

Software Delivery



Standard Packaging Rules for MVS-Based Products

Software Delivery



Standard Packaging Rules for MVS-Based Products

Note!

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

Tenth Edition, September 1999

This book replaces the previous edition, SC23-3695-08, which is now obsolete. Significant changes or additions to text and illustrations are indicated by a vertical line to the left of the change.

This edition applies to the following licensed programs:

- OS/390 Version 1, program number 5645-001
- OS/390 Version 2, program number 5647-A01
- System Modification Program Extended (SMP/E) Release 8.1, program number 5668-949

Order IBM publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address given below.

IBM welcomes your comments. A form for readers' comments appears at the back of this publication. If the form has been removed, address your comments to:

IBM Corporation
Owner, Standard Rules for Packaging MVS-Based Products
Department 31RA, Mail Station P526
522 South Road
Poughkeepsie, NY 12601-5400
United States of America

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

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

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

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

© **Copyright International Business Machines Corporation 1986, 1999. All rights reserved.**

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

Contents

Notices	ix
Trademarks	ix
About This Book	xi
How this Book Is Organized	xi
Conventions for Rules, Restrictions, and Recommendations	xii
Summary of Changes	xv
Revision SC23-3695-09 (September 1999)	xv
Revision SC23-3695-08 (March 1999)	xv
Revision SC23-3695-07 (March 1998)	xvii
Revision SC23-3695-06 (September 1997)	xvii
Revision SC23-3695-05 (March 1997)	xvii
Chapter 1. Introduction to MVS Product Processes	1
1.1 What Is Product Packaging?	1
1.2 Evolution of Product Packaging	1
1.3 Getting Products and Service from Development Libraries onto Users' Systems	2
1.3.1 Packaging and Distributing the SYSMODs	2
1.3.2 Writing the Installation Documentation	4
1.3.3 Installing the SYSMODs	5
1.4 A Simple Packaging Example	7
Chapter 2. Assessing Your Product's Packaging Requirements and Considerations	9
Chapter 3. Rules for Packaging Methods	11
3.1 Relative File Tapes	11
3.1.1 Format and Contents of the RELFILE Tape	12
3.1.2 Creating the RELFILE Tape	15
3.2 Inline Data	15
3.2.1 Example of Inline Element Updates	16
3.2.2 Example of Inline JCLIN Data	16
3.3 Indirect Libraries	17
3.3.1 Example of a RELFILE Tape with JCLIN Data in an Indirect Library	18
3.3.2 Example of Using Indirect Libraries Instead of a RELFILE Tape	19
Chapter 4. SYSMOD Types and Relationships	21
4.1 Types of SYSMODs	21
4.1.1 Functions	21
4.1.2 PTFs	23
4.1.3 APAR Fixes	24
4.1.4 USERMODs	24
4.2 Defining SYSMOD Relationships	24
4.2.1 Conditional and Unconditional Relationships	25
4.2.2 Hierarchy of SYSMOD Types	25
4.2.3 Specific SYSMOD Relationships	26
4.2.4 Coexisting SYSMODs	29

Chapter 5. Fundamental Packaging Considerations	33
5.1 Installation Methods	33
5.2 Evaluating SYSMOD Relationships	35
5.3 Adding FMIDs	36
5.4 Record Length, Record Format, and Block Size Requirements	36
5.5 Shared Libraries	39
5.6 Avoiding UCLIN	40
Chapter 6. Elements and Load Modules	43
6.1 General Packaging Rules, Restrictions, and Recommendations for Elements	44
6.2 Element Ownership	45
6.3 Using Aliases for Elements	45
6.4 Data Element Types	45
6.5 Hierarchical File System (HFS) Element Types	47
6.6 Shared Load Modules	48
6.7 Sample JCL and Data	49
6.8 Language-Sensitive Elements	52
Chapter 7. Using MCS Statements to Define Products	53
7.1 ++FUNCTION Statement	54
7.1.1 Specifying the SYSMOD ID (sysmod_id)	54
7.1.2 Identifying the REWORK Date (REWORK)	54
7.1.3 Specifying the Prefix for RELFILE Data Sets (RFDSNPF)	54
7.1.4 Specifying Copyright Information	55
7.2 ++VER Statement	56
7.2.1 General Packaging Rules (++VER)	56
7.2.2 Identifying the SREL	57
7.2.3 Identifying a SYSMOD's Base Function (FMID)	57
7.2.4 Deleting SYSMODs (DELETE)	57
7.2.5 Specifying Mutually Exclusive SYSMODs (NPRE)	60
7.2.6 Specifying Prerequisite Relationships (PRE)	60
7.2.7 Superseding SYSMODS (SUP)	61
7.2.8 Defining Ownership (VERSION)	64
7.3 ++IF Statement	67
7.3.1 Specifying the Function to which the Condition Applies (FMID)	67
7.3.2 Specifying Requisite Conditions (REQ)	67
7.4 ++HOLD Statement	70
7.5 ++element Statement	71
Chapter 8. Using MCS Statements to Manipulate Elements and Load Modules	73
8.1 Moving Elements and Load Modules (++MOVE)	75
8.2 Renaming Load Modules (++RENAME)	78
8.3 Deleting Load Modules (++DELETE)	81
8.4 Deleting Elements from Libraries and SMP/E Data Sets	83
8.5 Enabling Load Module Changes at the CSECT Level (++MOD CSECT)	84
8.6 Defining Ownership of Elements (++element VERSION)	84

Chapter 9. Using JCLIN	87
9.1 Providing JCLIN Data for Function SYSMODs	87
9.2 When Do You Need JCLIN?	88
9.3 General Packaging Rules for JCLIN Data	89
9.4 Assembler Steps	90
9.5 Copy Steps	91
9.5.1 Considerations for the SELECT Statement for Copy Operations	92
9.6 Link-Edit Steps	94
9.6.1 JCLIN Processing of DD Statements in Link-Edit Steps	96
9.6.2 Link-Edit Control Statements	97
9.6.3 Link-Edit Attribute Parameters	106
9.6.4 Cross-Product Load Modules for Products Installed in the Same Zone	107
9.6.5 Cross-Product Load Modules for Products Installed in Different Zones	109
9.6.6 Adding or Changing Load Modules in a PTF	111
9.7 Examples of JCLIN Data	112
9.7.1 JCLIN Data for Modules	112
9.7.2 JCLIN Data for Macros and Source	116
9.7.3 JCLIN Data for an Assembler Step to Create a Module from Source	116
9.7.4 JCLIN for Using the Link-Edit Automatic Library Call Function	117
9.7.5 JCLIN Data for Load Modules Residing in a Hierarchical File System	119
Chapter 10. Naming Conventions	121
10.1 Component Codes	121
10.2 SYSMOD IDs for Functions	121
10.3 Element, Alias, and Load Module Names	122
10.3.1 NLS Considerations for Element Types	123
10.3.2 Elements with the Same Name	123
10.3.3 Alias Names	123
10.4 Library Names	124
Chapter 11. Packaging for National Language Support (NLS)	127
11.1 Element Types for Translated Elements	128
Chapter 12. Packaging for Special Situations	131
12.1 High-Level Languages	131
12.1.1 Support in SMP/E Release 8 and Later for the Automatic Library Call Facility	131
12.1.2 If You Cannot Use the Automatic Library Call Facility	131
12.2 Using the C Language Prelinker	134
12.2.1 Example of a Product Requiring the C Prelinker	135
12.3 Packaging Workstation Code to Be Installed on the Host	136
12.4 Hierarchical File System (HFS)	136
Chapter 13. SYSMOD Packaging Examples	137
13.1 Conventions Used in This Chapter	137
13.2 Example 1: A Stand-Alone Function	138
13.2.1 Initial Release	138
13.2.2 PTF Service for the Initial Release	138
13.2.3 PTF Service That Depends on Previous Service	139
13.2.4 Ensuring That a Fix for a Previous Release Is Not Lost	140
13.2.5 Replacing the Initial Release	141
13.3 Example 2: Corequisite Base Functions	142

13.3.1	Initial Releases of Corequisite Functions	142
13.3.2	PTF Service for One of the Base Functions	143
13.3.3	Cross-Product Service between Corequisite Base Functions	143
13.3.4	Deleting and Superseding a Base Function	144
13.4	Example 3: Dependent Functions	145
13.4.1	Initial Release of a Dependent Function	146
13.4.2	PTF Service for a Dependent Function	146
13.4.3	Corequisite PTFs with an Element Common to the Base and Dependent Functions	147
13.4.4	Corequisite PTFs with All Elements Common to Base and Dependent Functions	149
13.4.5	Deleting a Dependent Function Without Superseding It	152
13.4.6	Establishing the Order of Additional Dependent Functions	152
13.4.7	Conditional Corequisite Dependent Functions	153
13.5	Example 4: Base Functions with Prerequisites	153
13.5.1	Initial Release of a Base Function with a Functional Prerequisite	153
13.5.2	Dependency on an SPE or Service for Another Base Function	154
13.5.3	Cross-Product Service for a Base Function with a Prerequisite	155
13.6	Example 5: Mutually Exclusive Dependent Functions	156
13.7	Example 6: Functions Supporting More Than One Language	157
13.7.1	A Base Function Supporting Two Languages	157
13.7.2	PTF Service for Language-Sensitive Elements	158
13.7.3	Supporting Two Languages for a Base Function and Its Related Dependent Function	159
13.7.4	PTF Service for Common Language-Sensitive Elements	160
13.8	Changing the Contents of Products	161
13.8.1	Adding Elements	162
13.8.2	Combining Elements	162
13.8.3	Migrating Elements by Updating Both Functions	163
13.8.4	Migrating Elements by Using a PTF	164

Appendixes 165

Appendix A. Summary of Rules, Restrictions, and Recommendations	167
A.1 Rules	167
A.2 Restrictions	191
A.3 Recommendations	194

Appendix B. MVS Service Packaging Rules	209
B.1 Introduction	209
B.1.1 Service Terminology	210
B.2 MVS Service Packaging Rules	212
B.2.1 PTF Size, Format, and Content	212
B.2.2 Standard PTF Structure	213
B.2.3 PTF Cover Letter	222
B.3 IBM Service Delivery	234
B.3.1 Service Process Initialization	234
B.3.2 PTF Submission	235
B.4 Naming Conventions for Service	236

Appendix C. Mapping of Old Rule Numbers to New Rule Numbers	237
--	-----

Glossary	243
Bibliography and Classes	249
SMP/E Books in the OS/390 Library	249
The SMP/E Release 8.1 Library	249
Classes and Self-Study Courses for SMP/E	250
Index	253

Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, are the user's responsibility.

IBM may have patents or pending patent applications covering subject matter 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
500 Columbus Avenue
Thornwood, NY 10594
USA

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Mail Station P300
522 South Road
Poughkeepsie, NY 12601-5400
USA
Attention: Information Request

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

Trademarks

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

BookManager
CBIPO
CBPDO
DB2
IBM
MVS/ESA
MVS/SP
MVS/XA
OS/390
OS/390 Security Server

About This Book

This publication references the Release 8.1 and OS/390 manuals using generic titles. See Table 20 on page 249 and Table 21 on page 249 to map to the appropriate reference manual for your system.

This publication is designed to help you package software that can be installed by System Modification Program Extended (SMP/E) on an MVS system. It is directed at two groups of users:

- Packagers outside of IBM whose software will be distributed through IBM Software Delivery.
These packagers must follow the rules and restrictions in this publication. They must also adhere to SMP/E's requirements, as described in the *SMP/E Reference* manual and the *SMP/E User's Guide* that apply to your release of SMP/E or OS/390, and (for OS/390) the *OS/390 SMP/E Commands*.

- Packagers whose software will not be distributed through IBM Software Delivery.

These packagers can follow the rules and restrictions in this publication, but are not required to. However, they need to adhere to SMP/E's requirements, as described in the *SMP/E Reference* manual and the *SMP/E User's Guide* that apply to your release of SMP/E or OS/390, and (for OS/390) the *OS/390 SMP/E Commands*.

This publication is not intended for IBM packagers whose programs will be distributed through IBM Software Delivery. They should contact the owner of this publication for more information.

Before using this publication, you should be familiar with the *SMP/E User's Guide*, to acquire a general understanding of SMP/E and how to use it to install SYSMODs. You should also be familiar with the *SMP/E Reference* manual, which contains more detail about SMP/E syntax.

By following the rules, recommendations, and restrictions in this book, you can:

- Improve customer acceptance of your products
- Make it easier for customers to install and maintain your product on an IBM system
- Reduce the time and effort spent to analyze packaging problems, rework product packages, and test the associated deliverables
- Ensure that your product can be processed by IBM's common installation and distribution vehicles for MVS-based products

How this Book Is Organized

Table 1 on page xii describes the way this book is organized and what each chapter contains.

<i>Table 1. Organization of This Book</i>	
Chapter	Description
Chapter 1, "Introduction to MVS Product Processes"	Provides an introduction for developers who are relatively new to the MVS product processes, or who need only general information.
Chapter 2, "Assessing Your Product's Packaging Requirements and Considerations"	Lists questions that you can answer to help plan your packaging requirements.
Chapter 3, "Rules for Packaging Methods"	Describes the methods that can be used to package SYSMODs.
Chapter 4, "SYSMOD Types and Relationships"	Introduces you to SMP/E packaging concepts.
Chapter 5, "Fundamental Packaging Considerations"	Describes some fundamental topics for software packaging.
Chapter 6, "Elements and Load Modules"	Describes considerations for packaging the elements that make up a product.
Chapter 7, "Using MCS Statements to Define Products"	Describes how to use MCS statements to package a product.
Chapter 8, "Using MCS Statements to Manipulate Elements and Load Modules"	Describes how to perform basic operations on elements.
Chapter 9, "Using JCLIN"	Describes when to use JCLIN and how to use JCLIN.
Chapter 10, "Naming Conventions"	Describes the naming conventions for components, elements, libraries, and SYSMOD IDs.
Chapter 11, "Packaging for National Language Support (NLS)"	Provides information about NLS packaging considerations.
Chapter 12, "Packaging for Special Situations"	Provides information about some packaging areas that require special handling.
Chapter 13, "SYSMOD Packaging Examples"	Provides examples of SYSMOD packaging.
Appendix A, "Summary of Rules, Restrictions, and Recommendations"	Is a summary list of all the packaging rules, restrictions, and recommendations.
Appendix B, "MVS Service Packaging Rules"	Summarizes the rules for packaging service.
"Glossary"	Describes the terms used in this book.
"Bibliography and Classes"	Lists books and classes that provide additional useful information.

Conventions for Rules, Restrictions, and Recommendations

Rules, restrictions, and recommendations for the various packaging processes follow particular conventions.

For rules:

- The text of rules is enclosed in a box.
- "Must" is used instead of "should."

Following is an example of how a rule would be indicated:

Packaging Rule
<input type="checkbox"/> <i>n</i> . All elements must....

Restrictions are based on the limitations of SMP/E or IBM processes.

Recommendations indicate a preferred method of handling a situation; however, there may be other acceptable alternatives.

Restrictions and recommendations are indicated with text indicating the beginning and end of a small section of recommendations or restrictions; for example:

_____ IBM Software Delivery Solutions Restriction _____

The **IBM Software Delivery Solutions** process does not support ...

_____ End of IBM Software Delivery Solutions Restriction _____

_____ Packaging Recommendation _____

It is recommended that all products ...

_____ End of Packaging Recommendation _____

Rules and restrictions that have been deleted in previous versions of this book do not appear in the text of the book.

Also, note that *++element* and *++hfs_element* are the generic terms for the MCS that identify data elements and hierarchical file system (HFS) elements.

Summary of Changes

This section summarizes the changes made to this book.

Revision SC23-3695-09 (September 1999)

This edition was updated with the following changes:

- Information used in contacting IBM for component code registration was modified.
- The following rules were changed:
 - 2710 (return code from ACCEPT processing)
 - 3700 (packaging elements as members of a partitioned data set)
 - 3800 (Record Format for elements)
- Restrictions have been deleted from the following sections:
 - 3.1.1, “Format and Contents of the RELFILE Tape” on page 12:
 - RELFILEs must not be partitioned data sets extended (PDSEs).
- Recommendations have been added to the following sections:
 - 6.7, “Sample JCL and Data” on page 49
- Recommendations have been changed in the following sections:
 - 6.7, “Sample JCL and Data” on page 49

Revision SC23-3695-08 (March 1999)

This edition was updated with the following changes:

- The numbering scheme for rules has been changed to allow for future addition of rules while avoiding the use of decimal numbers wherever possible. Refer to Appendix C, “Mapping of Old Rule Numbers to New Rule Numbers” on page 237 for a mapping of the previous rule numbers to the new ones.
- Service Packaging Rule numbers were changed by prefacing the rule number with the letter “S”.
- The following rules were added:
 - Rule 2330 (negative prerequisite SYSMODs)
 - Rule 3410 (distribution libraries must be partitioned)
 - Rule 3510 (changing existing dataset attributes)
 - Rule 5820 (DDDEF jobs for products installing into the HFS)
 - Rule 5830 (specifying /etc with DDDEF entries)
 - Rule 14260 (using CALLLIBS)
 - Rule 14270 (handling return code 8 from APPLY processing)
 - Rule 15810 (++PROGRAM elements pre-bound with parts from another product)
 - Rule 18810 (symbolic links must not exist in the /tmp, /dev, /var or /etc directories)
 - Rule 18820 (installing directly into the /etc directory during APPLY processing)

-
- Rule 18830 (permission bits settings for files or directories in the HFS)
 - The following rules were changed:
 - Rule 800 (packaging sequential datasets as members of partitioned data sets)
 - Rule 2200 (common elements): This has been changed to a recommendation
 - Rule 13200 (DDNAME rules for JCLIN data)
 - The following rules were deleted:
 - Rule 2300 (was 18)
 - Rule 5100 (was 34)
 - Rule 15700 (was 149)
 - Rule 15800 (was 150)
 - The following restrictions have been added to the following sections
 - 3.1.2, “Creating the RELFILE Tape” on page 15:
 - Record format for RELFILES
 - ++MOD elements
 - 9.3, “General Packaging Rules for JCLIN Data” on page 89:
 - Requiring macro libraries during APPLY processing
 - Restrictions have been changed in the following sections:
 - 6.1, “General Packaging Rules, Restrictions, and Recommendations for Elements” on page 44 - ++MAC and ++SRC restrictions
 - Restrictions have been deleted from the following sections:
 - 4.1.1.3, “Choosing between Base and Dependent Functions” on page 22
 - 6.1, “General Packaging Rules, Restrictions, and Recommendations for Elements” on page 44
 - 6.6, “Shared Load Modules” on page 48
 - Recommendations have been added to the following sections:
 - 4.2.4, “Coexisting SYSMODs” on page 29:
 - 5.1, “Installation Methods” on page 33:
 - 5.4, “Record Length, Record Format, and Block Size Requirements” on page 36:
 - 6.7, “Sample JCL and Data” on page 49:
 - Table 13 on page 47 was added, which lists the MCS statements for hierarchical file system (HFS) elements.
 - Various editorial and technical corrections have been made.

Revision SC23-3695-07 (March 1998)

This edition was updated with the following changes:

- The following rule was deleted:
 - Rule 15410 (was 146) (Language-sensitive elements)
This rule was replaced by additional packaging recommendations.
- Various editorial and technical corrections have been made.

Revision SC23-3695-06 (September 1997)

This edition was updated with the following changes:

- This book now applies to OS/390 Version 2 (5647-A03)
- The following rules were changed:
 - Rule 1320 (was 14) (use of VOLSER on tapes)
 - Rule 9300 (was 72) (function specifying its own FMID)
 - Rule 100 (was 79) (conditional and unconditional relationships)
 - Rule S193 (PTF SYSMOD value)
- Various editorial and technical corrections have been made.

Revision SC23-3695-05 (March 1997)

This edition was updated with the following changes:

- The following rules were added:
 - Rule 5810 (was 39.1) (avoid invoking SMP/E in catalogued or instream procedures)
- The following rules were changed:
 - Rule 14230 (was 131.1) (INCLUDE statements)
 - Rule 14500 (was 134) (unique names for load modules)
 - Rule 14900 (was 137) (exceptions for library names)
 - Rule 15010 (was 140) (exceptions for library names)
- The following rules were deleted:
 - Rule 9400 (was 73) (specifying a SYSMOD for new release of base function)
 - Rule 5300 (was 36) (using ++DELETE on JCLIN for load)
 - Rule 12320 (was 108.1) (DELETE for elements) (The SMP/E that made this rule necessary was corrected by APAR IR32416.)
- Various editorial and technical corrections have been made.

Chapter 1. Introduction to MVS Product Processes

This chapter explains the following:

- What is meant by product packaging
- How packaging and processes for MVS products evolved
- How a program gets from development libraries onto the users' systems
- A simple example of a packaging program

1.1 What Is Product Packaging?

A program consists of elements such as modules, macros, and other types of data. Packaging is the science of building these elements into a deliverable product that can be installed and maintained on a computer system.

For MVS systems, *System Modification Program Extended (SMP/E)* is used to install a product, install changes (service, user modifications, new functions) to the product, and track the current status of each of the elements of the product. All products and service for MVS-installable products must be packaged so that they can be installed and maintained by SMP/E.

For SMP/E to install a product and service for that product, you must code *SMP/E modification control statements (MCS)* for the elements. MCS statements describe the elements of the product and any relationships the product has with other products that may also be installed on the same MVS system. The combination of elements and MCS statements is called a *system modification (SYSMOD)*.

Product packaging includes combining the appropriate MCS statements with the elements of a program to create one or more SYSMODs, then putting the SYSMODs in the proper format on a *relative file tape (RELFILE tape)*. This relative file tape is used to distribute the product to customers.

1.2 Evolution of Product Packaging

The way in which MVS systems have provided products has changed through the years. From the total system replacements of the early days, packaging has evolved to individual products, and to custom-built packages of multiple products.

- Individual Products – To make the software even more independent from the hardware and to allow a broader scope of independent software development, MVS release 3.8 restructured the software into many discrete functional areas identified by one or more function modification identifiers (FMIDs).
- SMP4 – MVS 3.8 included SMP Release 4 (SMP4), which supported function, PTF, APAR, and USERMOD SYSMODs. With SMP4, each functional area became separately installable and could be developed on asynchronous schedules (with proper considerations for dependencies on other functional areas). These functional areas are internally referred to as “products.”
- SMP/E – To further enhance SYSMOD processing, SMP/E was developed, which uses zones in a VSAM data set (the SMPCSI) to manage the system's target and distribution libraries. These zones can be defined by the user to manage the increasingly complex relationships between products.

