentry
distribution zone 328
global zone 340
target zone 328
SRC entry
++SRC MCS 107
listing 315
subentries
DISTLIB 314
FMID 314
LASTUPD 314
LASTUPDTYPE 315
RMD 315
SYSLIB 315
UMID 315
summary 314
UCLIN for 318
unloading 317
SRCUPD
MCS statement 112
SYSMOD subentry
distribution zone 328
global zone 340
target zone 328
SREL
++PRODUCT MCS operand 90
++VER MCS operand 119
DLIBZONE subentry 216
GLOBALZONE subentry 224
SYSMOD subentry
global zone 340
TARGETZONE subentry 346
SRI
PRODUCT entry 306
SSI
++MAC MCS operand 68
++SRC MCS operand 109
SYSPPUNCH
  defining 168
SYSTEM
  See also SYSMOD ID
++HOLD MCS operand 39
++RELEASE MCS operand 100
reason IDs 40, 101, 514
system generation
  indicated by REGEN subentry 327
system reason IDs
  naming conventions 514
values
  ACTION 41, 101, 514
  AO 41, 102, 515
  DB2BIND 41, 102, 515
  DDDEF 41, 102, 515
  DELETE 41, 102, 515
  DEP 41, 102, 515
  DOC 41, 102, 515
  DOWNLD 41, 102, 515
  DYNACT 41, 102, 515
  EC 41, 102, 515
  ENH 41, 102, 515
  EXIT 41, 102, 515
  EXRF 41, 102, 515
  FULLGEN 41, 102, 515
  IPL 42, 103, 516
  IGEN 42, 103, 516
  IOCUSYS 42, 103, 516
  RESTART 42, 103, 516
SYSTSPRT
  specified in UTILITY entry 353
SYSTSPRT option
  of GIMXTRX 472
SYSUT1
  defining 168
SYSUT3 option
  of GIMUNZIP 456
SYSUT4
  defining 168
SYSUT4 option
  of GIMUNZIP 456
SZAP
SYSMOD subentry
distribution zone 329
global zone 340
target zone 329

T

TALIAS
  ++MOD MCS operand 80
  ++ZAP MCS operand 124
MOD entry 280
target libraries
  defining 169
target zone
ASSEM entry 183
cross-zone, automatic updating 347
cross-zone, information added by LINK MODULE command 347
data element entry 191
DDDEF entry 197
DLIB entry 211
TARGETZONE entry 346
target zone (continued)
target zone
history 229
target zone