To install an individual product, the customer uses SMP/E to install function SYSMODs, which contain the software, install logic, and JCLIN data for the product. For some products, the customer must also do a system, subsystem, or product generation to provide some job streams and SMP/E JCLIN data. SMP/E is also used to install preventive service and corrective service, with improved handling of exception SYSMODs.

- Custom-Built Product Packages – To further reduce the resources required to install products and service, IBM now provides custom-built packages of products as an alternative to individual products. Customers can order custom-built replacements, as well as custom-built updates for their systems or subsystems. Examples of these offerings include Custom-Built Installation Process Offering (CBIPO), Custom-Built Product Delivery Offering (CBPDO), and CustomPac.

1.3 Getting Products and Service from Development Libraries onto Users' Systems

To provide your users with products and service that can be processed by SMP/E, you must package the elements, the JCL used to install them (*JCLIN data*), and the associated MCSs into SYSMODs. The distribution medium can be a relative file tape or tape with inline data if the users will not have access to the data sets containing the elements and JCLIN data, or it can be the actual libraries containing the elements and JCLIN data if the users will have access to those data sets. (For a description of these methods, see Chapter 3, “Rules for Packaging Methods” on page 11.) The users will use the installation documentation you provide to install the SYSMODs onto their MVS systems.

1.3.1 Packaging and Distributing the SYSMODs

Figure 1 on page 3 is a summary of the steps you should follow to get the elements and JCLIN data from the development libraries into the SYSMOD format that will be used to distribute your product and service. Once the elements for the program have been coded, follow these steps:

- 1 Integrate the code:** Use the available tools and procedures to create a set of data sets that SMP/E can process.
 - a. Create any required JCLIN data.
 - b. Collect the elements and JCLIN data from the various development libraries.

Note: If you plan to distribute updates for macros or source later on, make sure they are initially shipped with sequence numbers in columns 73–80. Otherwise, SMP/E is not able to install the updates.
 - c. If you plan to distribute modules, compile and assemble the macros and source to create the object modules.
 - d. Link-edit the resulting object modules. The format of the link-edited modules must be the same as the format produced by the MVS/370, MVS/XA, or MVS/ESA linkage editor.

- 2 Build the SYSMOD package:** Use the available tools and procedures to create files and a relative file tape that SMP/E can process.
- a. Create a sequential data set that contains the MCS statements for the elements.
 - b. Create the relative files by unloading the integrated data sets. The format of the unloaded data sets must be the same as the format produced by the IEBCOPY utility.

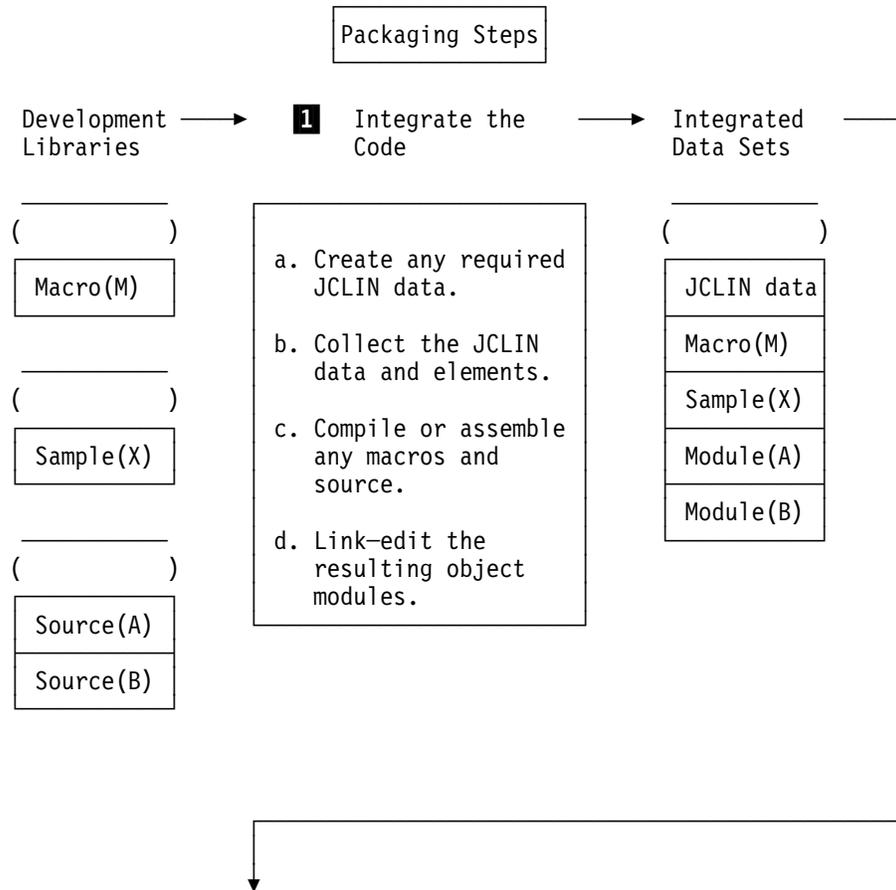


Figure 1 (Part 1 of 2). Getting Code from Development Libraries into SYSMOD Format

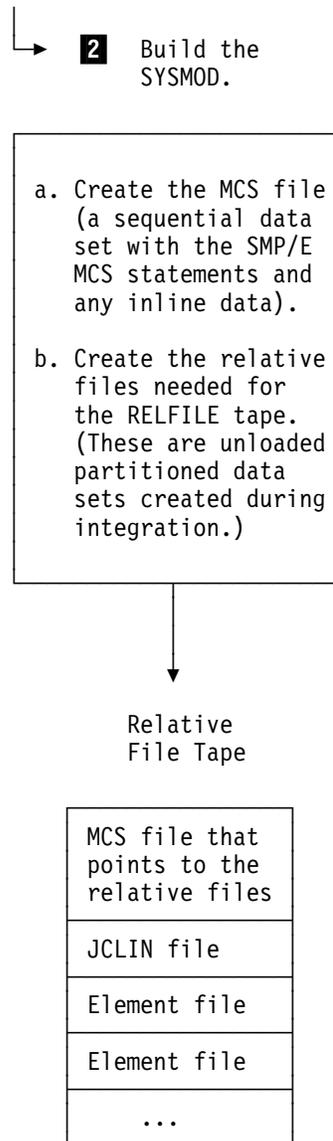


Figure 1 (Part 2 of 2). Getting Code from Development Libraries into SYSMOD Format

1.3.2 Writing the Installation Documentation

The installation documentation provided with a product should tell the users everything they need to know to install the product onto their systems.

Note: If you are packaging service, you should not have to provide any installation documentation. These procedures are described for users in the *SMP/E User's Guide*.

For product installation, you should include the following information:

- The product name, release, and SYSMOD IDs of the associated function SYSMODs.
- A description of the distribution medium.

Include the volume serial number, the file number, data set name, and contents of each file on the relative file (RELFILE) tapes.

- Requirements for hardware, software, and storage.

Also include a list of any requisites that cannot be defined in the MCSs.

For requisites defined in the MCSs, but which might be installed in a different zone from your product, instruct the users to run the REPORT CROSSZONES command to see if these requisites are installed.

- A list of all fixes that have been incorporated (if this is a new release of an existing product).
- A description of the installation method for the product.

These methods are described for users in the *SMP/E User's Guide*. Instead of writing detailed installation instructions, you may be able to select one of these methods and refer users to the *SMP/E User's Guide* for detailed steps.

However, you should provide any installation instructions that are unique to your product.

Subsequent chapters of this publication will help you provide this information.

1.3.3 Installing the SYSMODs

Figure 2 on page 6 is a summary of the steps users would follow to install SYSMODs onto their system using the standard SMP/E RECEIVE-APPLY-ACCEPT method. Once the users have access to the distribution medium, they should follow these steps:

- 1** Receive the SYSMODs. Use the SMP/E RECEIVE command to:
 - a. Store the SMP/E MCSs and any inline JCLIN data or inline elements in the SMPPTS data set.
 - b. Store the elements and JCLIN data in SMPTLIB data sets if the SYSMOD was distributed on a relative file tape,
- 2** Apply the SYSMODs. Use the SMP/E APPLY command to:
 - a. Process JCLIN data and save descriptions of the elements in the target zone.
 - b. Install the elements in the target libraries. The target libraries contain the executable code that constitutes the running system.
- 3** Accept the SYSMODs. Use the SMP/E ACCEPT command to:
 - a. Save descriptions of the elements in the distribution zone.
 - b. Install the elements in the distribution libraries. The distribution libraries (DLIBs) contain the master copy of each element for a system. They are used by SMP/E for backup when elements in the target libraries need to be replaced or updated.

What is Product Packaging?

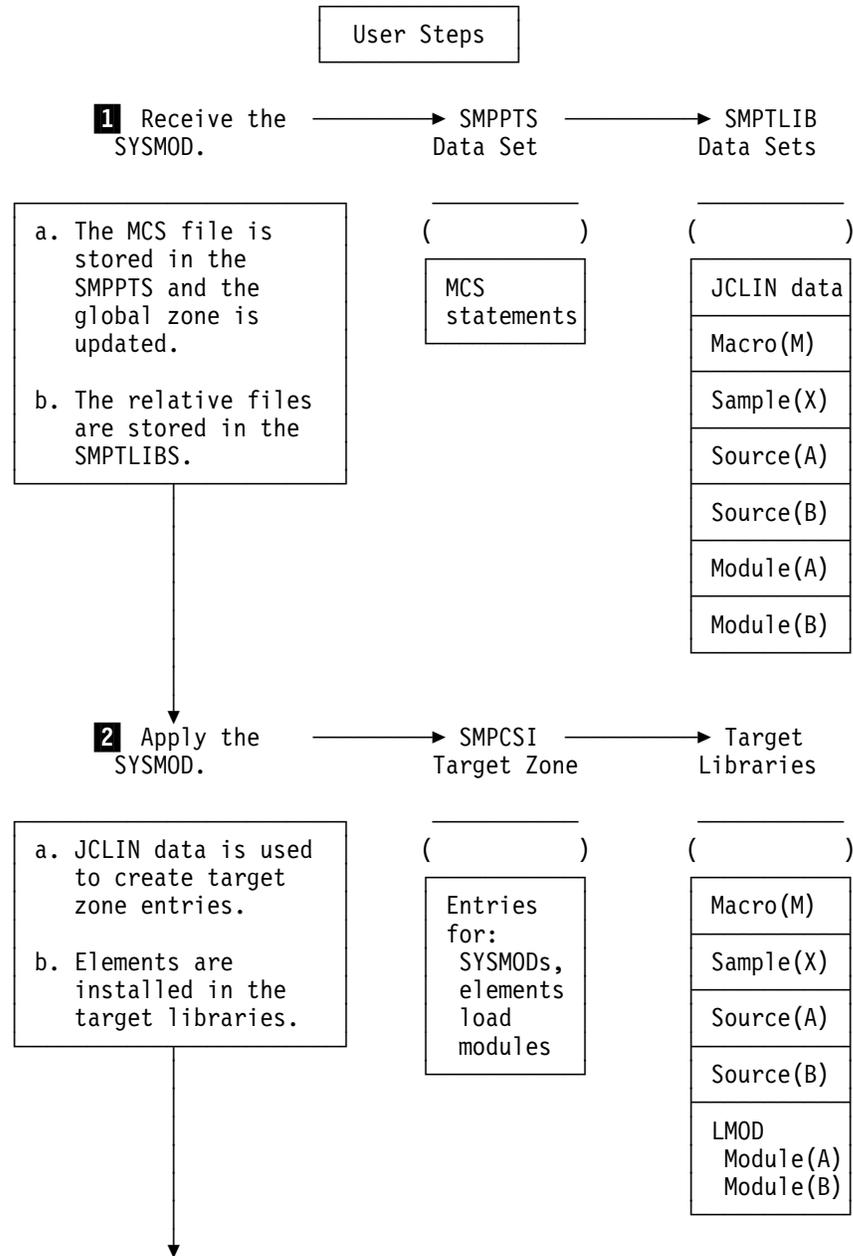


Figure 2 (Part 1 of 2). Getting Code from the SYSMOD onto the Users' Systems

What is Product Packaging?

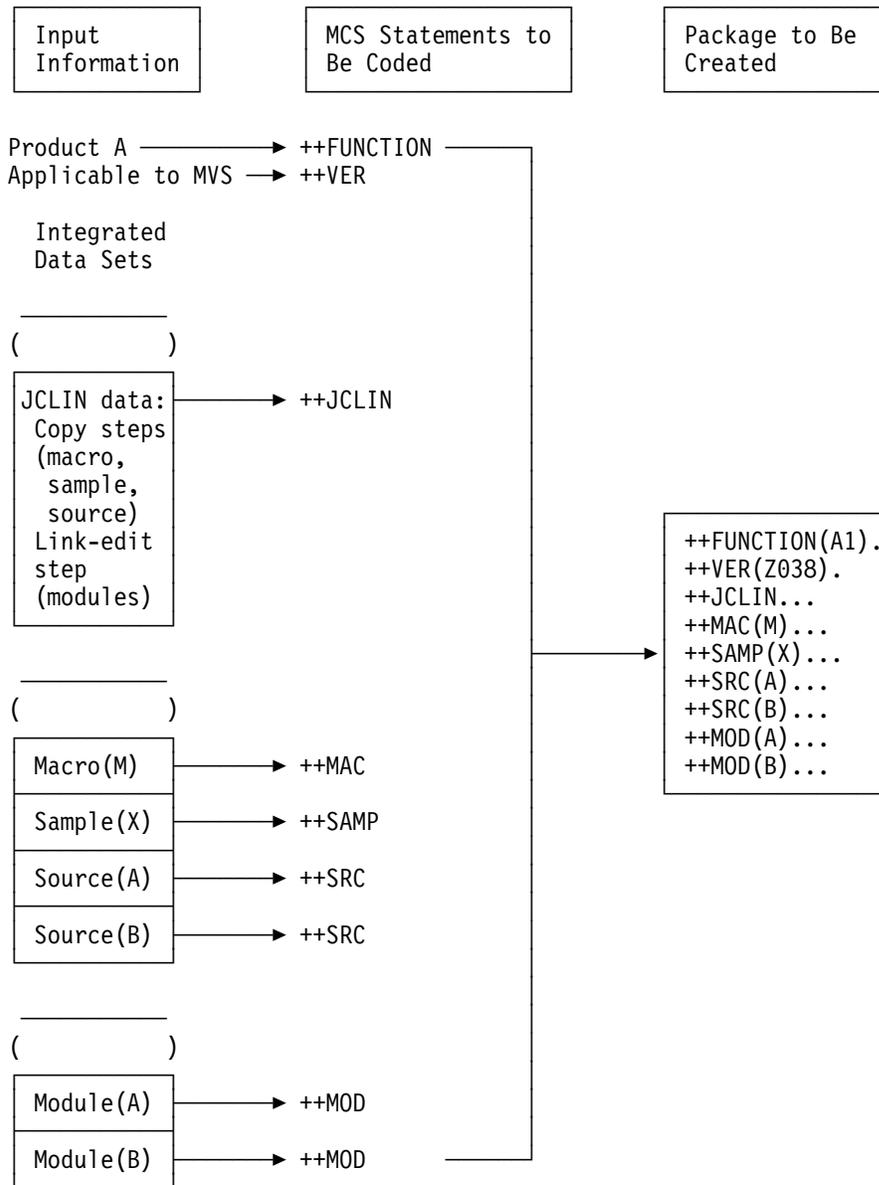


Figure 3. A Simple SYSMOD Packaging Example

Chapter 2. Assessing Your Product's Packaging Requirements and Considerations

The following questions help you determine:

- Your packaging requirements
- Areas you need to address
- Where to find the information you need

___ 1. Does the product use the standard SMP/E installation path (RECEIVE, APPLY, ACCEPT)?

If so, see 9.3, "General Packaging Rules for JCLIN Data" on page 89

___ 2. Is the product a base or dependent function?

See 4.1.1, "Functions" on page 21

___ 3. What SRELS does the product install in?

See 7.2.2, "Identifying the SREL" on page 57

___ 4. Do you require language support?

See the following:

- Chapter 11, "Packaging for National Language Support (NLS)" on page 127
- 13.4, "Example 3: Dependent Functions" on page 145

___ 5. Is this a new version, release, modification, or replacement?

See the following:

- 4.2.3.4, "Deleting and Superseding SYSMODs" on page 27
- 5.2, "Evaluating SYSMOD Relationships" on page 35

___ 6. Do you use high-level languages?

See 12.1, "High-Level Languages" on page 131

___ 7. Does your product use modules from another product?

See the following:

- 4.2.3.2, "Corequisite SYSMODs" on page 27
- 6.6, "Shared Load Modules" on page 48
- 8.5, "Enabling Load Module Changes at the CSECT Level (++MOD CSECT)" on page 84
- Chapter 12, "Packaging for Special Situations" on page 131
- 13.3.3, "Cross-Product Service between Corequisite Base Functions" on page 143
- 13.5.3, "Cross-Product Service for a Base Function with a Prerequisite" on page 155
- 13.8, "Changing the Contents of Products" on page 161

- ___ 8. Are there dependencies on other products, either within the same zone or residing in different zones?

See the following:

- 5.5, “Shared Libraries” on page 39
- 13.3.3, “Cross-Product Service between Corequisite Base Functions” on page 143
- 13.5.3, “Cross-Product Service for a Base Function with a Prerequisite” on page 155

Chapter 3. Rules for Packaging Methods

This chapter describes the rules to follow when putting SYSMODs on a tape or direct access data set to distribute the code to your users. This is the final packaging step after coding the MCSs needed for your SYSMODs. The following packaging methods are discussed:

- Relative file tapes
- Inline data
- Indirect libraries

A SYSMOD can be packaged using more than one method. Table 2 summarizes each of these methods and when each is used.

<i>Table 2. Comparison of SYSMOD Packaging Methods</i>		
Method	Description	When to Use
Relative file tapes	Elements and JCLIN data are in separate relative files from the MCSs.	<ul style="list-style-type: none"> • Allowed for anything except element updates • Used mostly for function SYSMODs
Inline data	Elements and JCLIN data immediately follow the associated MCSs.	<ul style="list-style-type: none"> • Only method allowed for element updates • Used for most IBM PTFs
Indirect libraries	Elements and JCLIN data are in partitioned data sets that are separate from the MCSs.	<ul style="list-style-type: none"> • Allowed for anything except element updates • Used when users have access to the integrated data sets containing the elements

IBM Software Delivery Restrictions

The **IBM Software Delivery Solutions** process supports only functions packaged in relative files that can be processed with SMP/E. Functions packaged in TXLIB or LKLIB data sets are not supported.

End of IBM Software Delivery Restrictions

3.1 Relative File Tapes

Packaging Rules (RELFILE Tapes)

- 110. A RELFILE tape can contain only files that can be installed by SMP/E and that meet the requirements for the format and contents of a RELFILE tape.
- 120. All files on an MVS-installable product tape must be SMP/E-installable.

A *relative file* tape, or *RELFILE* tape, is a standard label tape made up of two or more files. It contains a file of the MCS statements for one or more functions, and one or more relative files containing unloaded source data sets and unloaded, link-edited data sets containing executable modules. The relative files may also contain other data, such as sample procedures. These unloaded partitioned data sets must be in a format that can be installed on an MVS system or subsystem by SMP/E.

Note: You can create a RELFILE tape as either an actual tape or as a tape image. When you see references to “RELFILE tape,” they apply to tape images as well as to tapes.

Also, bear in mind that creating a RELFILE tape does not force users to receive the SYSMOD from tape. For example, a user may choose to load the RELFILE tape to DASD data sets, and then receive the SYSMODs from DASD. Or, a user may be sent a SYSMOD electronically, read the files into DASD data sets, and receive the SYSMOD from DASD.

The following sections discuss these topics:

- The format and contents of the RELFILE tape
- Creating the RELFILE tape

3.1.1 Format and Contents of the RELFILE Tape

Table 3 is an example of a RELFILE tape for two function SYSMODs: WLBZ100 (a base function) and XLBZ1B0 (a dependent function for the base function).

Table 3 (Page 1 of 2). Example of a RELFILE Tape		
File	Data Set Name	Contents
1	SMPMCS	<pre> ++FUNCTION(WLBZ100) REWORK(1997160) FILES(3) RFDSNPF(X) (PROD) /***** /* product copyright statement */ *****/ . ++VER(Z038). ++JCLIN RELFILE(1). ++MOD(A) DISTLIB(AGIMMODS) RELFILE(2). ++MOD(B) DISTLIB(AGIMMODS) RELFILE(2). ++MAC(X) DISTLIB(AGIMMACS) RELFILE(3). ++FUNCTION(XLBZ1B0) REWORK(1997160) FILES(3) RFDSNPF(X) (PROD) /***** /* product copyright statement */ *****/ . ++VER(Z038) FMID(WLBZ100). ++JCLIN RELFILE(1). ++MOD(A) DISTLIB(AGIMMODS) RELFILE(2). ++MOD(C) DISTLIB(AGIMMODS) RELFILE(2). ++MAC(Y) DISTLIB(AGIMMACS) RELFILE(3). </pre>
2	PROD.WLBZ100.F1	Unloaded partitioned data set containing member WLBZ100, which is JCLIN data for function WLBZ100
3	PROD.WLBZ100.F2	Unloaded partitioned data set containing modules A and B for function WLBZ100
4	PROD.WLBZ100.F3	Unloaded partitioned data set containing macro X for function WLBZ100

Table 3 (Page 2 of 2). Example of a RELFILE Tape

File	Data Set Name	Contents
5	PROD.XLBZ1B0.F1	Unloaded partitioned data set containing member XLBZ1B0, which is JCLIN data for function XLBZ1B0
6	PROD.XLBZ1B0.F2	Unloaded partitioned data set containing modules A and C for function XLBZ1B0
7	PROD.XLBZ1B0.F3	Unloaded partitioned data set containing macro Y for function XLBZ1B0

Packaging Rules (RELFILE Tape: Format and Contents)

- 300. The SMPMCS file must be a sequential data set consisting of 80-byte, fixed-length records.
- 400. All the other files on the tape or set of tapes must be relative files for the functions defined in the SMPMCS file.
- 500. All the elements for a function SYSMOD must be on the same logical tape as the SMPMCS file that defines the function.
- 600. There can be only one element with the same name in a given relative file. This includes element names and element alias names.
- 700. Each relative file must contain partitioned data sets that were unloaded in IEBCOPY format.
- 800. Sequential data sets must be packaged as members of a partitioned data set so that they can be unloaded by IEBCOPY into a relative file. A postinstallation job can be provided to copy such an element into a sequential data set. OS/390 Release 7 SMP/E or later can also be used to copy such an element into a sequential data set.
- 900. Modules must be in link-edited format. (This is RECFM=U, undefined record format.) The input parameters used for the link-edited format must include NCAL. Providing modules in link-edited format eliminates the need for the LEPARM operand and other data that is required on the ++MOD statement when modules are provided inline. Contrast with the restriction in 9.3, "General Packaging Rules for JCLIN Data" on page 89 regarding what to do for a PTF that introduces a new ++MOD requiring link-edit parameters other than the default.
- 1000. VSAM data set elements must be in AMS REPRO format.
- 1100. The partitioned data sets to be unloaded must have a member for each element MCS, plus a directory entry for each ALIAS associated with an element MCS. Likewise, each member in a RELFILE must be defined by an element MCS.
- 1300. If a member in a relative file contains JCLIN data for a SYSMOD, the member name must match the function's FMID.
- 1310. Follow the requirements in Table 11 on page 38 when specifying the MCS statements and data set attributes for elements being packaged in RELFILEs.

SMP/E assumes that modules on RELFILE tapes are link-edited and were unloaded in IEBCOPY format. SMP/E invokes the IEBCOPY utility, not the linkage editor, when copying LMODs. The IEBCOPY utility requires that all partitioned data sets have the same format.

IBM Software Delivery Restrictions

- Sequential data sets may not be used as target libraries. As long as this restriction exists, post-APPLY jobs may be provided to copy elements into a sequential data set. This is a restriction of the **IBM Software Delivery Solutions** process.
- RELFILES containing a RECFM=U must specify a BLKSIZE of 6144, so that they can be reblocked upwards at installation.
- ++MOD elements must not contain linkage editor ALIAS statements inline.

End of IBM Software Delivery Restrictions

Packaging Recommendations

To provide a consistent standard for customers so that they will not have to manually modify JCL, the files should be in this order:

- The SMPMCS file (only one)

Notes:

1. If the package consists of several tapes, there must be one SMPMCS file per logical RELFILE tape.
2. If the package consists of a base function and related dependent functions, the MCS statements for a base function must precede those for all its related dependent functions.

- The relative files

Note: If the tape contains more than one function, the relative files must be in this order:

1. The relative files for the first SYSMOD defined in the SMPMCS file. The order of each of these files must correspond to the value of the RELFILE operand specified on ++JCLIN and element statements.
2. The relative files for the second SYSMOD defined in the SMPMCS file, and so on.

End of Packaging Recommendations

Packaging Recommendations

Modules should be single-CSECT load modules.

End of Packaging Recommendations

3.1.2 Creating the RELFILE Tape

Any tool can be used to create the RELFILE tape, provided that the output has the required format and contents. The RELFILE tape that is created must follow certain rules for volume serial numbers and data set names.

Packaging Rules (RELFILE Tape: Volume Serial Numbers)

- 1320. If two tapes have the same volume serial number (VOLSER), they must contain the same FMIDs. It is permissible for different SUP levels of the same FMIDs to use the same VOLSER.

Packaging Rules (RELFILE Tape: Data Set Names)

- 1350. The data set name of each relative file must be *rfdsnpx.sysmod_id.Fnnnn*, where:
 - rfdsnpx*
is the prefix, if any, for the relative file data set names.

If a prefix is used in the data set names, that value must also be defined by the RFDSNPFX operand on the header MCS for the SYSMOD. RFDSNPFX tells SMP/E what prefix to use when allocating the data set names for the relative files being loaded.

If no prefix is used in the data set names, no RFDSNPFX value should be specified on the header MCS for the SYSMOD.

Note: Do not use "IBM" as the prefix.
 - sysmod_id*
is the FMID of the function to which the file is related.
 - Fnnnn**
is the letter F followed by the number specified on the RELFILE operand of the corresponding MCS statement in the SYSMOD. Do not use leading zeroes in the RELFILE number.

_____ Packaging Recommendations _____

The data set name of the first file should be *SMPMCS*.

_____ End of Packaging Recommendations _____

3.2 Inline Data

With inline data, the MCSs, JCLIN data, and elements are in a single package. The JCLIN data and elements immediately follow the associated MCS, instead of being packaged in a separate relative file or data set. The RELFILE, TXLIB, or LKLIB operand is not coded on the element or ++JCLIN statement.

Note: Data packaged inline must be in fixed-block 80 format. However, HFS elements and data elements are not always in fixed-block 80 format. If the original format of such an element is not fixed-block 80, you can use GIMDTS (a service routine provided with SMP/E) to transform the element so it can be packaged inline. Later, when the element is installed, SMP/E retransforms it to its original format. For more information about data elements, hierarchical file system (HFS) elements, and GIMDTS, see the *SMP/E Reference* manual.

With inline data, users will need extra storage for the SMPPTS and work data sets used to process the updates. When users receive a SYSMOD with inline data, SMP/E writes the entire SYSMOD to the SMPPTS. Later, when the SYSMOD is installed, SMP/E reads the element data from the SMPPTS and writes the data to the appropriate work data set before calling the utility programs to update the target or distribution libraries.

3.2.1 Example of Inline Element Updates

In Table 4, changes for macro USRMAC2 are packaged inline in a USERMOD.

<i>Table 4. Example of an Inline Element Update</i>	
DDNAME of Data Set	Contents
SMPMCS	<pre> ++USERMOD(USR0002). ++VER(Z038) FMID(QUSR001). ++MACUPD(USRMAC2) DISTLIB(AUSRMACS) SYSLIB(USRMACS) MALIAS(TERMINAL) ASSEM(USRASM01,USRASM02, USRSRC01,USRSRC02). ./ CHANGE NAME=USRMAC2 IEBUPDTE control cards and data go here ... </pre>

3.2.2 Example of Inline JCLIN Data

In Table 5 on page 17, the JCLIN data is packaged inline, and the elements are in relative files.

File	Data Set Name	Contents
1	SMPMCS	<pre> ++FUNCTION(PCC3100) FILES(2) RFDSNPFX(ABC). ++VER(Z038). ++JCLIN. //JOB JOB 'accounting info', // MSGLEVEL=(1,1) //STEP1 EXEC PGM=IEWL //SYSLMOD DD DSN=SYS1.LINKLIB,DISP=SHR //ACCDLIB DD DSN=SYS1.ACCDLIB,DISP=SHR //SYSLIN DD * INCLUDE ACCDLIB(A) ENTRY A ENTRY A1 ALIAS A1 NAME LA(R) INCLUDE ACCDLIB(B) ENTRY B ENTRY B1 ALIAS B1 NAME LB(R) /* ++MOD(A) DISTLIB(ACCDLIB) RELFILE(1). ++MOD(B) DISTLIB(ACCDLIB) RELFILE(1). ++MAC(X) DISTLIB(ACCMACS) RELFILE(2). </pre>
2	ABC.PCC3100.F1	Unloaded partitioned data set containing modules A and B for function PCC3100
3	ABC.PCC3100.F2	Unloaded partitioned data set containing macro X for function PCC3100

3.3 Indirect Libraries

With indirect libraries, elements and JCLIN data are in partitioned data sets instead of on a RELFILE tape, and are separate from the file containing the MCSs. These data sets (called *indirect libraries*) are pointed to by the MCSs for the elements or JCLIN data contained in those data sets. Indirect libraries may be text libraries or link libraries.

- Text libraries may contain JCLIN data, macros, source code, data elements, or modules that have not been link-edited. They are pointed to by the TXLIB operand on the ++JCLIN or element statement.

If JCLIN data is in a text library, the name of the member containing it must match the SYSMOD ID that the JCLIN is for.

- Link libraries contain link-edited modules. They are pointed to by the LKLIB operand on the ++MOD statement.

Indirect libraries can be used with a RELFILE tape or instead of a RELFILE tape. You might want to use indirect libraries:

- If users must supply the JCLIN data describing the function.

In this case, you could provide the elements on the RELFILE tape and point to the indirect library that will contain the JCLIN data.

- If users will have access to the integrated data sets containing the JCLIN data and elements.

In this case, there is no need to ship a tape. You could simply point to the integrated data sets containing the JCLIN data and elements.

This method has advantages over the inline data method because the data in the indirect libraries is not copied into the SMPPTS data set. Performance is improved because elements and JCLIN data do not have to be moved into work data sets when the SYSMOD is applied and accepted, and less space is needed for the SMPPTS.

3.3.1 Example of a RELFILE Tape with JCLIN Data in an Indirect Library

Instead of having a SYSMOD supply JCLIN data, you may need to have the user supply it. For example, if a function SYSMOD is installed using a generation procedure and it deletes the previous release of the function, users will have to save the Stage 1 generation output JCL and use it as JCLIN data when they apply the SYSMOD. In this case, you would use the TXLIB operand on the ++JCLIN statement to point to the data set in which the users must supply the JCLIN data. The macros and modules could still be in relative files. The following example shows a RELFILE tape where the JCLIN data will be provided by the users in a TXLIB data set.

<i>Table 6. Example of JCLIN Data in an Indirect Library</i>		
File	Data Set Name	Contents
1	SMPMCS	<pre> ++FUNCTION(PDD3200) FILES(2) RFDSNPF(ABC). ++VER(Z038) DELETE(PDD3100). ++JCLIN TXLIB(DDTXLIB1). ++MOD(A) DISTLIB(ADDDLIB) RELFILE(1). ++MOD(B) DISTLIB(ADDDLIB) RELFILE(1). ++MAC(X) DISTLIB(ADDMACS) RELFILE(2). </pre>
2	ABC.PDD3200.F1	Unloaded partitioned data set containing modules A and B for function PDD3200
3	ABC.PDD3200.F2	Unloaded partitioned data set containing macro X for function PDD3200

After receiving and accepting the function, the users will do a Stage 1 generation and save the output JCL in the data set pointed to by DDTXLIB1. When applying the function, the users must then provide a DD statement or DDDEF entry for that data set to process the JCLIN data that was saved. For example:

```
//DDTXLIB1 DD DSN=PDD3200.DDTXLIB1,DISP=SHR
```

Be sure to mention this requirement in the installation documentation for the function.

3.3.2 Example of Using Indirect Libraries Instead of a RELFILE Tape

A SYSMOD may be packaged in indirect libraries instead of on a RELFILE tape if users will have access to the integrated data sets containing the code. The following example shows a set of indirect libraries where JCLIN data is contained in a TXLIB data set, link-edited modules are contained in an LKLIB data set, and macros are contained in a TXLIB data set.

DDNAME of Data Set	Contents
SMPMCS	<pre> ++FUNCTION(PDD3200). ++VER(Z038) DELETE(PDD3100). ++JCLIN TXLIB(DDTXLIB1). ++MOD(A) LKLIB(DDLKLIB1). ++MOD(B) LKLIB(DDLKLIB1). ++MAC(X) TXLIB(DDTXLIB2).</pre>
DDTXLIB1	Partitioned data set containing JCLIN data for function PDD3200
DDLKLIB1	Partitioned data set containing modules A and B for function PDD3200
DDTXLIB2	Partitioned data set containing macro X for function PDD3200

When applying and accepting a SYSMOD packaged in indirect libraries, the users must provide DD statements or DDDEF entries for the TXLIB and LKLIB data sets so SMP/E knows where to find the code. Be sure to mention this requirement in the installation documentation for the SYSMOD. Here is an example of DD statements for the TXLIB and LKLIB data sets specified above.

```

//DDTXLIB1 DD DSN=PDD3200.DDTXLIB1,DISP=SHR
//DDLKLIB1 DD DSN=PDD3200.DDLKLIB1,DISP=SHR
//DDTXLIB2 DD DSN=PDD3200.DDTXLIB2,DISP=SHR
```

Chapter 4. SYSMOD Types and Relationships

A program is made up of elements such as macros, modules, or other types of data. For SMP/E to install and service software, you must code modification control statements (MCS statements) for the software elements. MCS statements describe the elements and any relationships the software has with other software that may also be installed on the same MVS system or subsystem. The combination of elements and MCS statements is called a system modification, or SYSMOD.

The following sections describe fundamental SMP/E concepts about SYSMODs and how they relate to product packaging.

4.1 Types of SYSMODs

Before coding any MCS statements for software changes, you must decide what type of SYSMOD to use. The SYSMOD type you choose depends on how the changes affect the system on which they are installed.

- A **function** introduces a new base or dependent function, or a new version or release (or both) of a function.
- A **program temporary fix (PTF)** corrects a problem that may affect all customers.
- An **APAR fix** corrects a problem that affects a specific user.
- A **user modification (USERMOD)** makes a change to an IBM product or to a user-written product.

Function SYSMODs for base functions define the environment for other SYSMODs that may be installed. All other SYSMODs are applicable to the base or dependent function that they support. For example, a PTF may fix an element that was introduced by a function SYSMOD for a particular base function—the PTF SYSMOD is therefore applicable to that function SYSMOD.

The following sections describe the various types of SYSMODs.

Note: Refer to Chapter 10, “Naming Conventions” on page 121 for information about the required naming conventions for SYSMODs.

4.1.1 Functions

Software products can be differentiated by the type of SYSMOD, or by their relationship to other functions. The two relationships are *base* and *dependent* functions. Each of these types of functions are packaged as *function SYSMODs*.

4.1.1.1 Base Functions

A *base function* is a collection of elements (such as source, macros, modules, and CLISTs) that provides a general user function and is packaged independently from other functions.

A base function is packaged as a function SYSMOD on a RELFILE tape, identified by an *FMID*. The FMID is described under 10.2, “SYSMOD IDs for Functions” on page 121. For more information on the implications of National Language Support

on base functions, see Chapter 11, “Packaging for National Language Support (NLS)” on page 127.

Function SYSMODs for base functions are applicable to any MVS environment, although they may have interface requirements that require the presence of other base functions.

4.1.1.2 Dependent Functions

A *dependent function* is a collection of elements (such as source, macros, modules, and CLISTs) that provides an enhancement to a base function. It may provide optional, additional function for a *base function*—this is called an “additive dependent function.” A dependent function may also provide language support for a base function or for an additive dependent function—this is called a “language-support dependent function.” A dependent function is identified with one, and only one, base function. On the other hand, there may be several dependent functions identified with the same base function.

A dependent function that provides language support may be for only one language. While one language may span multiple FMIDs, one FMID may not contain multiple languages. For more information, see Chapter 11, “Packaging for National Language Support (NLS)” on page 127.

A dependent function is packaged as a function SYSMOD on a RELFILE tape, identified by an FMID. FMIDs are described in 10.2, “SYSMOD IDs for Functions” on page 121.

Function SYSMODs for dependent functions are only applicable to the parent base function. Each dependent function specifies an FMID operand on the ++VER MCS to indicate the base function to which it is applicable. (The FMID is described in 10.2, “SYSMOD IDs for Functions” on page 121.) The hierarchy defined by this relationship determines the order in which the function SYSMODs must be installed: the base function specified on the FMID operand must be installed before or concurrently with the dependent function that contains that FMID operand. For more information, see 4.2.2, “Hierarchy of SYSMOD Types” on page 25.

4.1.1.3 Choosing between Base and Dependent Functions

Depending on the needs of your product and your customers, you may need to package your product as a combination of base and dependent functions. In addition, you may need to consider how to define these functions as features for your product, as well as the charges (and terms and conditions) that will apply to your product.

Remember these points when making your decision:

<i>Table 8 (Page 1 of 2). Comparison of Base and Dependent Functions</i>	
Base Function	Dependent Function
Is installable without its dependent functions. May require another base function.	Must be installed with its parent base function.
Can have no, one, or more than one dependent function	Must be associated with only one base function.

Table 8 (Page 2 of 2). Comparison of Base and Dependent Functions	
Base Function	Dependent Function
Can be explicitly deleted by another base function.	Can be explicitly deleted by another dependent function. Can be explicitly deleted by a base function.

4.1.1.4 General Packaging Rules for Functions

Packaging Rules (Functions)
<ul style="list-style-type: none"> <input type="checkbox"/> 2100. All elements included in an MVS-based product that is installed on an MVS system must be SMP/E-installable. <input type="checkbox"/> 2200. This rule has been changed to a recommendation (see below). <input type="checkbox"/> 2300. This rule has been deleted. <input type="checkbox"/> 2305. All unique elements for a given language must be packaged in a unique SYSMOD for that language.

Packaging Recommendations

Common elements should be packaged in a common SYSMOD.

End of Packaging Recommendations

Packaging common elements in a single SYSMOD makes it easier to service elements. When the common elements are packaged in the base function instead of in each language-support dependent function, you reduce the number of copies of that element that have to be updated when the element is serviced.

4.1.2 PTFs

A PTF SYSMOD provides *preventive service*, *corrective service*, or enhancements to a function.

Preventive service is service for a problem that a customer may not have yet encountered. CBPDO and ESO tapes provide preventive service PTFs to customers. By applying these PTFs, a customer may prevent problems from occurring on the system.

Corrective service is service explicitly requested by a customer to fix a problem that has occurred on the system.

A given PTF may be applicable to one or more releases of a function. Likewise, there may be more than one PTF for a given release of a function. Each PTF fixes one or more problems associated with the function.

Each PTF has a unique, 7-character name called a SYSMOD ID. The format of this SYSMOD ID is described under B.4, "Naming Conventions for Service" on page 236.

4.1.3 APAR Fixes

An APAR fix provides corrective service for a function. Corrective service is service explicitly requested by a customer to fix a problem that has occurred on the system.

An APAR fix is applicable to one, and only one, release of a function. Likewise, there may be more than one APAR fix for a given release of a function.

Each APAR fix has a unique, 7-character name. The format of this SYSMOD ID is described under B.4, “Naming Conventions for Service” on page 236.

4.1.4 USERMODs

You may want to provide a sample USERMOD with your product to let customers tailor your product to their needs. For example, you may want to include a USERMOD to help your customers change or add such things as:

- A procedure in PROCLIB
- A parameter or table in PARMLIB
- A sample job in SAMPLIB
- A user exit routine

By making your product tailorable through a USERMOD, you and the user benefit from SMP/E, which does the following:

- Keeps a record of the changes
- Reports any intersections with other SYSMODs
- Makes sure the changes are not regressed
- Makes sure the changes are installed properly in the correct libraries
- Allows the users to remove the changes, if necessary

Use the ++SAMP MCS statement to package the USERMOD as an element for the associated function. Define the element as being installed in an appropriate data set for sample code. Use the same 7-character name for both the element in which the USERMOD is packaged and the USERMOD SYSMOD ID. (See 10.3, “Element, Alias, and Load Module Names” on page 122 for more information about element naming conventions.)

The USERMOD can be installed by the customer as is, or it can be changed before it is installed.

For examples of packaging USERMODs, see the *SMP/E User's Guide*.

4.2 Defining SYSMOD Relationships

Understanding the relationships between SYSMODs is one of the most important aspects of planning how to package SYSMODs. A SYSMOD can only be installed properly and run correctly if its requirements regarding other SYSMODs on the system are met. For example, one SYSMOD may require the presence of another—a dependent function requires a particular base function. This section describes the types of SYSMOD relationships you may have to define in the course of developing and servicing a product. Specifically, it discusses the following:

- Conditional and unconditional SYSMOD relationships
- The hierarchy of SYSMOD types

- An overview of specific types of SYSMOD relationships
- Coexisting SYSMODs

4.2.1 Conditional and Unconditional Relationships

All SYSMOD relationships are either conditional or unconditional. Table 9 contrasts these two types of relationships.

<i>Table 9. Comparison of Conditional and Unconditional SYSMOD Relationships</i>	
Conditional Relationships	Unconditional Relationships
Specified on ++IF statements	Specified on ++VER statements
Enforced if the specified function SYSMOD is present	Always enforced

4.2.2 Hierarchy of SYSMOD Types

When defining SYSMOD relationships, take into account the hierarchy of SYSMOD types. SMP/E uses this hierarchy to determine which version of an element to install, if the element is contained in several SYSMODs. Figure 4 on page 26 shows this hierarchy, from the lowest functional level to the highest.

All of the SYSMODs in the hierarchy are part of the same product version. The product version includes all SYSMODs that have the same product version in the FMID specified on their ++FUNCTION or ++VER statements. (This convention is described under 10.2, “SYSMOD IDs for Functions” on page 121.) For example, SYSMODs with these statements would be in the same product version because they all have the same product version code (MX1):

```
++FUNCTION(WMX1200).
```

```
++FUNCTION(XMX1210).
++VER(Z038) FMID(WMX1200).
```

```
++PTF(UZ10000).
++VER(Z038) FMID(WMX1200).
```

```
++FUNCTION(WMX1300).
```

For SMP/E to process SYSMODs in the correct order, you must define that order to SMP/E. As Figure 4 on page 26 shows, if a dependent function has an element in common with its base function, the element in the dependent function is used instead of the one in the base—it is functionally higher.

You define SYSMOD hierarchy with operands on the ++VER statement.

- Base functions are the lowest level in the hierarchy. Therefore, they do not specify an FMID on the ++VER statement.
- Dependent functions specify the FMID of a base function on the ++VER statement. This base function must not be for a different product.
- PTFs and APAR fixes specify the FMID of a base or dependent function on the ++VER statement.