UCL syntax (continued)
data element entry 191
DDDEF entry
distribution zone 197
target zone 197
DLIB entry 211
DLIBZONE entry 215
FMIDSET entry 221
GLOBALZONE entry 224
hierarchical file system element entry
  summary 229
LMOD entry
  summary 249
MAC entry
  summary 267
MOD entry
  summary 275
OPTIONS entry
  summary 292
ORDER entry 302
PROGRAM entry 309
SRC entry
  summary 314
SYSMOD entry
distribution zone 322
global zone 336
target zone 322
TARGETZONE entry 346
UTILITY entry
  summary 351
ZONESET entry 357
UCLDATE
  FEATURE entry 219
  PRODUCT entry 306
SYSMOD subentry
  distribution zone 329
target zone 329
UCLIN class value 38, 516
UCLTIME
  FEATURE entry 219
  PRODUCT entry 307
SYSMOD subentry
  distribution zone 329
target zone 329
UMID
  ++JAR MCS operand 53
  ++MAC MCS operand 68
  ++MOD MCS operand 81
  ++SRC MCS operand 109
JAR subentry 245
MAC entry 268
MOD entry 280
SRC entry 315
uniform resource locator (URL)
in ++PRODUCT MCS 90
UNIT
  DDDEF entry 202
UNIX file system
data sets residing in
  CLIENT 141, 144, 149, 150, 160, 166
  ORDERSERVER 142
  SERVER 143
  SMPCLN 144
  SMPDEBUG 147
  SMPLIST 151
  SMPOUT 155
  SMPPFIN 156
UNIX file system (continued)
data sets residing in (continued)
SMPPUNCH 159
SMPRPT 159
HFS copy utility
interaction with hierarchical file
system element entry 352
OPTIONS entry 295
UTILITY entry for 295
UPCASE
LMOD subentry 254
MOD entry 279
UPDATE
++JCLIN MCS operand 62
OPTIONS subentry 299
UTILITY entry for 350
update utility
default values 299, 350
OPTIONS subentry 299
restrictions 353
specifying on JCLIN 62
UTILITY entry for 299, 350
UPLEVEL
DLIBZONE subentry 216
GLOBALZONE subentry 224
TARGETZONE subentry 347
URL
++PRODUCT MCS operand 90
PRODUCT entry 306
usage recommendations
library change file records 436
USER
See also user reason IDs
++HOLD MCS operand 39
++RELEASE MCS operand 100
reason IDs 42, 103, 516
user modification
See USERMOD
user reason IDs
naming conventions 516
USERID
ORDER subentry 303
USERMOD
defining 115
naming conventions 518
source update example 113
SYSMOD subentry
distribution zone 329
global zone 341
target zone 329
UTILITY entry
listing 354
subentries
LIST 351
NAME 351
PARM 352
PRINT 353
RC 354
summary 351
summary 350
UCLIN for 355
utility programs
default values
access method services
(AMS) 292, 350
assembler 292, 350
compress 293, 350
utility programs (continued)
default values (continued)
copy 293, 350
HFS copy 295
hierarchical file system copy 350
link-edit utility 295, 350
retry 298, 350
superzap 299, 350
update 299, 350
OPTIONS entry pointer to
access method services 292
assembler 292
compress 293
copy 293
HFS copy utility 295
IEHIOSUP utility 295
link-edit utility 295
superzap 299
update 299
OPTIONS subentry pointer to
retry 298, 352
parameters passed to 350, 352
return codes for
default values 350
threshold 354
SYSPRINT output 350, 353
UTILITY entries for 350
UTIN
LMOD subentry 255
V
variable block library
installing CLIST data element in 10
VENDOR
++PRODUCT MCS operand 90
PRODUCT entry 306
VER
MCS statement 118
VERNUM
SYSMOD subentry
distribution zone 329
target zone 329
VERSION
++JAR MCS operand 54
++MAC MCS operand 68
++MOD MCS operand 81
++PROGRAM MCS operand 94
++SRC MCS operand 109
++VER MCS operand 120
data element MCS operand 15
hierarchical file system element MCS operand 34
SYSMOD subentry
distribution zone 329
global zone 341
target zone 329
VERSION command
g GI MAPI 394
VLF
effect on ++ZAP processing 126
volume
ARCHDEF attribute 457
FILEDEF attribute 484
VOLUME
DDDEF entry 202
vv.rr.mm
PRODUCT entry 306
W
WAIT
DDDEF entry 202
WAIT option
g IM X SID 464
WAITFORDSN
DDDEF entry 202
X
XML statements
gen eral syntax rules 3
XREF
tsample LIST output
ASSEM subentry 184
data element entry 193
hierarchical file system element entry 233
JAR entry 246
LMOD subentry 258
MAC entry 269
MOD entry 283
PROGRAM entry 311
SRC entry 316
SYSPRINT entry 333
XZL MOD
MOD entry 280
XZL MODP
MOD entry 281
XZ MOD
LMOD subentry 256
XZ MODP
LMOD subentry 256
XZREQCHK
ZONESET entry 357
Y
YR2000 class value 38, 516
YR2000 SOURCEID
naming conventions 517
Z
z/OS Basic Skills information center xi
z/OS SecureWay Security Server
protection for data sets 201
ZAP
MCS statement 124
OPTIONS subentry 299
UTILITY entry for 350
ZONE
  ZONESET entry  357
ZONEDESCRIPTION
  DLIBZONE subentry  216
  GLOBALZONE subentry  225
  TARGETZONE subentry  348
ZONEINDEX
  GLOBALZONE subentry  225
  zones
    defining  169
    naming conventions  215, 346
ZONES
  ORDER subentry  303
ZONESET entry
  listing  357
  subentries
    XZREQCHK  357
    ZONE  357
  summary  357
  UCLIN for  358