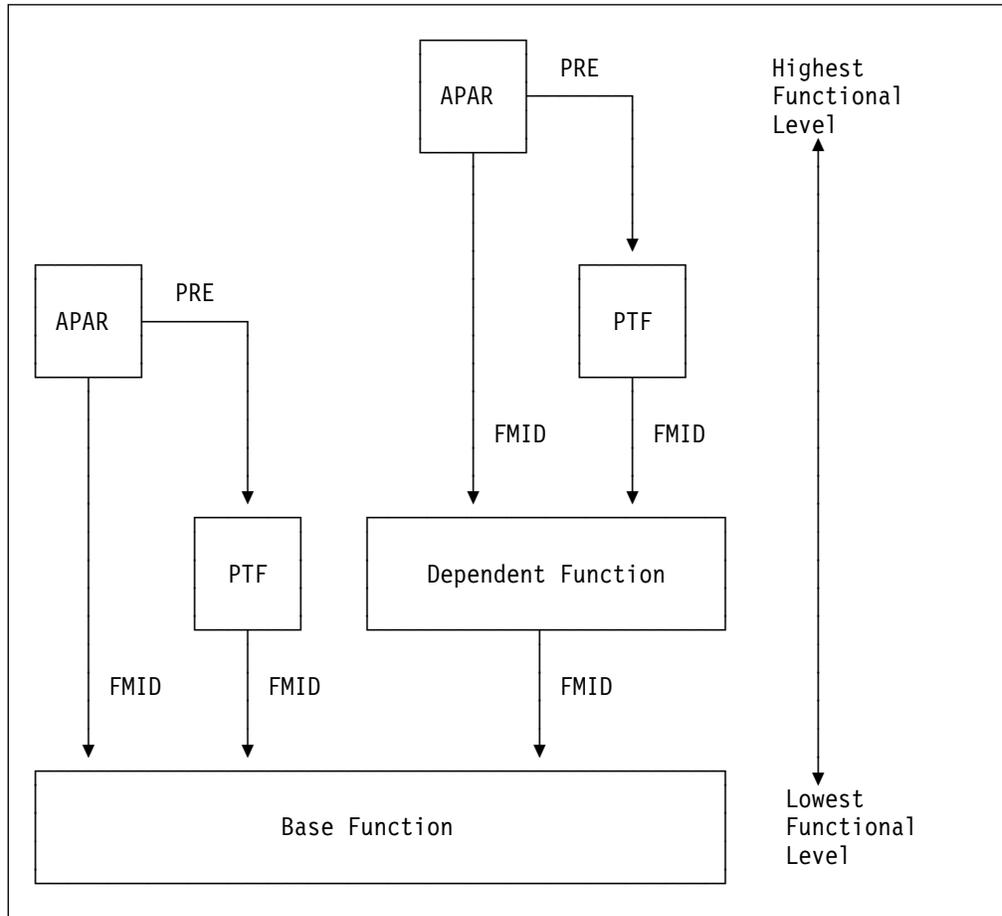


Figure 4. Hierarchy of SYSMOD Types

4.2.3 Specific SYSMOD Relationships

These are the types of relationships that may exist between SYSMODs:

- Prerequisite SYSMODs
- Corequisite SYSMODs
- Negative prerequisite SYSMODs
- Deleting and superseding SYSMODs

4.2.3.1 Prerequisite SYSMODs

Prerequisite SYSMODs have a relationship where one SYSMOD requires another.

If SYSMOD(2) needs SYSMOD(1) for proper operation, but SYSMOD(1) does not need SYSMOD(2), SYSMOD(1) is a prerequisite for SYSMOD(2). These are some cases when you would define a SYSMOD as a prerequisite:

- Defining the base function for a dependent function
- Defining the order of dependent functions
- Defining one product that is needed for another product
- Defining service for one product that is needed for another product

See 7.2.6, “Specifying Prerequisite Relationships (PRE)” on page 60 for information about how to specify this relationship, and 13.2.3, “PTF Service That Depends on Previous Service” on page 139 and 13.4.6, “Establishing the Order of Additional Dependent Functions” on page 152 for more examples.

4.2.3.2 Corequisite SYSMODs

Corequisite SYSMODs have a relationship where the two SYSMODs require each other.

If SYSMOD(2) and SYSMOD(1) need each other for proper operation, they are corequisites of each other. These are some cases when you would define SYSMODs as corequisites:

- Defining two products that need each other
- Defining two dependent functions for different products that need each other
- Defining related service for a dependent function and its parent base function

Packaging Recommendations

When you are building SYSMODs that may be installed as a group, such as pre-requisite or corequisite SYSMODs, do not construct the SYSMODs in such a way that their proper installation depends on the internal processing order within SMP/E. From time to time, the processing order may be changed and SYSMODs that depend on that order may not be installed correctly. Follow the packaging rules in this book to define how the SYSMODs should be installed.

End of Packaging Recommendations

4.2.3.3 Negative Prerequisite SYSMODs

Negative prerequisite SYSMODs have a relationship where one SYSMOD requires the absence of another.

If SYSMOD(2) can be installed only if SYSMOD(1) is not also on the system, SYSMOD(1) and SYSMOD(2) are negative prerequisites. For example, you might define dependent functions as negative prerequisites if they are mutually exclusive because they tailor a product to two different environments.

Note: All negative prerequisites are unconditional and may be specified only in function SYSMODs.

Packaging Rules (Negative Prerequisite SYSMODs)

- 2330. If two function SYSMODS cannot be installed in the same zone, the MCS of the function SYSMOD with the later availability date must have an NPRES for the other function SYSMOD. If both function SYSMODS have the same availability date, then the MCS for each one must have an NPRES for the other.

4.2.3.4 Deleting and Superseding SYSMODs

Deleting and superseding SYSMODs have a relationship where one SYSMOD replaces another.

- If other functions have a relationship with the function to be replaced, you should evaluate those relationships.
- If SYSMOD(2) takes the place of SYSMOD(1), it can delete SYSMOD(1), supersede SYSMOD(1), or both delete and supersede SYSMOD(1). For example, if Function A2 is a later release of Function A1, it can delete Function

A1, supersede it, or both. The differences between deleting a SYSMOD, superseding a SYSMOD, or doing both are shown in Table 10 on page 28.

See 7.2.7, “Superseding SYSMODS (SUP)” on page 61 for more information about deleting and superseding SYSMODs.

Packaging Recommendations

- A new release of a function should both delete and supersede the previous release if all of the following are true:
 - The new release contains at least all the function that was in the previous release.
 - If other products specified the deleted function as a requisite, all the internal and external interfaces used by those other products are unchanged in the new release.
 - Other products that specified the previous release as a requisite can run with the new release.
- Evaluate a replacement function using Table 10 as a guide. If the replacement function matches that description, then the preferred and recommended way to replace the previous function is to both delete and supersede it.

End of Packaging Recommendations

By both deleting and superseding the previous function, you gain the combined benefits of the DEL and SUP operands of the ++VER MCS.

- The DELETE operand is required in order to ensure that previous releases of the product are removed. This prevents accidental mixture of old and new elements in the same library.
- The SUP operand specifies that the new function completely and compatibly replaces all functions of the old function.
- The DELETE/SUP combination allows you to replace a function without disturbing any other SYSMODs that depend on that function. By specifying SUP, you are saying that the new function meets all dependencies identified by these dependent functions. You must ensure that this is the case before using the DELETE/SUP combination.

Table 10 (Page 1 of 2). Comparison of Deleting, Superseding, and Both Deleting and Superseding a SYSMOD

Delete	Supersede	Delete and Supersede
The new SYSMOD specifies DELETE on its ++VER statement.	The new SYSMOD specifies SUP on its ++VER statement.	The new SYSMOD specifies DELETE and SUP on its ++VER statement.
SMPCSI entries for the deleted SYSMOD are deleted. SMP/E no longer considers the deleted SYSMOD to be installed on the system.	SMPCSI entries for the superseded SYSMOD are saved. SMP/E considers the new SYSMOD to be a substitute for the superseded SYSMOD.	SMPCSI entries for the deleted and superseded SYSMOD are deleted. SMP/E considers the new SYSMOD to be a substitute for the deleted and superseded SYSMOD.

Table 10 (Page 2 of 2). Comparison of Deleting, Superseding, and Both Deleting and Superseding a SYSMOD		
Delete	Supersede	Delete and Supersede
Elements for the deleted SYSMOD are deleted from the target and distribution libraries.	Elements for the superseded SYSMOD are not deleted from the target and distribution libraries.	Elements for the deleted and superseded SYSMOD are deleted from the target and distribution libraries.
Note: The new SYSMOD may replace some elements at the same or higher functional level than the deleted, superseded, or deleted and superseded SYSMOD. The new SYSMOD may also add new elements.		

Using the previous example, if no other functions have a relationship with Function A1, Function A2 can delete Function A1. On the other hand, if some other function specified A1 as a requisite, A2 should *both* delete and supersede A1. This ensures that the requisite relationship is satisfied by both A1 and A2; no missing requisite prevents the other function from being installed.

Note: All deleting and superseding relationships are unconditional.

For specific examples of defining these relationships, see Chapter 13, “SYSMOD Packaging Examples” on page 137.

4.2.4 Coexisting SYSMODs

If two function SYSMODs can be installed in the same zone, they are said to “coexist.” Two function SYSMODs can coexist if they meet all these requirements:

- They apply to the same SREL.
- Neither SYSMOD deletes nor supersedes the other.
- Neither SYSMOD is a negative prerequisite of the other.
- If the SYSMODs are base functions, they are for different products.

Packaging Recommendations

Products should not require the customer to install them into their own unique zones; every product should be installable in the same target and distribution zones as any other product in the SREL. This gives the customer the ability to decide which combinations of products will reside together.

The installation information may suggest that the customer use new zones initially to avoid deleting previous releases of the product, but this should not be required.

Products should not require any function or service to be accepted before another function can be applied.

End of Packaging Recommendations

Although you cannot control the specific zone where a SYSMOD is installed, you can help users install the SYSMOD in the correct zone by packaging the SYSMOD correctly and by providing any additional information in the installation material. To provide this information, you must understand the rules for coexistence.

There are two ways for SYSMODs to coexist:

- **Unconditionally:** SYSMOD(A) is required by SYSMOD(B), and must be installed in a zone that contains SYSMOD(B). SYSMOD(A) “unconditionally coexists” with SYSMOD(B).
- **Conditionally:** SYSMOD(A) is not required by SYSMOD(B), and need not be installed in a zone that contains SYSMOD(B). SYSMOD(A) “conditionally coexists” with SYSMOD(B).

4.2.4.1 SYSMODs that Unconditionally Coexist

A requisite must be installed before (or concurrently with), and in the same zone as, the SYSMOD that specifies the requisite. The specifying SYSMOD cannot be installed without its requisite. Therefore, the requisite “unconditionally coexists” with the SYSMOD that specifies the requisite.

Reminder

“Unconditionally coexists with” means “must be installed before (or concurrently with) and in the same zone as.”

These are some examples of types of SYSMODs that “unconditionally coexist” with a dependent function:

- Its parent base function
- Any prerequisite dependent functions
- Any corequisite dependent functions.

4.2.4.2 SYSMODs that Conditionally Coexist

A SYSMOD that specifies a requisite is generally not required to be installed concurrently with, and in the same zone as, the SYSMOD it specifies as a requisite. Most requisites are one-way: SYSMOD(B) requires SYSMOD(A), but SYSMOD(A) does not require SYSMOD(B). The requisite (A) can be installed without the SYSMOD that needs it (B).

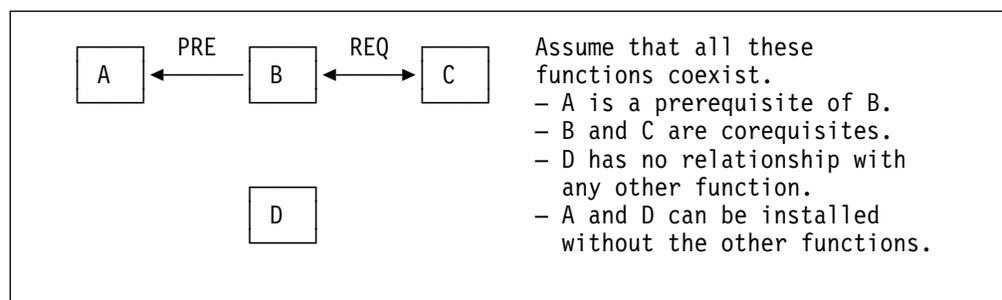
Likewise, any function SYSMODs that can coexist but do not require each other (or that do not even define any relationship to each other) are said to “conditionally coexist.” They can be installed either with or without each other.

Reminder

“Conditionally coexists with” means “can be installed with but is not needed for.”

4.2.4.3 Example: Conditional and Unconditional Coexistence

Look at the following example:



Based on these relationships:

- A unconditionally coexists with B and C.
- B and C unconditionally coexist with each other.
- B and C conditionally coexist with A.
- A, B, and C conditionally coexist with D.
- D conditionally coexists with A, B, and C.

Chapter 5. Fundamental Packaging Considerations

This chapter presents the following basic considerations for product packaging:

- Installation methods for function SYSMODs
- Evaluating SYSMOD relationships
- Adding FMIDs
- Record length, record format, and block size requirements
- Shared libraries
- Avoiding UCLIN

5.1 Installation Methods

A product is completely SMP/E-installable when it can be installed with both of these methods:

- RECEIVE–APPLY–ACCEPT
- RECEIVE–ACCEPT BYPASS(APPLYCHECK)–GENERATE.

The installation method using GENERATE reveals problems that may occur only when the entire system is generated together; these problems may be masked in the RECEIVE–APPLY–ACCEPT scenario. Also, GENERATE is more strict about product dependencies. For example, GENERATE does not allow DD statements to override DDDEFs; as a result, duplicate library names may be exposed when a system is generated and might not be visible when an individual product is installed.

Figure 5 on page 34 provides an overview of these methods. You should plan to test both of these installation paths to ensure that the results are identical. For details on all the steps in each method, see the *SMP/E User's Guide*.

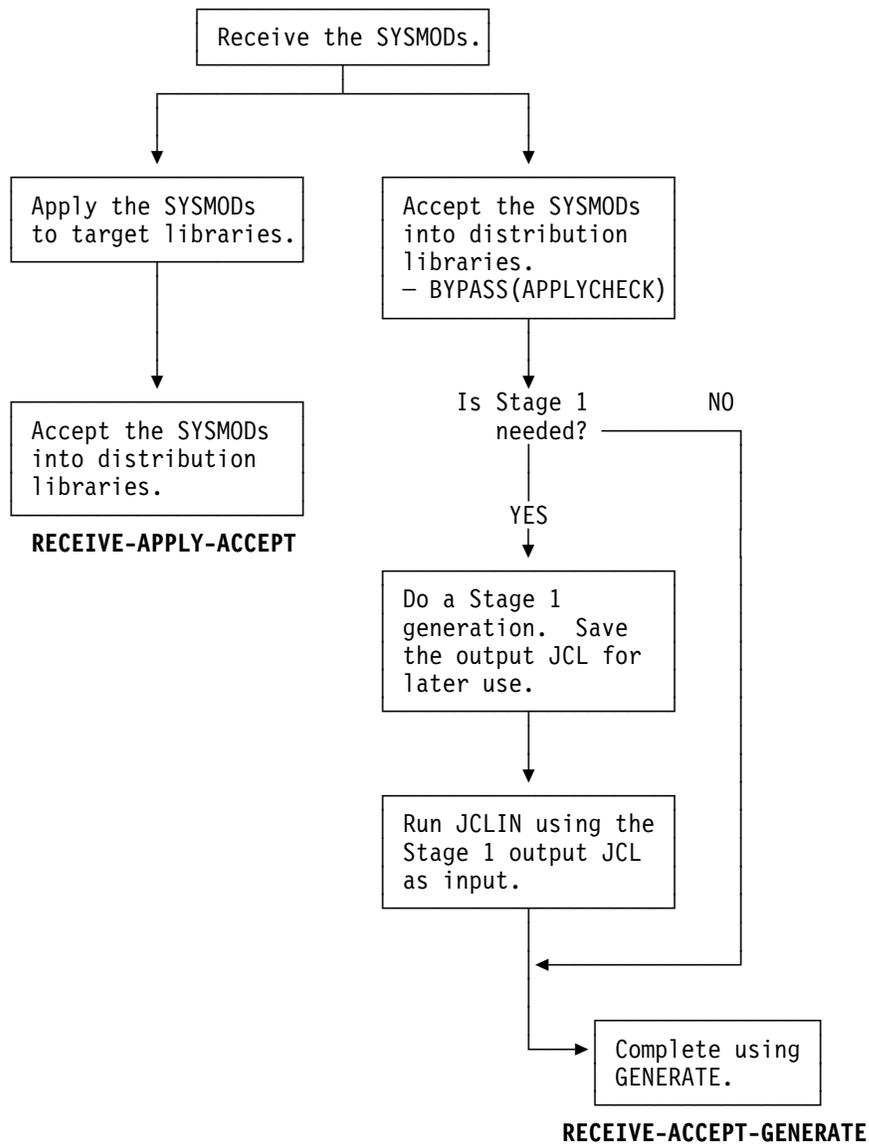


Figure 5. Overview of Methods for Installing Functions

Packaging Rules (Installation)	
<ul style="list-style-type: none"> □ 2600. All files on an MVS-installable product tape must be SMP/E-installable. □ 2700. All products must be packaged so that they can be individually installed using both the RECEIVE-APPLY-ACCEPT method and the RECEIVE-ACCEPT BYPASS(APPLYCHECK)-GENERATE method. 	

Packaging Rules (Installation)

- 2710. The return code from ACCEPT processing for all function SYSMODs must be zero, with these exceptions:
 - Warning message GIM39701W, SYSMOD *sysmod-id* HAS NO ELEMENTS.
 - Warning message GIM50050W, concerning the DESCRIPTION operand.
 - A warning message issued only in certain environments (for example, a product tries to delete an element or load module that is not on the system).

Packaging Recommendations

A PTF should not increase its product's driving system requirements beyond what is documented in the installation instructions.

End of Packaging Recommendations

5.2 Evaluating SYSMOD Relationships

At various stages in the life of a product you must consider the SYSMOD relationships you may have to define:

- **Packaging the initial release of a function**

The first time you package a product as a function, you must consider whether that function has any dependencies on other SYSMODs that might be installed on the same system.

Some SYSMODs have no relationships to any other SYSMODs. For example, you may have a simple function SYSMOD that stands on its own and has no requirements for any other functions to be installed. On the other hand, you may have a function that uses code provided by another function.

If a SYSMOD stands on its own, you do not have to define any relationships. However, if there are requirements for other SYSMODs, you must define these relationships.

Refer to 4.2, “Defining SYSMOD Relationships” on page 24 for more information.

- **Replacing a function**

After a function has been available for a while, you may develop enough changes to distribute a new release of that function. You must define the relationship of that new release to the previous release, as well as decide whether to carry over relationships defined in the previous release.

See 4.2.3.4, “Deleting and Superseding SYSMODs” on page 27 for more information.

- **Enhancing a function**

When you have changes for a function, you may want to update the function instead of replacing it.

For example, you may want to provide some optional capability for a particular environment. These enhancements could be packaged as a dependent function, which adds to the base function without replacing it. You must define the relationship between the base function and its dependent functions. If you develop several dependent functions, you must also define any relationships there may be among them.

Refer to 4.2.2, “Hierarchy of SYSMOD Types” on page 25 and 4.2, “Defining SYSMOD Relationships” on page 24 for more information.

- **Supporting language-sensitive elements**

When a product provides language-sensitive elements (such as messages and dialog elements), those language-sensitive elements should be packaged in a separate dependent function for each language, including U.S. English (even if U.S. English is the only language supported). The remaining elements remain in the base function. You must define the relationship between the base function and its dependent functions for language-sensitive elements.

Refer to Chapter 11, “Packaging for National Language Support (NLS)” on page 127 for more information.

- **Servicing a function**

At each of the stages in developing a function you may need to provide service to fix problems with the function. You must define the relationship between the service and the function. As you provide more service, you may also have to define relationships among service fixes.

Refer to 4.1.2, “PTFs” on page 23, 4.1.3, “APAR Fixes” on page 24, and to Appendix B, “MVS Service Packaging Rules” on page 209.

5.3 Adding FMIDs

New FMIDs are required for functions in new versions, releases, or modification levels. You need to be aware of the requirements for adding FMIDs.

Packaging Rules (++FUNCTION SYSMOD ID)

- | |
|---|
| <ul style="list-style-type: none">□ 2900. A new FMID is required for each new version, release, or modification level of an existing base or dependent function.□ 2950. A new FMID can be added only by a new version, release, or modification level. |
|---|

5.4 Record Length, Record Format, and Block Size Requirements

You should take these requirements into consideration as you develop your product; this will minimize problems when you build your RELFILE tape. Table 11 on page 38 summarizes the requirements for packaging elements in RELFILES.

Packaging Rules (Macros, Modules, Source)

- 3400. Macros, modules, and source elements must be members of a partitioned data set (DSORG=PO).
- 3410. Distribution libraries must be partitioned; only target libraries may be sequential. Use of sequential distribution libraries would tend to increase the total number of datasets required on the system.
- 3500. The record format (RECFM) for load modules must be U. For more information, see Table 11 on page 38.
- 3510. A product should not change any of the following attributes of an existing dataset:
 - RECFM
 - PDS vs. PDS/E
 - PATH attributes

If such a change is required, a new dataset must be created. It must have a new DDDEF entry as well as a new DDNAME and dataset name.

- 3650. The record format (RECFM) for macros and source must be FB, and the record length (LRECL) must be 80. For more information, see Table 11 on page 38.

Packaging Recommendations

- A dependent function should contain only those elements needed to provide the additive function, or that are needed to provide the additional language support. This reduces the number of element versions and makes servicing the elements easier.
- Sample job streams and other special data that might be helpful to the customer can be stored as a member of a partitioned data set that is unloaded to a relative file on the RELFILE tape. Examples include:
 - A procedure to allocate and catalog libraries
 - Installation verification procedures (IVPs)

This data should be packaged as sample code using the ++SAMP MCS statement, and it should be defined to be installed in an appropriate data set for sample code.

When SMP/E installs the SYSMOD, it copies this member into the libraries specified by the SYSLIB and DISTLIB operands on the element MCS statement. The sample job stream or other data can then be retrieved from the appropriate library for further processing.

- Jobs allocating target or distribution libraries must specify BLKSIZE=32760 for all RECFM=U datasets, and BLKSIZE=0 (utilizing system-determined block sizes) for all non-RECFM-U datasets, with the following exceptions:
 - SYS1.UADS
 - Font libraries

End of Packaging Recommendations

Packaging Rules (Data Elements, Hierarchical File System (HFS) and ++PROGRAM Elements)

- 3700. Data elements, hierarchical file system (HFS) elements, and ++PROGRAM elements must be packaged as members of a partitioned data set (DSORG=PO).
- 3800. The record format (RECFM) must be F, FA, FM, FB, FBA, FBM, V, VA, VM, VB, VBA, or VBM. The record format (RECFM) of ++PROGRAM elements must be U.

Notes:

1. Elements with fixed-length records are not restricted to a logical record length (LRECL) of 80.
 2. A VSAM data set may be a data element if it is in AMS REPRO format. However, after the data is installed by SMP/E, the customer will also have to run an AMS REPRO job to create the original form of the VSAM data. (SMP/E does not support native VSAM data sets as elements.)
- 3810. The maximum LRECL for a data element is 32,654.
 - 3900. Elements with variable-length records may not contain spanned records.
 - 3910. CLISTs must not have sequence numbers.
 - 4000. When packaging data elements, use the MCS statements shown in Table 12 on page 45. When packaging hierarchical file system (HFS) elements, use the ++hfs_element MCS statement.

Packaging Recommendations

- If a function SYSMOD uses unique target or distribution libraries, you may want to include a procedure to allocate and catalog the libraries. This procedure should be in an appropriate data set for sample code, must be a member in one of the relative files, and must be defined by the appropriate element MCS, as described above.
- A product may have an Installation Verification Procedure (IVP) that may be used by customers to verify that the product has been installed. If an IVP is included in the product package, it should be in an appropriate data set for sample code, must be a member in one of the relative files, and must be defined by the appropriate element MCS.

End of Packaging Recommendations

Table 11 (Page 1 of 2). Summary of Requirements for Packaging Elements in RELFILES

Element Type	MCS	RECFM	LRECL	Recommended BLKSIZE
Macro	++MAC	FB	80	8800 (See note 1.)

Table 11 (Page 2 of 2). Summary of Requirements for Packaging Elements in RELFILES

Element Type	MCS	RECFM	LRECL	Recommended BLKSIZE
Module (Each object module must be link-edited into a single-module load module.)	++MOD	U	No specific required value	6144 (See note 1.)
Source	++SRC	FB	80	8800 (See note 1.)
Data element (See note 2.)	++ <i>element</i>	No specific required value	No specific required value	For FB80, use BLKSIZE 8800. For other formats, BLKSIZE depends on DASD. (See note 1.)
HFS element (See note 2.)	++ <i>hfs_element</i>	No specific required value	No specific required value	For FB80, use BLKSIZE 8800. For other formats, BLKSIZE depends on DASD. (See note 1.)
<p>Notes:</p> <ol style="list-style-type: none"> 1. Use the most efficient block size for the DASD you support. The block size must not exceed that of the smallest DASD supported by your product, as indicated in the product's documentation. If the smallest DASD supported is the 3350, the block size must not exceed 19069. 2. Data elements and hierarchical file system (HFS) elements (unlike macros, modules, and source) have no required record format or logical record length. 				

5.5 Shared Libraries

This section discusses shared libraries. For information about library names, refer to 10.4, "Library Names" on page 124.

Packaging Rules (Shared Libraries)

- 4200. A library cannot contain two or more elements with the same name or alias name, even if they are different types. Therefore, if your product is to be installed in libraries shared with another product, you must ensure that none of your product's elements have the same name or alias name as those for elements of the other product that are installed in the same library.

Packaging Rules (Shared Libraries)

- 4300. If products share a library, each product must use the same data set attributes for that library. This means that if a product adds elements to an existing product-specific library, the new product must specify the same DCB attributes as the existing library.

If different products contain like-named elements (or aliases), data can be overlaid; this can produce unpredictable results.

If products share a library but specify different data set attributes, installation errors can occur.

If products share a library, the products must make sure that there will be sufficient space left in the library after installation. The library must be able to fit on all DASD types supported by all the products. Also, there must be sufficient space remaining so that the products can be serviced.

_____ Packaging Recommendations _____

To avoid problems with like-named elements or aliases, do not install your product in shared libraries.

_____ End of Packaging Recommendations _____

5.6 Avoiding UCLIN

UCLIN can cause many complications and must be avoided. Some potential problems resulting from UCLIN are:

- Increased chances for introducing errors
- Difficulty in debugging errors
- Performance impact for customized offerings

_____ Packaging Recommendations _____

Do not use UCLIN. Use MCS statements instead.

Note: UCLIN is acceptable, and recommended, to create or modify DDDEF entries.

_____ End of Packaging Recommendations _____

MCS statements can:

- Add modules to existing load modules
- Change ownership of an element
- Move macros, modules, source, and load modules
- Rename load modules
- Delete load modules
- Delete elements

Table 15 on page 73 describes some things you can do through MCS statements to avoid using UCLIN.

Refer to Chapter 8, “Using MCS Statements to Manipulate Elements and Load Modules” on page 73 for more information.

Chapter 6. Elements and Load Modules

The term “element” is used as a collective name for such things as:

- Source
- Macros
- Modules
- CLISTs
- Panels
- Procedures
- Sample programs that make up a product

Each element is distributed under a unique name (starting with the 3-character prefix for the product and referred to as the *element name*) and performs some particular function for the product that owns the element.

The element statements describe elements contained in a function. All elements of a product on the product tape must be described in its MCS. SMP/E provides a variety of MCS statements to accommodate a broad spectrum of element types, including language-sensitive versions of many element types.

- ++MAC describes a new or replacement macro.
- ++MOD describes a new or replacement module (a single-CSECT load module).
- ++SRC describes a new or replacement source module.
- Data element MCS statements describe new or replacement elements that are not macros, modules, or source. See 6.4, “Data Element Types” on page 45 for a list of the element data types.
- ++hfs_element MCS describes a new or replacement element that is installed in a hierarchical file system (HFS).

This chapter describes various considerations for packaging the elements that make up a product. These topics are discussed:

- Element ownership
- Element aliases
- Data element types
- Load modules
- Packaging sample JCL and data
- Language sensitive elements

6.1 General Packaging Rules, Restrictions, and Recommendations for Elements

This section describes general rules for packaging elements.

Packaging Rules (Elements)

- | |
|--|
| <ul style="list-style-type: none">□ 4910. An element can be owned by only one function. Ownership is defined by the FMID and VERSION operands on the ++VER and element statements. |
|--|

IBM Software Delivery Restrictions

- The TXLIB and LKLIB operands are not supported by the **IBM Software Delivery Solutions** process.
- The **IBM Software Delivery Solutions** process does not support SYSMODs requiring assemblies at ACCEPT time. Therefore, the following restrictions apply to the ++MAC and ++SRC statements:
 - A ++MAC statement must not specify the ASSEM, DISTMOD, DISTOBJ, or PREFIX operands unless the macro is accompanied by the assembled object modules and ++MOD statements.
 - A ++SRC statement must not specify the DISTMOD or DISTOBJ operand. In addition, the source can be accompanied by the assembled object module and ++MOD statement.
- A ++SRC statement must not specify the DISTMOD or DISTOBJ operand. In addition, the source can be accompanied by the assembled object module and ++MOD statement, unless the DISTLIB for the associated object module is SYSPUNCH. (To set up a DISTLIB or SYSPUNCH for the object module, JCLIN data must contain an assembly step for the source that assembles it into SYSPUNCH, and a link-edit step that includes the object module from SYSPUNCH.)

End of IBM Software Delivery Restrictions

Packaging Recommendations

Single-CSECT modules are recommended where possible. This makes it easier for the module to be serviced. A single CSECT can be distributed rather than shipping the entire module. SMP/E can perform a CSECT replacement.

End of Packaging Recommendations

For more details, see Chapter 8, “Using MCS Statements to Manipulate Elements and Load Modules” on page 73.

6.2 Element Ownership

An element must be exclusively owned by one product. For guidelines on moving an element from one product to another, see 13.8, “Changing the Contents of Products” on page 161.

You may have an element that is owned by one particular product and is being shared between your product, the owning product, and other products that have already been designed and delivered. In that case, for any new release of the owning product, your product (and each other sharing product) may need to provide either a PTF or a new release to ensure that all these products can still be installed.

6.3 Using Aliases for Elements

An *alias* is an alternative name assigned to an element or load module. It is important to maintain the uniqueness of these names to:

- Ensure that pieces are not unintentionally overlaid
- Make each entity identifiable to its owning product
- Allow each piece of a product to be serviced

For MVS products, an element is defined in the SMPMCS file using the `++element(ccccxxx)` statement, where `ccccxxx` is the name assigned to that particular element. Each piece that is shipped on a product tape must be defined in the SMPMCS file as either an element or an alias of an element. If an alias name is assigned to an element, the RELFILE tape must contain both the element and the alias in a RELFILE.

Refer to 10.3.3, “Alias Names” on page 123 for information about alias names.

6.4 Data Element Types

Table 12 lists the MCS statements that can be used to define data elements. It may not reflect the most currently supported values. For the latest information, see the *SMP/E Reference* manual.

<i>Table 12 (Page 1 of 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 18 on page 129.</i>	
MCS	Description
++BOOKxxx	Online book member
++BSINDxxx	Index for an online publications library (bookshelf)
++CGMxxx	Graphics source for an online book
++CLIST	CLIST
++DATA	Data not covered by other types
++DATA1–++DATA5	IBM generic data types 1–5 These are for IBM use only, to define elements that are not covered by any existing data types.

Table 12 (Page 2 of 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 18 on page 129.

MCS	Description
++DATA6xxx	IBM generic data type 6 This is for IBM use only to define an element not covered by any existing data types.
++EXEC	EXEC
++FONTxxx	Printer Font Object Contents Architecture (FOCA) font
++GDFxxx	GDF graphics panel
++HELPxxx	Help information (for example, a member in SYS1.HELP or a dialog help panel)
++IMGxxx	Graphics image for an online book
++MSGxxx	Message member (such as for a dialog or for a message data set)
++PARM	PARMLIB member
++PNLxxx	Panel for a dialog
++PROBJxxx	Printer object element
++PROC	Procedure in PROCLIB
++PRSRCxxx	Printer source element
++PSEGxxx	Graphics page segment for an online book
++PUBLBxxx	Online publications library (bookshelf)
++SAMPxxx	Sample data, program, or JCL in a data set for sample code
++SKLxxx	File skeleton for a dialog
++TBLxxx	Table for a dialog
++TEXTxxx	PDS member containing text plus SCRIPT tags
++USER1--++USER5	User-defined data types 1–5 These are for user-defined elements that are not covered by any existing data types.
++UTINxxx	General utility input
++UTOUTxxx	General utility output

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 xxx to indicate which language is supported by a given element. Refer to 11.1, “Element Types for Translated Elements” on page 128 for a description of national language identifiers. Figure 6 on page 47 shows an example where product XX1 (with a component code of ZZZ) must provide both English and French support for a message module, a panel, a panel message, and a sample element.

++FUNCTION(LXX1101).		++FUNCTION(LXX1102).
++VER(Z038) FMID(KXX1100).		++VER(Z038) FMID(KXX1100).
++MOD(ZZZMOD0E)... DISTLIB(AZZZMOD1).	message modules	++MOD(ZZZMOD0F)... DISTLIB(AZZZMOD1).
++PNLENU(ZZZPNL01)... DISTLIB(AZZZPNLE) SYSLIB(SZZZPNLE).	panels	++PNLFRA(ZZZPNL01)... DISTLIB(AZZZPNLF) SYSLIB(SZZZPNLF).
++MSGENU(ZZZMSG01)... DISTLIB(AZZZMSGE) SYSLIB(SZZZMSGE).	dialog messages	++MSGFRA(ZZZMSG01)... DISTLIB(AZZZMSGF) SYSLIB(SZZZMSGF).
++SAMPENU(ZZZSMP01)... DISTLIB(AZZZSAME) SYSLIB(SZZZSAME).	samples	++SAMPFRA(ZZZSMP01)... DISTLIB(AZZZSAMF) SYSLIB(SZZZSAMF).

Figure 6. 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 : GIM99XMP 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.)

6.5 Hierarchical File System (HFS) Element Types

Table 13 lists the MCS statements that can be used to define hierarchical file system (HFS) elements. It may not reflect the most currently supported values. For the latest information, see the *SMP/E Reference* manual.

Table 13 (Page 1 of 2). MCS Statements for Hierarchical File System 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 18 on page 129.

MCS	Description
++AIX1 through ++AIX5	Elements to be used by an AIX client.
++CLIENT1 through ++CLIENT5	Elements to be used by any client (intended for clients not described by other elements types).
++HFSxxx	Generic hierarchical file system element (data not covered by other types)
++SHELLSCR	UNIX shell script elements.

Table 13 (Page 2 of 2). MCS Statements for Hierarchical File System 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 18 on page 129.

MCS	Description
++OS21 through ++OS25	Elements to be used by an OS/2 client.
++UNIX1 through ++UNIX5	Elements to be used by a UNIX client.
++WIN1 through ++WIN5	Elements to be used by a Windows client.

Some types of hierarchical file system elements may have to be translated into several languages. In these cases, the corresponding MCSs contain xxx to indicate which language is supported by a given element. Refer to 11.1, "Element Types for Translated Elements" on page 128 for a description of national language identifiers.

6.6 Shared Load Modules

Table 13 on page 47 lists the MCS that can be used to define data elements.

Sometimes, products need to share load modules.

- A load module can contain multiple modules, some of which are owned by different FMIDs. This is called a *shared load module*. Examples of shared load modules are load modules that contain:
 - Compiler or high-level language (HLL) modules
 - Callable system services
 - Subsystem or product interfaces (for example, CICS, DB2, ISPF)
 - Modules from base and dependent functions or multiple dependent functions.
 - Modules from different products
- A module that can be link-edited into more than one load module or be dynamically accessed by more than one load module. This is called a *shared module*. Examples of shared modules are:
 - Compiler or high-level language (HLL) modules
 - Callable system services
 - Subsystem or product interfaces (such as CICS, DB2, and ISPF)
 - Modules being *reused* by more than one load module

Note: Refer to 10.3, "Element, Alias, and Load Module Names" on page 122 for more information about load module names and to 8.5, "Enabling Load Module Changes at the CSECT Level (++MOD CSECT)" on page 84 for information about using the ORDER statement for load modules.

Packaging Rules (Shared Load Modules)

- 5100. This rule has been deleted.
- 5200. One product must not use JCLIN to redefine the content of another product's load module (even for shared load modules). For more information, see 6.1, “General Packaging Rules, Restrictions, and Recommendations for Elements” on page 44.
- 5300. This rule has been deleted.
- 5400. Ensure that the product owner of a module that is shared across products does not use ++MOD DELETE and that it does not change the SYSLIB or DISTLIB of the shared module. For more information about deleting load modules, see 8.3, “Deleting Load Modules (++DELETE)” on page 81.
- 5500. If a module in Product A requires elements from Product B and the products are installed in different zones, the program directory for Product A must define Product B as a prerequisite.

Packaging Recommendations

- If a shared module is loadable and is used by more than one product, then products that share modules should dynamically load the shared modules during initialization and then link (or branch) to it as needed (there are performance considerations). This way, the latest level of the module is used without having to link-edit the module every time it is serviced.
- If a module is link-edited into a known existing load module and does not require link edit control statements (such as ENTRY, ALIAS, and ORDER), the ++MOD LMOD operand should be used.

End of Packaging Recommendations

For more information about sharing load modules, see 9.6.4, “Cross-Product Load Modules for Products Installed in the Same Zone” on page 107 and 9.6.5, “Cross-Product Load Modules for Products Installed in Different Zones” on page 109.

6.7 Sample JCL and Data

Sample JCL and data for a product may be stored as a member of a partitioned data set (PDS), then packaged in a relative file on a RELFILE tape for a function SYSMOD.

- To package sample JCL, use the ++SAMP statement.
- To package sample data, use either the ++SAMP statement or the ++DATA*n* statement.

For both sample JCL and sample data, make sure to specify an appropriate data set for sample code.

When SMP/E installs the function SYSMOD, it copies the elements into the libraries specified by the SYSLIB and DISTLIB operands on the MCS statement. The

sample JCL or data can then be retrieved from one of these libraries for further processing.

Packaging Rules (Samples)

- 5750. Sample installation JCL is optional, but if shipped, it must be packaged in a relative file.
- 5810. Products should not ship catalogued or instream procedures to invoke SMP/E during installation. Sample installation jobs should invoke SMP/E directly, and should require the installer to create DDDEF entries for all libraries.
- 5820. If a product installs into the hierarchical file system (HFS), its DDDEF job must include DDDEFs for the /usr/lpp/product_id/vendor_name/ directories it creates or uses. The DDDEF job must create the pathname in the DDDEF, and then provide a separate step to edit the DDDEF and change the path to the user-defined prefix. This is necessary to accommodate long pathnames that are not easily edited by hand.
- 5890. Products must not create DDDEF entries with /etc as a directory in the pathname.

Packaging Recommendations

- Products should supply sample jobs to perform the SMP/E APPLY and ACCEPT functions.
- Products should ship a sample job to allocate any target or distribution libraries that are created by the product, and should require the installer to run it. If any of these libraries are shared libraries that may have been allocated by other products, such libraries should be allocated in a separate job or job step, with instructions to the user explaining when the job or job step should be run.
- Products should ship a sample job to create DDDEF entries for new target and distribution libraries, as well as any existing libraries that may not have entries in this product's target or distribution zone. Entries for all distribution libraries should be created in the distribution zone, and entries for all target libraries should be created in the target zone. In addition, entries for all distribution libraries should be created in the target zone to support RESTORE processing.
- The installation instructions should identify the names of the sample jobs and in which RELFILE they reside, so that customers can download the jobs directly from the tape, if desired. The installation instructions should also state that the customer can perform an SMP/E RECEIVE to load the jobs into temporary libraries, copy them into private data sets, and then modify and run the jobs from these data sets.
- If Function A requires Function B with the FMID, PRE, or REQ operands, and Function A uses Function B's libraries, then Function A is not required to ship allocation or DDDEF jobs for any libraries allocated by Function A.
- Sample jobs should include clear and detailed comments. Information necessary to update the job prior to submission should be in the job, not in the installation instructions for the product.

- If a sample job is provided on the tape, the text of the job should not appear in the installation instructions. This will reduce the size of the installation instructions, as well as avoid situations where the tape and the installation instructions do not match.
- DDDEF jobs should adhere to the following:
 1. Use ADD DDDEF, not REP DDDEF.
 2. Use the WAITFORDSN operand.
 3. Use separate job steps to divide datasets into logical groups. For example, a product could use one step for new datasets, and other steps for datasets introduced in previous releases.
- Middle-level qualifiers of VxRxMx should not be specified in sample allocation jobs.
- Symbolic links for hierarchical file system (HFS) files should be created in the MKDIR EXEC, and should be relative, not absolute. To ensure that the MKDIR EXEC can run multiple times without damage, products creating symbolic links in the MKDIR EXEC should also provide UNLINK statements for every symbolic link ever created in this or previous levels, including those that have become obsolete.
- The MKDIR EXEC should be called zzzMKDIR, and the jobname of the JCL invoking it should be called zzzISMKD, where zzz is the three-character prefix of the product shipping the elements.
- A PTF should not add or delete DDDEF entries, or change dataset or pathnames in a DDDEF entry. If this is unavoidable, the following is recommended:
 1. A ++HOLD ACTION should be placed on the PTF.
 2. The changes should be shipped in a separate DDDEF or MKDIR EXEC shipped in the PTF, not by updating and reshipping the existing DDDEF or MKDIR EXEC.
 3. The new DDDEF or MKDIR EXEC should appear in the HOLDDATA of the PTF
- Sample allocation jobs should specify UNIT=SYSALLDA for all target and distribution libraries.
- Sample DDDEF creation jobs should specify UNIT(SYSALLDA) for all target and distribution libraries.
- All products installing into the hierarchical file system (HFS) should statically create their directories in a MKDIR EXEC. The installation documentation should document how to run the EXEC during the installation of the product, similar to the documentation on running dataset allocation jobs.
- The MKDIR EXEC should meet the following requirements:
 1. It accepts a parameter for the highest-level directory, rather than hard-coding it.
 2. Output is sent to the SYSOUT held queue. It contains a report of what was created, what was not created, and what directories already existed. It also includes the return code received and the date and time it was run.
 3. The directory names all appear together.

4. It can be executed multiple times successfully.
5. If a product provides EXECs run during installation, such as MKDIR EXECs, a batch job invoking the EXEC should be provided for the customer's use. This does not apply to EXECs run after installation, such as IVPs or customization.

- Products should not use the SMP/E dynamic allocation function to allocate target and distribution libraries as new data sets; the usage of DDDEFs is recommended only after the datasets have been allocated outside of SMP/E.

_____ End of Packaging Recommendations _____

For more information about packaging a RELFILE tape, see 3.1, "Relative File Tapes" on page 11.

6.8 Language-Sensitive Elements

A product may have elements that require translation for national language support (NLS). In this case, you must use a base function or additive dependent function for the elements that do not have to be translated, and a separate language-support dependent function for each language into which elements are translated.

For more information, refer to Chapter 11, "Packaging for National Language Support (NLS)" on page 127.

Chapter 7. Using MCS Statements to Define Products

SMP/E MCS statements are used to define products as function SYSMODs. This is the order of MCS statements for a function SYSMOD:

MCS Type	How Many
++FUNCTION	(one)
++VER	(one or more)
++IF	(none, one, or more)
++HOLD	(none, one, or more)
++MOVE	(none, one, or more)
++RENAME	(none, one, or more)
++DELETE	(none, one, or more)
++JCLIN	(none or one)
++element	(one or more to replace or update elements)

This chapter discusses the rules and MCS considerations you must follow when specifying the following statements:

- ++FUNCTION
- ++VER
- ++IF
- ++HOLD
- ++element

For more information about the ++MOVE, ++RENAME, and ++DELETE statements, see Chapter 8, “Using MCS Statements to Manipulate Elements and Load Modules” on page 73. For more information about the ++JCLIN statement, see Chapter 9, “Using JCLIN” on page 87. For details on all MCS statements, see the *SMP/E Reference* manual.

Notes:

1. Not every statement is fully documented here. The emphasis here is on those statements and operands used for packaging.
2. All input to SMP/E must be in uppercase (except comments, the LINK value on the ++hfs_element MCS, the ALIAS value on the ++DELETE statement, and alias values in link-edit JCLIN). For details on SMP/E's syntax rules, see the *SMP/E Reference* manual.
3. All references to releases or modification levels of a function also apply to versions—each version consists of at least one release.

IBM Software Delivery Restrictions

The ++ ASSIGN statement is not described in this section and must not be used.

End of IBM Software Delivery Restrictions

7.1 ++FUNCTION Statement

The ++FUNCTION statement identifies the SYSMOD as a base function or dependent function. A function SYSMOD may include only one ++FUNCTION statement.

Operands on the ++FUNCTION statement are used to:

- Specify the SYSMOD ID
- Identify the REWORK date
- Specify the prefix used for relative file data set names
- Specify the copyright information

7.1.1 Specifying the SYSMOD ID (*sysmod_id*)

The *sysmod_id* operand is the name, or SYSMOD ID, of the function. The SYSMOD ID is required, and only one value can be specified. It is also called the function modification identifier (FMID). See 10.2, “SYSMOD IDs for Functions” on page 121 for more information about the naming convention for FMIDs.

7.1.2 Identifying the REWORK Date (REWORK)

The REWORK operand indicates the date that a function was first released or last updated. The REWORK operand is important, because it can be used to distinguish every time a given function is updated and reissued with the same FMID. The date the work was done is specified as *yyyyddd*, which is the year followed by the Julian date (for example, 1993110).

Packaging Rules (++FUNCTION REWORK)
<ul style="list-style-type: none">□ 5900. REWORK is required on all ++FUNCTION statements, including the initial release. <p>Specify the REWORK date as <i>yyyyddd</i>, which is the year followed by the Julian date (for example, 1995110).</p> <p>You must change the date every time the function is updated and reissued with the same FMID.</p>

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.)

7.1.3 Specifying the Prefix for RELFILE Data Sets (RFDSNPFX)

The RFDSNPFX operand identifies to SMP/E the prefix that was 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. When you specify a value on the RFDSNPFX operand, remember to use that value in the names of your RELFILE data sets. For more information, see 3.1.1, “Format and Contents of the RELFILE Tape” on page 12.

Packaging Rules (Prefix for RELFILE Data Sets)

- 5910. If you specify a value for RFDSNPFX, do not specify RFDSNPFX(IBM). Use a different prefix.

7.1.4 Specifying Copyright Information

A copyright comment should be used for all products. Make sure to check on the legal requirements for your company.

Packaging Recommendations

The copyright statement should be within the first 15000 bytes of object code on the distribution media.

End of Packaging Recommendations

Here is an example of the placement of the copyright statement for a licensed program:

```
++FUNCTION(sysmod_id) FILES(nn) REWORK(yyyyddd)
/*****/
/* --- copyright statement goes here --- */
/*****/.
```

Additional comments may be included as separate records after the copyright statement and before the final delimiter (*).

Notes:

1. The comment statement begins in column 2.
2. If an FMID is included in more than one product, the copyright statement can refer to all the products in which that FMID is included.

7.2 ++VER Statement

The ++VER statement describes the environment required for this SYSMOD. A SYSMOD must contain a separate ++VER statement for each environment to which it applies.

The ++VER statement is required for all SYSMODs.

The operands on the ++VER statement are used to:

- Identify the SREL
- Delete SYSMODs
- Identify the base function to which a dependent function applies
- Specify mutually exclusive relationships
- Specify prerequisite relationships
- Specify requisite relationships
- Specify SYSMODs that are superseded by another SYSMOD
- Define ownership of SYSMODs

7.2.1 General Packaging Rules (++VER)

Packaging Rules (++VER)
<ul style="list-style-type: none">□ 6200. Every SYSMOD referenced on a single ++VER statement must reside in the same zone.□ 6205. When two or more SYSMODs affect the same element, you must specify the relationship among those SYSMODs. Specifically, you must define the order in which they should be processed (indicated by the PRE, SUP, or FMID operand) and the correct version of the element to be installed (indicated by the FMID or VERSION operand).

Packaging Rules (Multiple SYSMODs Affecting an Element)
<ul style="list-style-type: none">□ 6210. When two or more SYSMODs affect the same element, you must specify the relationship among those SYSMODs.<ul style="list-style-type: none">– If both SYSMOD A and SYSMOD B ship the element (or updates to it), the MCS statements in both SYSMODs must define the order in which the SYSMODs should be processed (indicated by the PRE, SUP, or FMID operand) and the correct version of the element to be installed (indicated by the FMID or VERSION operand).– If Product A includes an element from Product B via an INCLUDE statement in a JCLIN link-edit step without changing the element, and Product A requires a particular level of Product B, then Product A's MCS statements must specify an unconditional requisite for the appropriate level of Product B.– If Product A includes an element from Product B via an INCLUDE statement in a JCLIN link-edit step without changing the element, and multiple levels of Product B would fill the needs of Product A, then Product A's program directory must identify Product B as an installation requirement, specifying the lowest acceptable level of Product B.

7.2.2 Identifying the SREL

Packaging Rules (++VER SREL)

- 6300. On a single ++VER statement, all SYSMODs specified on the NPRES, PRE, REQ, SUP, and VERSION operands must be applicable to the same SREL as the SYSMOD containing this ++VER statement.
- 6400. You must use one of the following SRELs: Z038 for MVS, C150 for CICS, P004 for NCP, or P115 for IMS and DB2.

7.2.3 Identifying a SYSMOD's Base Function (FMID)

The FMID operand identifies the base function to which this SYSMOD applies.

Packaging Rules (++VER FMID)

- 6500. The FMID operand can be used only in a dependent function, not in a base function. The FMID specified in the operand must be the FMID of a base function. Both functions must be applicable to the same SREL. FMID is required for dependent functions.
- 6600. A SYSMOD cannot be both a base function and a dependent function. The FMID operand identifies a SYSMOD as a dependent function; therefore, if you specify the FMID operand, you must include it on all the ++VER statements for the SYSMOD.

7.2.4 Deleting SYSMODs (DELETE)

The DELETE operand indicates which function SYSMODs should be deleted when this function is installed. Using the DELETE operand for deleting SYSMODs is shown in Chapter 13, "SYSMOD Packaging Examples," on pages 141, 144, and 152.

DELETE is a multiple entry operand that specifies the functions to be deleted, such as a previous release of a base or dependent function.

Note: Generally, any function specified must be part of the same product. However, a new release of a product may need to delete older, equivalent releases of a different product that is applicable to the same SREL. For example, a new release of Product B might include function that was previously in Product A. In this case, Product B would need to delete all previous releases of itself and Product A. In such cases, the owner of Product B would have to negotiate with the owner of Product A for ownership approval.

The deleted function may have had requisites, JCLIN data, or other SYSMOD relationships information that must be considered when you package the deleting function. These considerations are the same as those for superseding SYSMODs, as shown in Table 14 on page 63. Table 10 on page 28 also provides more information about deleting and superseding SYSMODs.

If the specified SYSMODs are installed, SMP/E deletes them from the target and distribution libraries and from the SMP/E data sets. These SYSMODs are *explicitly*

deleted. (SMP/E does not delete ++IF REQ data for SYSMODs that are explicitly deleted.

SMP/E also deletes any SYSMODs (such as PTFs) that depend on the specified SYSMODs (that is, any SYSMODs that name the specified SYSMODs on the FMID operand of their ++VER statements). These SYSMODs are *implicitly deleted*. (SMP/E does not delete ++IF REQ data for SYSMODs that are implicitly deleted.

Notes:

1. If a function requires any service that was previously installed on a deleted function, the user may have to reinstall that service. (This may be the case when a PTF applies to more than one release of a function.) When SMP/E installs the deleting SYSMOD, it will identify which SYSMODs are being deleted.
2. SMP/E tracks when a module is deleted from a load module composed of modules to be deleted and modules not to be deleted. For each deleted module, SMP/E keeps a record of the connection between the deleted module and the load module. If any of these deleted modules are ever reintroduced, SMP/E looks for load modules having a record of a connection to those modules, and automatically rebuilds the load modules to include these modules again.

If you are replacing a product that contained cross-product modules or load modules, and the new release of the product eliminates the previous cross-product connections without deleting the modules or load modules that were involved, you need to ensure through packaging of the new release that SMP/E does not try to perpetuate the previous cross-product connections. For examples, see 9.6.4.1, “Linking a Module from Another Function” on page 107 and 9.6.4.2, “Linking Modules into a Load Module for Another Function” on page 108.

Packaging Rules (++VER DELETE)
<ul style="list-style-type: none">□ 6700. If the DELETE operand is used in a base function, it can specify the FMID of a base function or a dependent function. If the DELETE operand is used in a dependent function, it can only specify the FMID of a dependent function.□ 6800. Base functions (other than the initial release) must use ++VER DELETE to delete all previous releases and versions of the product. Note: Optionally, dependent functions can delete previous releases and versions of the product.□ 6900. A language-support dependent function must not delete an additive dependent function, and vice versa.□ 7000. A function cannot delete itself.

For more information about deleting SYSMODs, refer to 4.2.3.4, “Deleting and Superseding SYSMODs” on page 27.

- You should specify additive dependent functions that are applicable to a deleted base function. This allows customers to determine what is deleted by a function by reading the associated MCS. (Specifying these functions is for documentation purposes only. Dependent functions are automatically deleted when the associated base functions are deleted.)
- It is not necessary to specify language-support dependent functions that are applicable to a deleted base function. These functions are automatically deleted when the associated base functions are deleted.
- To improve SMP/E performance during installation, very large products should consider providing users with an example of how to package the ++VER DELETE information separately in a dummy function SYSMOD.

This dummy function SYSMOD is received, applied, and accepted to delete the previous releases of your product from the existing target and distribution libraries, and UCLIN is run to delete the SYSMOD entries for the deleted function and for the dummy function. The new release of the product is then installed.

For example, assume the previous release of your product is MYFUNC1, and you want to explain to users how to delete it with 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, along with the instructions you should provide to your users:

```
++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 */
        (DELFUNC MYFUNC1) /* Delete the FMIDs from the */
                               /* GLOBALZONE entry.      */.
```

End of Packaging Recommendations

7.2.5 Specifying Mutually Exclusive SYSMODs (NPRE)

The NPRES operand is an optional, multiple-entry operand. It indicates which function SYSMODs are mutually exclusive and cannot exist in the same zone as the specifying function. The SYSMOD ID specified on the NPRES operand cannot be already installed and must not be superseded by a SYSMOD being installed concurrently. These are called negative prerequisite SYSMODs. Using the NPRES operand for mutually exclusive functions is shown under Chapter 13, “SYSMOD Packaging Examples” on page 156. Also see 4.2.3.3, “Negative Prerequisite SYSMODs” on page 27 for more information.

Packaging Rules (Mutually Exclusive Versions)

- 7200. If the NPRES operand is used in a base function, it can only specify the FMID of a base function. If the NPRES operand is used in a dependent function, it can specify the FMID of a base function, the FMID of a dependent function, or both. In either case, all functions involved must be applicable to the same SREL.

7.2.6 Specifying Prerequisite Relationships (PRE)

The PRE operand is an optional, multiple-entry operand for dependent functions. It indicates which SYSMODs are prerequisites for the specifying SYSMOD. A prerequisite must either be already installed, or must be installed concurrently with the specifying SYSMOD. Using the PRE operand for prerequisite SYSMODs is shown under Chapter 13, “SYSMOD Packaging Examples” on pages 139, 152, and 153. See 4.2.3.1, “Prerequisite SYSMODs” on page 26 for more explanation.

Note: You cannot use PRE to assume ownership of an element from another function. A dependent function must use the FMID operand, and a base function must use the VERSION operand on the ++*element* statement.

Packaging Rules (++VER PRE)

- 7400. The PRE operand can be used only in a dependent function. It can specify the FMID of a base function (other than its own base) or a dependent function, or it can specify a PTF number. In any case, all functions involved must be applicable to the same SREL.
Note: Do not use the PRE operand in a dependent function to indicate its own base function. You must use the FMID operand for this purpose.
- 7500. The specified prerequisite (or a valid replacement) must be available as long as the specifying SYSMOD is available. When neither the prerequisite function nor the replacement SYSMOD is available, all the functions specifying the prerequisite must be repackaged.
- 7600. If a dependent function specifies a PTF as a prerequisite, the dependent function and the PTF must be applicable to the same base function.

7.2.7 Superseding SYSMODS (SUP)

The SUP operand is an optional, multiple-entry operand. It indicates which SYSMODs are contained in and replaced by this SYSMOD. For example, it could be used for a new release of a dependent function or for a service update. When a SYSMOD specifies SUP on its ++VER statement, this indicates to SMP/E that the superseded SYSMODs do not need to be installed once the superseding SYSMOD has been installed. Using the SUP operand for superseding SYSMODs is shown under Chapter 13, “SYSMOD Packaging Examples” on pages 138, and 144. Table 10 on page 28 also provides a comparison of deleting and superseding SYSMODs.

Note: You cannot use SUP to assume ownership of an element from another function. A dependent function must use the FMID operand, and a base function must use the VERSION operand on the ++*element* statement.

Packaging Rules (++VER SUP)

- 7700. If the SUP operand is used in a base function, it can specify the FMID of a base function, the FMID of a dependent function, a PTF number, or an APAR number. If the SUP operand is used in a dependent function, it can specify the FMID of a dependent function, a PTF number, or an APAR number. In either case, all functions involved must be applicable to the same SREL.
- 7800. A function must provide all the supported function contained in all the SYSMODs it supersedes.
- 7900. All the superseded SYSMODs must be in the same product as the superseding SYSMOD.
- 8000. For each environment (++VER FMID and SREL), all the elements in the superseded SYSMODs must be contained either in the superseding SYSMOD or in the combination of the superseding SYSMOD and its requisites (other SYSMODs specified on the ++VER REQ operand), unless the element is deleted by the superseding SYSMOD.
- 8100. The environment of a superseded SYSMOD must not be at a higher functional level than the level of the superseding function.
 - If the superseded SYSMOD is a base function, it must apply to the same SREL as the superseding SYSMOD.
 - If the superseded SYSMOD is a dependent function, it must apply to the same SREL as the superseding SYSMOD. In addition, the superseded dependent function must do one of the following:
 - Be applicable to the same base function as the superseding dependent function
 - Be applicable to a lower-level function than the superseding function
- 8200. A new release of a base function can supersede a previous release of that base function only if it also deletes that previous release. Likewise, a new release of a base function can supersede a dependent function applicable to a previous release of that base function only if the new release also deletes that dependent function.
- 8300. A new dependent function can supersede previous releases of that dependent function only if it also deletes those releases.
- 8500. A superseding function (or its requisites) must carry on the SYSMOD relationships defined in the superseded function SYSMODs. Table 14 on page 63 shows the relationships and processing information that the superseding SYSMOD or its requisites may need to include from the superseded SYSMODs.

Note: Table 14 on page 63 also applies to deleting SYSMODs and the information that they or their requisites may need to include from the deleted SYSMODs.

- A new release of a function should both delete and supersede the previous release if all of the following are true:
 - The new release contains at least all the function that was in the previous release.
 - If other products specified the deleted function as a requisite, all the internal and external interfaces used by those other products are unchanged in the new release.
 - Other products that specified the previous release as a requisite can run with the new release.
- Evaluate a replacement function using Table 10 on page 28 as a guide. If the replacement function matches that description, then the preferred and recommended way to replace the previous function is to both delete and supersede it.

End of Packaging Recommendations

<i>Table 14 (Page 1 of 2). Considerations for Superseding (and Deleting) SYSMODs</i>	
If the superseded (and deleted) SYSMOD(1) specified this:	Evaluate whether the statement is still valid and do the following as appropriate:
++VER PRE(<i>sysmod</i> ,...)	Specify ++VER PRE or SUP for the same SYSMODs (<i>or</i>) Specify ++VER PRE or REQ for another SYSMOD(3) that is either superior to or that specifies ++VER PRE or SUP for the same SYSMODs
++VER REQ(<i>sysmod</i> ,...)	Specify ++VER SUP, PRE, or REQ for the same SYSMODs (<i>or</i>) Specify ++VER PRE or REQ for another SYSMOD(3) that is either superior to or that specifies ++VER SUP, PRE, or REQ for the same SYSMODs
++VER SUP(<i>sysmod</i> ,...)	Specify ++VER SUP for the same SYSMODs
++VER VERSION(<i>sysmod</i> ,...) or ++element VERSION(<i>sysmod</i> ,...)	Specify ++VER VERSION or ++element VERSION for the same SYSMODs Note: The ++VER VERSION value affects all new or replacement elements that do not specify an overriding ++element VERSION value.
++IF FMID(<i>fmid</i>) REQ(<i>sysmod</i> ,...)	Specify ++IF REQ for the same SYSMODs (<i>or</i>) Specify ++IF REQ for another SYSMOD(3) that is either superior to or that specifies ++VER SUP, PRE, or REQ for the same SYSMODs (<i>or</i>) Specify ++VER PRE or REQ for another SYSMOD(3) that is either superior to or that specifies ++IF REQ for the same SYSMODs Note: All of the ++IF statements must specify the same FMID value as the original ++IF statement.
++HOLD statement	Evaluate to see whether the ++HOLD statement is required, or can be deleted by updating the installation documentation
++MOVE statement	Include the ++MOVE statement, unless: <ul style="list-style-type: none"> • An element statement deletes the element • A ++DELETE statement deletes the load module. Note: When moving an element, make sure to specify the new libraries on the DISTLIB and SYSLIB operands in the appropriate ++element statements.

Table 14 (Page 2 of 2). Considerations for Superseding (and Deleting) SYSMODs

If the superseded (and deleted) SYSMOD(1) specified this:	Evaluate whether the statement is still valid and do the following as appropriate:
++RENAME statement	Include the ++RENAME statement, unless a ++DELETE statement deletes the load module.
++DELETE statement	Include the ++DELETE statement.
++JCLIN statement and JCLIN data	Include the ++JCLIN statement and JCLIN data, merging the ++JCLIN operands and JCLIN data from SYSMOD(1). Note: If several SYSMODs are superseded (or deleted), merge the JCLIN data so that the most recent data is properly reflected in SYSMOD(2).
++MOD CSECT(<i>name</i>)	Include the CSECT data.
++MOD LMOD(<i>name</i>)	Evaluate to see whether the LMOD operand is still required on the ++MOD statement. Note: The new JCLIN data may eliminate the need for the LMOD operand.
Element updates	Merge all of the updates contained in the superseded (or deleted) SYSMODs into the new elements
UCLIN data	Evaluate to see whether the UCLIN data is required, or whether an alternative to UCLIN may be used
UCLIN to move an element or load module	Use a ++MOVE statement to move the element or load module, unless: <ul style="list-style-type: none"> • An element statement deletes the element • A ++DELETE statement deletes the load module.
UCLIN to rename a load module	Use a ++RENAME statement to rename the load module, unless a ++DELETE statement deletes the load module.
UCLIN to delete a load module	Use a ++DELETE statement to delete the load module.

Packaging Rules (Moving and Replacing Elements)

- 8600. The ++VER statement for each SYSMOD that contains an element that is replaced or moved to a new library must use the PRE or SUP operand to specify the previous SYSMOD, if any, that also replaced or moved that element.

7.2.8 Defining Ownership (VERSION)

The VERSION operand specifies one or more dependent function SYSMODs whose elements should be considered functionally lower than the version of those elements in the specifying function SYSMOD. The VERSION operand is also used to add a version of an element to a dependent function when that element exists only in lower-level dependent functions.

When a dependent function SYSMOD that specifies the VERSION operand on the ++VER or element statement is installed, the dependent function will assume ownership of the elements from the functions specified on the VERSION operand. Subsequent processing of service SYSMODs or USERMODs applicable to the functions that previously owned the elements will not update or replace the affected elements.

Packaging Rules (++VER VERSION)

- 8700. You must specify the lower-level function SYSMODs on the VERSION operand of each ++VER statement in the higher-level function SYSMOD.

VERSION is required to establish which elements are functionally higher when SYSMODs for different dependent functions have elements with the same name and type in common. Also, specifying the lower-level function SYSMODs on the VERSION operand on the ++VER statement in the higher-level function SYSMODs ensure that ownership of the elements is given to the highest level SYSMOD.
- 8800. If a dependent function uses the VERSION operand, any subsequent function replacing this dependent function must contain all the elements whose ownership was assumed by the dependent function.
- 9000. A new release of a dependent function can have elements in common with a lower-level dependent function for the same base function. If so, the new release must incorporate those elements and, if the lower-level dependent function is not deleted, must establish the superiority of its version of those elements, as well as its installation relationship with the lower-level function. The superiority of the elements is established by the VERSION operand on either the ++VER or element statement. The installation relationship is established by either the PRE or SUP operand on the ++VER statement. For more information, see the descriptions of these operands elsewhere in this chapter.
- 9100. VERSION must specify all the dependent functions that are functionally lower than the specifying function and that include the elements to be versioned.
- 9200. The VERSION operand must be specified on the ++VER statement if all elements affected by this SYSMOD are to be versioned the same way. The VERSION operand must be specified on the element statement if individual elements can be versioned differently.

Note: For the VERSION operand to take effect, the specified functions must be installed in the same zone as the specifying SYSMOD.

Packaging Recommendations

If use of the VERSION operand between two products is unavoidable, it is the responsibility of the development owner of Product B to ensure that the development owner of Product A understands and agrees to what has been done.

VERSION can also be specified on an element statement to establish the functional level of elements and override the VERSION values specified on the ++VER statement. However, the VERSION operand on the element statement is not additive; it does not automatically take over ownership from the functions specified on the ++VER VERSION operand. To take over ownership from any of the functions specified on the ++VER VERSION operand, you must repeat those values on the VERSION operand for the element statement.

7.3 ++IF Statement

The ++IF statement defines conditional requisites. This is an optional statement associated with the ++VER statement that precedes it. Several ++IF statements may be associated with a single ++VER statement. If a SYSMOD contains several ++VER statements, there may be ++IF statements associated with each one. Using the ++IF statement is shown under Chapter 13, “SYSMOD Packaging Examples” on pages 140, 143, 147, 149, 153, 154, and 155.

The operands of the ++IF statement are used to:

- Specify the function to which the condition applies
- Specify the SYSMODs that must be installed if the condition exists

7.3.1 Specifying the Function to which the Condition Applies (FMID)

The FMID operand is a required, single-entry operand. It indicates the function to which the conditional requisite applies.

Packaging Rules (++IF FMID)

- 9210. The ++IF statement can be used in a base function or a dependent function. In both cases, the FMID operand can specify either a base function or a dependent function.
- 9300. The function cannot specify its own FMID.
 - Note:** This Rule does not apply to products that require installation using the OS/390 Release 3 (or later) level of SMP/E.
- 9400. This rule has been deleted.
- 9500. If the FMID operand is used in a base function, the specified SYSMOD must be in a previous *version* of the product.

For example, Version 2 Release 2 of a product cannot specify ++IF FMID for Version 2 Release 1; however, it can specify Version 1 Release 3.

Note: A dependent function can specify any function SYSMOD, regardless of whether two functions are part of the same product or product version.

Packaging Recommendations

- The ++IF MCS should include a comment to identify the product required by the FMID operand.

End of Packaging Recommendations

7.3.2 Specifying Requisite Conditions (REQ)

The REQ operand is a required, multiple-entry operand. It specifies one or more SYSMODs that must be installed if the function SYSMOD specified on the FMID operand of the ++IF statement is installed.

- If the specified function is already installed (or is currently being installed) in the same zone where the specifying SYSMOD is being installed, the requisite must

also be installed in that zone; otherwise, the specifying SYSMOD will not be installed.

- If the specified function is not yet installed in the zone, SMP/E saves the information from the ++IF statement in case the specified function is installed later.

In both cases, SMP/E saves requisite data from the ++IF statement, even if the function specified on the ++IF FMID operand is restored or deleted.

Note: The specified SYSMOD may be in the same or a different product or product version as the specifying SYSMOD.

Packaging Rules (++IF REQ)

- 9600. The REQ operand can be used in a base function or a dependent function. In both cases, the REQ operand can specify either a dependent function or a PTF number.
- 9700. Any dependent function specified on the REQ operand (or a valid replacement) must be announced and must be available as long as the specifying SYSMOD is available.
- 9800. If the specified conditional requisite is a function and it is deleted by a new release of that function, one of the following must be done:
 - The new release can also supersede the specified requisite function. This way, the function specifying the requisite does not need to be repackaged.
 - If the specified requisite function is to be deleted by a new release without also being superseded, the specifying function must be repackaged and redesigned to refer to the new release as the requisite.
- 9900. If the specified conditional requisite is a PTF, any subsequent replacement must supersede the specified PTF. This eliminates the need to repackage the specifying function to redefine the conditional requisite.
- 10000. A SYSMOD cannot specify both a conditional and unconditional relationship for the same SYSMOD ID.

For example, the following statements cannot appear in the same SYSMOD:

```
++VER REQ(ABC1234) .  
++IF FMID(Z) REQ(ABC1234) .
```

Note: This Rule does not apply to products that require installation using the OS/390 Release 3 (or later) level of SMP/E.

- 10010. If the specified SYSMOD is a dependent function, the FMID to which it applies must either:
 - Match the FMID specified on the associated ++IF statement contained in the specifying SYSMOD
 - Unconditionally coexist with the FMID specified on the associated ++IF statement contained in the specifying SYSMOD

Provide ++IF REQs for all functionally required service, with comments explaining the reason for the REQ.

_____ End of Packaging Recommendations _____

7.4 ++HOLD Statement

The ++HOLD statement alerts the user that special SMP/E handling is required before the SYSMOD can be applied or accepted.

IBM Software Delivery Restrictions

The CLASS operand must not be used.

++HOLD statements are not permitted in function SYSMODs. This is a restriction of the **IBM Software Delivery Solutions** process.

End of IBM Software Delivery Restrictions

7.5 ++element Statement

Element statements describe the elements contained in a SYSMOD and are used by SMP/E to select which elements should be installed in the target and distribution libraries. If an element statement is not provided for an element, the element is not installed, even if it was defined in the JCLIN data. The following statements can be used to add or replace elements:

- ++MAC describes a new or replacement macro.
- ++MOD describes a new or replacement module.
- ++SRC describes new or replacement source code.
- ++hfs_element describes new or replacement elements that are installed in a hierarchical file system (HFS).
- Data element MCSs describe new or replacement elements that are not macros, modules, or source code. Types of data elements are shown in Table 12 on page 45 under 6.4, “Data Element Types.”

A single SYSMOD can contain any one of the following statements or combinations of statements for a given macro, module, or source element name.

- **++MAC statement:** Macros may be used during source (++)SRC assemblies and can be used to assemble source that is not defined by ++SRC statements. The resulting object modules are written to a work data set that is used as SYSPUNCH input to link-edit the modules into the target libraries. If you package code that is to be processed this way, you must provide for the JCLIN data that defines the assembly and link-edit steps to SMP/E. (This JCLIN data may be packaged with the code or created during a generation procedure.)
- **++MOD statement:** A module can be link-edited into a load module in a target library, or, for a single-module load module, can be copied into a target library. If you package code that is to be processed this way, you must provide for the JCLIN data that defines the link-edit or copy steps to SMP/E. (This JCLIN data may be packaged with the code or created during a generation procedure.)
- **++SRC statement:** Source can be supplied without the corresponding modules to cause the source to be assembled. The resulting object modules are written to a work data set that is used as SYSPUNCH input to link-edit the modules into the target libraries. If you package code that is to be processed this way, you must provide for the JCLIN data that defines the assembly and link-edit steps to SMP/E. (This JCLIN data may be packaged with the code or created during a generation procedure.)
- **++SRC and ++MOD statements:** A module may be provided in both source and executable forms. (Each form represents a different element type, and both must be in the same FMID.) In this case, the source will not be assembled. Users who do not need to change the source code will have an executable module they can install. Users who do need to change the source code can make those changes to the source so that it will be assembled to create an object module. The object module is installed as described above for the ++MOD statement.

If you package a module that is to be processed this way, you must provide for the JCLIN data that defines the link-edit or copy steps to SMP/E. (This JCLIN

data may be packaged with the code or created during a generation procedure.)

_____ Packaging Recommendations _____

If you package an element with a ++SRC statement, you should also include the associated ++MOD statement.

_____ End of Packaging Recommendations _____

_____ Packaging Recommendations _____

Do not use ++MAC, ++MOD, or ++SRC statements to package elements that are not macros, modules, or source, respectively. Use data element statements or HFS element statements (as appropriate) to package such elements.

_____ End of Packaging Recommendations _____

Packaging Rules (DISTLIB for Elements)

- 10100. Do not use SYSPUNCH as the DISTLIB. It is used by SMP/E and other products to process assembled modules.
- 10110. Do not specify a pathname in a hierarchical file system (HFS) as the DISTLIB.
- 10111. Do not specify SMP/E temporary data sets (SMPLTS, SMPMTS, SMPPTS, SMPSTS, etc.) as DISTLIB or SYSLIB values on MCS.
- 10112. If you must use a new library, it must have a unique ddname and a unique data set name to avoid conflicts with other products. For more information on naming distribution libraries, see 10.4, "Library Names" on page 124.

For details on specifying these operands, see the *OS/390 SMP/E Reference* or *SMP/E R8.1 Reference*, SC28-1107

Chapter 8. Using MCS Statements to Manipulate Elements and Load Modules

MCS statements can help you address packaging goals that at one time could be done only through UCLIN. This chapter describes how you can use MCS statements to:

- Move macros, modules, source, and load modules (++MOVE statement)
- Rename load modules (++RENAME statement)
- Delete load modules (++DELETE)
- Delete elements (++element DELETE)
- Enable load module changes at the CSECT level (++MOD CSECT)
- Change ownership of elements

Note: Regardless of the order in which ++MOVE, ++RENAME, and ++DELETE statements are coded in a SYSMOD, they are always processed in this order:

- ++MOVE
- ++RENAME
- ++DELETE

Afterwards, ++JCLIN statements are processed, followed by element statements.

Table 15 summarizes how you can use MCS statements to manipulate elements and load modules.

Goal	Packaging Solution	Where to Find More Information
Add a module to a load module.	To add a module to an existing load module, use the LMOD operand on the ++MOD statement. To add a module and create a new load module, use JCLIN data.	See 9.2, "When Do You Need JCLIN?" on page 88.
Change the owner of an element.	If a new function deletes an old function, the DELETE operand on the ++VER statement indicates that the owner has changed.	See 7.2.4, "Deleting SYSMODs (DELETE)" on page 57.
	If a new dependent function introduces a higher-level version of the element, you can use the VERSION operand on the ++VER or element statement to indicate that the owner has changed.	See 7.2.8, "Defining Ownership (VERSION)" on page 64.
	If an element is being migrated from one base function to another, you can use the VERSION operand on the element statement to indicate that the owner has changed.	For more information on migration to a new function, see 13.8.3, "Migrating Elements by Updating Both Functions" on page 163. For more information on migration using a PTF, see 13.8.4, "Migrating Elements by Using a PTF" on page 164.

Table 15 (Page 2 of 2). Performing Actions on Elements and Load Modules

Goal	Packaging Solution	Where to Find More Information
Move a macro, module, source, or load module.	Use the ++MOVE statement.	See 8.1, "Moving Elements and Load Modules (++)MOVE)" on page 75.
Rename a load module.	Use the ++RENAME statement.	See 8.2, "Renaming Load Modules (++)RENAME)" on page 78.
Delete a load module.	Use the ++DELETE statement.	See 8.3, "Deleting Load Modules (++)DELETE)" on page 81.
Delete a module (CSECT) from a load module.	Use the ++MOD CSECT operand	See 8.5, "Enabling Load Module Changes at the CSECT Level (++)MOD CSECT)" on page 84.

There may be times when you cannot avoid UCLIN changes. In these cases, follow these rules:

Packaging Rules (UCLIN)
<ul style="list-style-type: none"> □ 10115. Make sure the UCLIN data can be processed using SMP/E. Provide instructions for SMP/E customers to insert the appropriate SET BOUNDARY command before the UCLIN data. SMP/E needs the SET command to update the correct zone with the UCLIN data. □ 10117. Package the UCLIN data as an element that is installed in an appropriate data set for sample code. This allows customers, as well as product installation procedures, to have access to the UCLIN. You can package the data as sample code using the ++SAMP MCS statement. Use standard names for the element, the target library, and the distribution library. □ 10119. Describe any UCLIN data requirements and procedures in the installation instructions. This documentation must provide enough information so that the customer can either invoke SMP/E or use the SMP/E dialogs to process the UCLIN data. You must use a ++HOLD statement, even if your documentation clearly and fully explains how to handle the UCLIN.

8.1 Moving Elements and Load Modules (++MOVE)

The ++MOVE statement moves a macro, module, source, or load module from its current library to another library.

IBM Software Delivery Restrictions

The ++MOVE statement is not allowed for data elements, HFS elements, or ++PROGRAM elements. This is an SMP/E restriction.

End of IBM Software Delivery Restrictions

This is an optional statement. If you include it, it must immediately follow the last ++HOLD statement, or if there are none, it must follow the last ++VER statement or the last ++IF statement associated with that ++VER statement. It must precede all other MCS statements (++RENAME, ++DELETE, ++JCLIN, and element statements).

Packaging Rules (++MOVE)

- 10200. A dependent function can contain a ++MOVE statement for an element or load module it does not contain only if the element or load module is owned by the base function to which the dependent function applies, or by another dependent function for the same base function. In either case, the moving dependent function must specify the owning function as a prerequisite.

If a previous dependent function has performed a ++MOVE on the element or load module, then the new dependent function must specify that dependent function as a prerequisite.

- 10300. A function can contain only one ++MOVE statement for a given element.
- 10400. A function can contain no more than two ++MOVE statements for a given load module, one for each SYSLIB defined for the load module.
- 10500. All MCS statements following the ++MOVE statements and referring to the elements or load modules that were moved must reflect the new libraries for those elements or load modules. All SYSMODs applied subsequent to the move must reflect the new libraries for those elements or load modules.
- 10600. All changes caused by a ++MOVE MCS must also be specified in any JCLIN and SYSGEN macros that refer to the moved member.
- 10700. If SYSMOD(1) defines or moves an element, subsequent SYSMODs containing that element must specify SYSMOD(1) as a prerequisite.
- 10800. If SYSMOD(1) moves a given load module using a ++MOVE statement, any SYSMOD that supersedes SYSMOD1 must also contain the ++MOVE statement.
- 10900. If an element or load module to be moved to a new SYSLIB is a member of a totally copied library, the moving function must also move the same element or corresponding module to a new distribution library.

Packaging Recommendations

- If an element needs to be moved, a ++MOVE statement must be used instead of UCLIN.
- A base function should not contain a ++MOVE statement, unless a PTF containing the statement was integrated into a service update of that function.
- New releases of a base function do not own elements that would need to be moved from one library to another. However, there may be shared load modules that should be moved. In this instance, a base function may contain a ++MOVE for the shared load module.

End of Packaging Recommendations

Two ++MOVE statements are allowed for a load module because load modules can exist in two target libraries.

SMP/E processes the ++MOVE statements in a SYSMOD first; therefore, any MCS statements after the ++MOVE need to reflect the element's new library.

You must ensure that a totally copied library structure (a DISTLIB that was totally copied to a target library) is defined to SMP/E. If a load module was moved from the defined target library, its corresponding module must be moved to a new distribution library. This ensures that the totally copied structure is defined to SMP/E.

8.2 Renaming Load Modules (++RENAME)

The ++RENAME statement changes the name of a load module.

This is an optional statement. If you include it, it must immediately follow the last ++HOLD or ++MOVE statement, or if there are none, it must follow the last ++VER statement or the last ++IF statement associated with that ++VER statement. It must precede all other MCS statements (++DELETE, ++JCLIN, and element statements).

All MCS statements that follow the ++RENAME statements that refer to the load modules that were renamed must reflect the new name for those load modules.

SMP/E will not rename any aliases associated with the specified load module.

Packaging Rules (++RENAME)

- 11000. A dependent function can contain a ++RENAME statement for a load module associated with either the base function to which it applies, or with another dependent function that is applicable to that same base function and that is required by the function containing the ++RENAME statement.
- 11100. All changes caused by a ++RENAME MCS must also be specified in any JCLIN and SYSGEN macros that refer to the old name of the load module.
- 11200. A function can contain only one ++RENAME statement for a given load module.
- 11300. If SYSMOD(1) renames a given load module using a ++RENAME statement and SYSMOD(2) defines that load module under its new name with JCLIN data, SYSMOD(2) must specify its relationship to SYSMOD(1) using the PRE, DELETE, or SUP and DELETE operands on its ++VER statement.
- 11400. If SYSMOD(1) defines a given load module and SYSMOD(2) renames that load module using a ++RENAME statement, SYSMOD(2) must specify its relationship to SYSMOD(1) using the PRE operand on its ++VER statement.
- 11500. If a load module being renamed was totally copied from a distribution library into a target library (defined by JCLIN data as a totally copied load module), this function must also use a ++MOVE statement to move the identically named element (++MOD) to a new distribution library.
- 11600. If a dependent function is renaming a load module, that function must refer to the last previous lower-level dependent function (if any) that (1) moved the load module being renamed or (2) renamed a load module to the name of the load module being renamed again.
 - If that previous dependent function moved the load module being renamed, this dependent function can either delete or supersede and delete that dependent function, or specify it as a prerequisite.
 - If the previous dependent function renamed a load module to the name of the load module being renamed again, this dependent function must specify that previous dependent function as a prerequisite.

Packaging Recommendations

- A base function should not contain a ++RENAME statement, unless a PTF containing the statement was integrated into a service update of that function.
New releases of a base function do not own elements that would need to be renamed. However, there may be shared load modules that should be renamed. In this instance, a base function may contain a ++RENAME for the shared load module.
- If you want to rename a load module and use inline JCLIN to create a new load module with the original name of the renamed load module, you must package

++RENAME Statement

your changes in two SYSMODs: one to rename the existing load module, and one to create the new load module.

The two SYSMODs must not state any relationship to each other and must be applied separately: first the SYSMOD that renames the existing load module, then the one that creates the new load module.

If the SYSMODs need to be restored, they must also be restored separately, in the reverse order of the installation: first the SYSMOD that created the new load module, then the one that renamed the existing load module.

End of Packaging Recommendations

8.3 Deleting Load Modules (++DELETE)

The ++DELETE statement deletes a load module and any of its aliases from its current target library. It can also delete the aliases without deleting the load module.

This is an optional statement. If you include it, it must immediately follow the last ++HOLD, ++MOVE, or ++RENAME statement, or if there are none, it must be associated with that ++VER statement or the last ++IF statement associated with that ++VER statement. It must precede all other MCS statements (++JCLIN and element statements).

Using the ALIAS operand deletes the alias of a load module without deleting the load module itself. If the ALIAS operand is not specified, the load module and all of its aliases are deleted.

Packaging Rules (++DELETE)

- 11700. A dependent function can contain a ++DELETE statement for a load module associated with either the base function to which it applies, or with another dependent function that is applicable to that same base function and that is required by the function containing the ++DELETE statement.
- 11800. A function can contain only one ++DELETE statement for a given load module.
- 11900. A function containing a ++DELETE statement must also include the appropriate changes for its JCLIN or SYSGEN macros (if any) to reflect the change.
- 12000. If SYSMOD(1) deletes a given load module using a ++DELETE statement and SYSMOD(2) defines that load module with JCLIN data, SYSMOD(2) must specify its relationship to SYSMOD(1) using the PRE, DELETE, or SUP and DELETE operands on its ++VER statement.
- 12100. If SYSMOD(1) defines a given load module with JCLIN data and SYSMOD(2) deletes that load module using a ++DELETE statement, SYSMOD(2) must specify its relationship to SYSMOD(1) using the PRE or FMID operand on its ++VER statement.
- 12200. A dependent function that is deleting a load module must refer to the last previous lower-level dependent function (if any) that (1) moved the load module being deleted or (2) renamed a load module to the name of the load module being deleted.
 - If that previous dependent function moved the load module being deleted, this dependent function can either delete or supersede and delete that dependent function or specify it as a prerequisite.
 - If that previous dependent function renamed a load module to the name of the load module being deleted, this dependent function can either delete or supersede and delete that dependent function or specify it as a prerequisite.
- 12210. If a SYSMOD is deleting an alias for a load module but not the load module itself (ALIAS is specified on the ++DELETE statement), you must reflect this change using JCLIN. To do this, include a ++JCLIN statement with JCLIN data that contains 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 CSI.

Packaging Recommendations

A base function should not contain a ++DELETE statement.

Although a program object 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 without deleting the program object. Instead, you need to resupply JCLIN to define the program object 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.

End of Packaging Recommendations

If a load module resides in two or more system libraries, you need only one ++DELETE statement. Refer to the *SMP/E Reference* manual for information about the ++DELETE statement.

8.4 Deleting Elements from Libraries and SMP/E Data Sets

The DELETE operand on an element MCS indicates that the element is to be removed from the target libraries, distribution libraries, and SMP/E data sets. This operand can be used for all element types. This is an optional operand and is used only in dependent functions.

Packaging Rules (DELETE for Elements)

- 12300. A dependent function must not delete a macro or source element from a lower-level function (its parent base function or a lower-level dependent function for the same parent base function), because a PTF that is applicable to the lower-level function may need to update the element (such as by using ++MACUPD or ++SRCUPD). If that element were deleted, there would be nothing to update, and the PTF needed for the lower-level function could not be installed.
- 12310. This rule has been deleted.

8.5 Enabling Load Module Changes at the CSECT Level (++MOD CSECT)

The CSECT operand lists all the CSECTs that are contained in a module. Defining the contents of a load module by CSECT name allows SMP/E to change a load module at the CSECT level when a function or module is being deleted.

Packaging Rules (++MOD CSECT)
<ul style="list-style-type: none">□ 12400. If a SYSMOD changes the CSECTs contained in an existing module, CSECT must be specified and must list all the CSECTs in that module. This is true even if the module now contains only one CSECT whose name matches the module name on the ++MOD statement.□ 12500. This rule has been deleted. It has been replaced by rule 131.3 in 9.6, "Link-Edit Steps" on page 94.

Packaging Recommendations

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. If this is done, SMP/E can change the affected load module at the CSECT level when a function or module is being deleted.

End of Packaging Recommendations

Note: Simply ordering the INCLUDE statements is not sufficient to define the order of CSECTS, because SMP/E replaces CSECTS when relinking the load module and could change the order of the CSECTS.

Refer to 9.2, "When Do You Need JCLIN?" on page 88 for information about using JCLIN to define load modules.

8.6 Defining Ownership of Elements (++element VERSION)

The VERSION operand is required to establish which elements are functionally higher when different SYSMODs ship elements with identical element names and element types. For example, it could be used to add elements to a dependent function when those elements already belong to a lower-level dependent function. Or, it could be used when two language-support dependent functions contain a common element that was not translated (such as a CLIST).

Note: Although SMP/E uses the VERSION operand to determine the correct version of the elements to be installed, it does not use VERSION to determine the relationships of SYSMODs being installed. You must specify that information on the PRE or SUP operand of the ++VER statement.

This operand is optional for dependent functions. It is not allowed in base functions.

You may need to create a new version of an element that already exists in a product. For example, you may need to add a user function to or provide service

for an existing element. There are two ways of providing a new version of an element:

1. **Dependent function.** A new dependent function, or a new release of an existing dependent function, can provide a new version of an element.
2. **PTF.** A PTF can be used to create a new version of an element in a base function or a dependent function.

Packaging Rules (VERSION for Elements)

- 12510. VERSION is required to establish which elements are functionally higher when SYSMODs for different functions have elements with the same type and name in common. You must specify the lower-level function in the VERSION operand of the element statement in the SYSMOD associated with the higher-level function.
- 12600. The specified functions must be able to coexist with the specifying SYSMOD.
- 12700. The specified functions must contain the element described by the element statement.
- 12800. For dependent functions, VERSION must specify all the dependent functions that are functionally lower than the specifying function and include the element being versioned.
- 12900. If VERSION is also specified on a ++VER statement for this SYSMOD, the VERSION operand on the element statement overrides the VERSION values specified on the ++VER statement. However, the VERSION operand on the element statement is not additive; it does not automatically take over ownership from the functions specified on the ++VER VERSION operand. To take over ownership from any of the functions specified on the ++VER VERSION operand, you must repeat those values on the VERSION operand for the element statement.
- 13000. The VERSION operand must be specified on the element statement if individual elements may be versioned differently. The VERSION operand must be specified on the ++VER statement used if all elements affected by this SYSMOD are to be versioned the same way.

Packaging Recommendations

`++element` VERSION should be used only by different functions of the same product. If the VERSION operand is used by a function that is not part of the same product as the element it wants to assume ownership of, unpredictable results may occur. For example, if Product A owns an element and Product B uses VERSION to assume ownership of that element, it may not be clear which product should ship a given PTF for that element.

If use of the VERSION operand between two products is unavoidable, it is the responsibility of the development owner of Product B to ensure that the development owner of Product A understands and agrees to what has been done.

End of Packaging Recommendations

Chapter 9. Using JCLIN

JCLIN provides information to SMP/E about how to install a SYSMOD in the target and distribution libraries. JCLIN can be provided in several formats, such as assemble, copy, and link-edit steps. SMP/E processes these steps to determine the structure of the SYSMOD's elements. SMP/E builds and updates entries based on JCLIN data; however, it does not actually execute the JCLIN input.

To help you understand how to use JCLIN, this chapter describes:

- Providing JCLIN data for function SYSMODs
- When you need to use JCLIN
- General packaging rules for JCLIN data
- Assembler steps
- Copy steps
- Link-edit steps
- Examples of JCLIN data

For more information about JCLIN processing, see the *OS/390 SMP/E Commands* manual or the *SMP/E Release 8.1 Reference* manual.

9.1 Providing JCLIN Data for Function SYSMODs

There are several sources of JCLIN data:

- Data associated with a ++JCLIN statement
- Output from the SMP/E GENERATE command
- Stage 1 output JCL from a system, subsystem, or product generation

The output JCL from a generation procedure can be processed by the JCLIN command to update the CSI target zone with information about the products installed by that JCL. However, once the JCLIN command has processed that JCLIN data, the product information cannot be removed from the target zone unless the product is deleted or restored.

To avoid any potential problems this might cause your customers, package JCLIN data using a ++JCLIN statement. When customers apply a SYSMOD containing a ++JCLIN statement, SMP/E saves unchanged copies of target zone entries that will be updated by the JCLIN. This way, if customers need to restore the SYSMOD, they can do it because SMP/E saved the previous version of the entries.

For more information about JCLIN processing, see the *OS/390 SMP/E Commands* manual or the *SMP/E Release 8.1 Reference* manual.

Notes:

1. JCLIN data is processed only for macros, modules, and source. It is not processed for data elements, except to define totally copied libraries. It is not processed, and should not be specified, for elements installed in a *hierarchical file system (HFS)*. Such elements are defined by the ++*hfs_element* statement
2. SMP/E has no column limitations for operands beyond the normal JCL rules.

3. The ++JCLIN statement 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.

IBM Software Delivery Restrictions

The **IBM Software Delivery Solutions** process does not support the TXLIB operand. Therefore, it is not included in the list of ++JCLIN operands, nor is it described in the sections that follow.

End of IBM Software Delivery Restrictions

9.2 When Do You Need JCLIN?

You need JCLIN for a base function so that SMP/E has information about the structure of the product and target libraries:

- The library in which an element resides
- How modules are link-edited together for load modules
- Where the load modules exist and their characteristics

You also need JCLIN for changes introduced by a dependent function. You do not need to use JCLIN for structures and attributes that were not altered by the dependent function. This means you do not need JCLIN for every element; JCLIN is required only for those load modules with changed structure or attributes. Repeating JCLIN for unchanged elements increases the risk of errors.

Following are some situations that do not require JCLIN to be used:

- All elements (other than ++MOD elements that are totally copied modules) of a product are installed using a copy utility.
- A dependent function does not introduce new elements.
- A dependent function does not change the link-edit attributes for a load module.

Note: You never need JCLIN for data elements. SMP/E uses the SYSLIB and DISTLIB operands on the data element MCS statements to determine where the elements should be installed. The same is true for hierarchical file system (HFS) elements. SMP/E uses the SYSLIB and DISTLIB operands on the ++*hfs_element* statements to determine where the elements should be installed.

You need JCLIN when a dependent function does any of the following:

- Changes the link-edit attributes of a load module

The attributes of a load module include such things as RENT, REUS, and REFR.

- Changes the structure of a load module

Structure means the ENTRY, ORDER, and ALIAS statements that apply to a load module.

- Introduces a new load module

Refer to Table 15 on page 73 for information about using MCS statements to perform operations on elements and load modules. For example, you do not need JCLIN when a dependent function or PTF is adding a module to an existing load module. Instead, you should use the LMOD operand on the ++MOD statement.

9.3 General Packaging Rules for JCLIN Data

The following general rules define how JCLIN data must be coded.

Packaging Rules (JCLIN Data)
<ul style="list-style-type: none"> □ 13100. The combination of JCLIN data and element statements must completely describe all the elements in the function and their target and distribution libraries. □ 13110. A product's installation must not require the editing of the JCLIN. □ 13200. If the low-level qualifier of a data set name is in the format <code>xccczzzz</code>, as described in rule 140 in 10.4, "Library Names" on page 124, the low-level qualifier and the ddname must be identical. <p>NOTE: Since a DDNAME may refer to a subdirectory in the hierarchical file system (HFS), several DDNAMEs may point into one HFS. In these cases, the low-level qualifier and the ddname need not be identical.</p> <ul style="list-style-type: none"> □ 13300. Input data sets in link-edit steps must not be concatenated. <p>An exception to this rule is using the support for the automatic library call facility. For more information, see the description of the SYSLIB DD statement in 9.6.2, "Link-Edit Control Statements" on page 97.</p>

In addition to the above rules, the following recommendations apply to JCLIN data:

Packaging Recommendations
<ul style="list-style-type: none"> • Use the simplest possible JCL statements. • Specify all information in uppercase (verbs as well as values). <p>This is necessary to avoid syntax errors or incorrect results during SMP/E processing.</p> <p>Note: This convention does not apply to values on the ALIAS statement. These values must be specified in the desired case (uppercase or mixed-case), because they are used as is.</p> <ul style="list-style-type: none"> • Do not use update steps in JCLIN data; SMP/E ignores them. • In the JCLIN of the dependent function, describe only new or changed structure. <p>The JCLIN for a dependent function should not repeat data already provided in the JCLIN of the base function.</p> <ul style="list-style-type: none"> • Do not use abbreviations. <p>SMP/E may not recognize all abbreviations.</p>

- For copied members (except for ++MOD), use the SYSLIB and DISTLIB operands on the element statements instead of JCLIN to define copies.
- If possible, do not use continued utility control statements. Although SMP/E tries to support all existing formats of the utility control statements, it cannot completely duplicate the syntax checking of the utility. The safe method is to use the simplest format of the utility control statement without continuations.
- Test the JCLIN data as follows:
 - Perform RECEIVE, APPLY, and ACCEPT of the product on one system.
 - Perform RECEIVE, ACCEPT BYPASS(APPLYCHECK), GENERATE of the product on a second system.
 - Compare the SMP/E reports from the two products, checking for discrepancies.
 - Compare every library, member by member, between the two products, checking for discrepancies.
 - Run the JCLIN data outside of SMP/E and compare the resulting load modules with those built during the SMP/E installs. There should be no differences.

_____ End of Packaging Recommendations _____

_____ IBM Software Delivery Restrictions _____

- If a PTF introduces a new ++MOD requiring link-edit parameters other than the default, these parameters must be specified in the LEPARM operand of the ++MOD statement. Parameters specified in JCLIN data are not sufficient. This is a restriction of the **IBM Software Delivery Solutions** process.
- Products that require assemblies during APPLY processing may not require macro libraries provided by products in another SREL. This is a restriction of the **IBM Software Delivery Solutions** ServerPac process; the macro libraries will not be available during order build processing.

_____ End of IBM Software Delivery Restrictions _____

9.4 Assembler Steps

Packaging Rules (JCLIN Assembler Steps)
<ul style="list-style-type: none">□ 13400. Assembler steps must be identified by one of the following:<ul style="list-style-type: none">– EXEC PGM=IFOX00– EXEC PGM=IEV90– EXEC PGM=ASMA90– EXEC PGM=ASMBLR– EXEC ASMS

9.5 Copy Steps

Packaging Rules (JCLIN Copy Steps)

- 13500. Copy steps must be identified by the following:
 - EXEC PGM=IEBCOPY
- 13600. The RENAME function must not be used in JCLIN.
- 13700. If the SELECT MEMBER= statement is used to selectively copy elements, the COPY INDD=xxx,OUTDD=xxx control statements for selectively copied elements must include the comment TYPE=xxxx. The format of the TYPE comment on the COPY statement is:

```
COPY INDD=ddname ,OUTDD=ddname  TYPE=xxxx
```

where xxxx is MOD, MAC, SRC, or DATA.

Notes:

1. If the TYPE=xxxx parameter is not specified, the default used by SMP/E is TYPE=MOD.
2. TYPE=DATA is used for data elements.

Without this additional comment, the GENERATE command cannot determine what type of element is being copied. If the comment is not included, SMP/E assumes the element is a module and may create unnecessary module entries in the target or distribution zone.

For data elements and hierarchical file system (HFS) elements, you must use the SYSLIB and DISTLIB operands on the element statement to specify information used to install the element. During JCLIN processing, SMP/E bypasses any COPY SELECT statements that specify TYPE=DATA.

- 13800. The SELECT statement can specify either the name of the member to be copied or an alias name for the member. The SELECT statement for an alias must specify the comment "ALIAS OF *member*", where *member* is the member name for which *alias* is an alias.
- 13900. A SELECT statement that identifies an alias can specify only one name on the MEMBER operand.

Packaging Rules (JCLIN Copy Steps)

- 13910. If a ++MOD on a product tape defines either (1) a complete load module containing single or multiple CSECTs or (2) a partial load module containing multiple CSECTs, any ++MOD by the same name shipped in a subsequent PTF must also be the same type of load module (complete load module or multi-CSECT partial load module). If a CSECT shipped in the original ++MOD is not shipped in the replacement ++MOD, it will no longer exist.

To replace part of a copied ++MOD, the PTF must convert the ++MOD into a link-edited load module by splitting it into smaller serviceable parts, as follows:

1. Delete the original ++MOD with a ++MOD DELETE.
2. Ship a new ++MOD for each of the parts into which the original ++MOD has been split.
3. Provide link-edit JCLIN to define the link edit structure of the resulting load modules.

All future maintenance that affects the load module or any of its parts must explicitly or implicitly specify this PTF as a prerequisite.

Packaging Recommendations

Although JCLIN can be used to identify copied elements, the preferred way of copying elements other than ++MODs is to specify the DISTLIB and SYSLIB operands on the element MCS.

End of Packaging Recommendations

Notes:

1. Copy input must be **inline**, not pointing to another data set.
2. The only copy utility control statements allowed are COPY (or C) and SELECT (or S).

9.5.1 Considerations for the SELECT Statement for Copy Operations

When deciding whether to specify a SELECT statement in your copy steps, you need to consider how SMP/E processes copy steps:

- A COPY without SELECT MEMBER creates SMP/E DLIB entries.
- A COPY with SELECT MEMBER does not create the DLIB entries, but it either updates the SYSLIB subentry for MAC and SRC entries or builds MOD or LMOD entries (with the COPY indicator turned on) for modules.

Following are recommendations for IEBCOPY steps in product JCLIN:

- Do not use SELECT MEMBER statements for elements that are fully defined in the SMPMCS.

Data elements and hierarchical file system (HFS) elements must be fully defined in the SMPMCS.

- Use COPY statements with SELECT MEMBER statements for single-CSECT load modules that can be copied.
- Use COPY statements (without SELECT MEMBER statements) for each totally copied library.

9.5.1.1 Fully-Defined Elements

Copy steps are not required for fully-defined elements. Instead, the element statement should specify both DISTLIB and SYSLIB for all elements except ++MOD. Here are some examples:

```
++MAC(xxxxxxxx) DISTLIB(AMACLIB) SYSLIB(MACLIB) .
++SRC(xxxxxxxx) DISTLIB(AJES3SRC) SYSLIB(JES3SRC) .
++PROC(xxxxxxxx) DISTLIB(APROCLIB) SYSLIB(PROCLIB) .
++HFS(xxxxxxxx) DISTLIB(ABPXLIB) SYSLIB(BPXLIB) .
```

9.5.1.2 Single-CSECT Load Modules

Copy steps should be used for single-CSECT load modules that can be copied. Here is an example:

```
COPY INDD=ALINKLIB,OUTDD=LINKLIB TYPE=MOD
  SELECT MEMBER=xxxxxxxx
  SELECT MEMBER=yyyyyyyy
```

This statement indicates that SMP/E should build an LMOD entry with the same name as the module. In this LMOD entry, the COPY indicator should be turned on and the SYSLIB subentry should be LINKLIB.

9.5.1.3 Totally Copied Libraries

When no SELECT statement is specified for a copy step, SMP/E creates DLIB entries, which it uses to determine the appropriate target library (if none was specified on the element MCS and one didn't already exist). The DLIB entry indicates that the library specified in the INDD= parameter is totally copied to the library specified in the OUTDD= parameter. Here is an example:

```
COPY INDD=AMACLIB,OUTDD=MACLIB TYPE=MAC
```

This statement indicates to SMP/E that if an element being processed has a distribution library of AMACLIB but does not specify a SYSLIB of MACLIB—for example, ++MAC(xxxxxxxx) DISTLIB(AMACLIB)—SMP/E should install the macro in MACLIB and add the SYSLIB subentry of MACLIB to the macro entry.

Similar processing occurs for modules—except that the SYSLIB subentries are in LMOD entries (not MOD entries). Here is an example:

```
COPY INDD=ALINKLIB,OUTDD=LINKLIB
```

This statement indicates to SMP/E that if a module being processed has a distribution library of ALINKLIB but is not yet associated with a load module, SMP/E should build an LMOD entry with the same name as the module. In this LMOD entry the COPY indicator should be turned on and the SYSLIB subentry should be LINKLIB.

If you develop a new release of a function that uses totally copied libraries, and the new release copies the distribution library into a different target library from the previous release, you should instruct the users to delete the DLIB entry from the CSI before they apply or accept the new release. This ensures that when SMP/E installs the new release, it builds new DLIB entries pointing to only the new target library.

_____ End of Packaging Recommendations _____

9.6 Link-Edit Steps

Packaging Rules (JCLIN Link-Edit Steps)

- 14000. Link-edit steps must be identified by one of the following:
 - EXEC PGM=IEWL
 - EXEC PGM=HEWL
 - EXEC PGM=IEWBLINK
 - EXEC LINKS
- 14100. Link-edit steps must not be sensitive to the order of execution of other link-edit steps, either in the same FMID or in another FMID. Link-edit steps must also not be sensitive to the order of execution of the individual load module builds within the step.
- 14200. No elements to be included in a JCLIN link-edit step can be derived from the output of another JCLIN link-edit step, or from the output of a load module build within the same JCLIN link-edit step.
- 14210. Never specify a JCLIN link-edit step to indicate that a load module resides in the SMPLTS library.

SMP/E automatically link-edits a base version of any load module with a CALLLIBS subentry into the SMPLTS library.
- 14220. Do not specify a pathname in a hierarchical file system (HFS) as the distribution library.
- 14230. All INCLUDE statements in link-edit JCLIN data should specify the included module's distribution library, or SYSPUNCH if it is an assembled module. Do not use data sets such as SYSLIB or SYSLMOD.
- 14240. If a load module consists of more than one distribution library module, use an ENTRY statement; otherwise, the entry point of the load module might change each time the load module is relinked by SMP/E.
- 14250. If a specific order of CSECTs within a load module is required, use ORDER statements to define the load module structure.
- 14260. If Product A uses CALLLIBS to indicate libraries created by Product B:
 1. The SYSLIB DD statement in Product A's JCLIN must use the real DDNAME of the library.
 2. If Product A does not require Product B to exist in the same zone, Product A's DDDEF job must create a DDDEF entry for the library with its real DDNAME, using the ADD DDDEF command to avoid possible contamination of an existing DDDEF entry.
 3. If it is possible that the library does not exist on the system, the DDDEF job must instruct the customer to point either to the actual dataset (if it exists) or to an empty dataset. The job must not give the customer a choice of two or more legitimate datasets for one DDDEF. NOTE: The product may not allocate an empty dataset for this purpose.
- 14270. If a product documents in its installation documentation that a return code of 8 is acceptable from APPLY, then RC=8 must be coded on the NAME statement in the JCLIN for the appropriate load modules. This may be the case if the product uses a CALLLIBS library to obtain load modules created by an optional function.

SMP/E does not order the link-edit steps based on the order specified in the JCLIN. Instead, if multiple load modules and target libraries are involved, SMP/E organizes the link-edit steps for the most efficient invocations of the link-edit utility (which might not be the same order as the JCLIN data). For example, assume that a product consists of a base function and a dependent function.

- The dependent function conditionally coexists with the base function; it can be installed with the base function but is not a prerequisite for the base function.
- The base function must have its own JCLIN data that completely describes the elements it contains, because a user may choose to install the functions together or separately.

If the base function is separately installed, its JCLIN data cannot contain a link-edit step including elements from the dependent function, because those elements are not yet available.

Packaging Recommendations

- Product A should not INCLUDE modules created by Product B unless all of the following are true:
 1. Product B is guaranteed to always be present in the same target zone as Product A
 2. The module always exists in the same library, no matter which release of any product is present
 3. The library containing the module is always guaranteed to exist

If any of these conditions are not true, the product should use CALLLIBS instead of explicit INCLUDEs.

- The LMOD RC parameter should be specified on every JCLIN NAME statement. The value for each load module should match the expected return code from link-editing that load module, and the highest value within the JCLIN for an FMID should match the expected APPLY return code documented in the installation instructions for the product.
- If a product's JCLIN specifies INCLUDE AABCMODS(element), the product should REQ the product that installed the element into the library.

End of Packaging Recommendations

9.6.1 JCLIN Processing of DD Statements in Link-Edit Steps

Target libraries should be identified in link-edit steps by the SYSLMOD DD statement. All other DD statements for target libraries are ignored. JCLIN processing extracts the lowest-level qualifier from the data set name on the SYSLMOD DD statement, uses that qualifier as a ddname, and passes the link-edit utility a DD statement allocated to the data set with that ddname. For example, when JCLIN processing encounters this DD statement:

```
//SYSLMOD DD DSN=PROD1.SABCMOD1
```

it searches the target zone for a DDDEF entry with the name SABCMOD1. The data set name identified in that DDDEF entry is passed to the link-edit utility as the output (SYSLMOD) data set.

DD statements for distribution libraries are ignored by JCLIN processing. The ddnames specified on INCLUDE statements in the JCLIN are used as the DISTLIB value in the MOD entries that are created. For example, when JCLIN processing encounters this statement:

```
INCLUDE AABCLOAD(ABCMOD01)
```

it builds a MOD entry for ABCMOD01 and indicates a DISTLIB value of AABCLOAD. For more information about link-edit control statements, see 9.6.2, “Link-Edit Control Statements.” For details about JCLIN processing, see the *OS/390 SMP/E Commands* manual or the *SMP/E Release 8.1 Reference* manual.

9.6.2 Link-Edit Control Statements

All required link-edit control statements must be specified. This section describes considerations for specific link-edit statements. Here are some special considerations to keep in mind:

- If a load module consists of more than one distribution library module, use an ENTRY statement; otherwise, the entry point of the load module might change each time the load module is relinked by SMP/E.
- If a specific order of CSECTs within a load module is required, use ORDER statements to define the load module structure. See 8.5, “Enabling Load Module Changes at the CSECT Level (++MOD CSECT)” on page 84 for more information.
- If PLISTART is listed first in a PL/I load module, ORDER cards do not work; any requirement for ORDER should be changed to ENTRY.

ALIAS statement

To ensure that SMP/E can process your link-edit ALIAS control statements, you must address the following considerations:

- **General considerations**

- An ALIAS control statement can span any number of 80-byte records.

Note: If you assign a load module residing in a PDSE an alias value greater than eight characters, you cannot later use the ++DELETE statement to delete that alias value (and not the associated load module). 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.

- Column 1 of all 80-byte records composing an ALIAS control statement must contain a blank (X'40').
- The data for the first 80-byte record of an ALIAS control statement must start in column 2 or later and end anywhere up to and including column 71.
- The control statement type (ALIAS) must be followed by at least 1 blank (X'40').
- The control statement type (ALIAS) must be in uppercase.
- Columns 73 through 80 of an 80-byte record are ignored.

- An alias value can be from one to 64 characters.
 - An alias value can be composed of characters in the range X'41' through X'FE'.
- Note:** Although the binder also accepts characters X'0E' (shift-out character) and X'0F' (shift-in character), SMP/E does not accept them.
- An alias value can be enclosed in single apostrophes. It must be enclosed in single apostrophes in the following cases:

- It contains a character other than uppercase alphabetic, numeric, national (\$, #, or @), slash, plus, hyphen, period, and ampersand. Here is an example:

```

      1      2      3      4      5      6      7      8
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----0
ALIAS 'This_alias_contains_special_characters!!!!'
```

- It is continued to another 80-byte record of the control statement. Here is an example:

```

      1      2      3      4      5      6      7      8
-----+-----+-----+-----+-----+-----+-----+-----+-----0
ALIAS      'This_is_a_very_long_value_that_is_continued_to_the_next*_
_card!'
```

- If an apostrophe is part of the alias value (not a delimiter), two apostrophes need to be specified in the appropriate location in the alias value. These two apostrophes count as two characters in the 64-character limit for an alias value. Here is an example:

```

      1      2      3      4      5      6      7      8
-----+-----+-----+-----+-----+-----+-----+-----+-----0
ALIAS 'It''s_the_quote_that_makes_apostrophes_necessary.'
```

- The single apostrophes used to enclose an alias value do not count as part of the 64-character limit for an alias value. For example, the alias value in the following example contains 10 characters:

```

      1      2      3      4      5      6      7      8
-----+-----+-----+-----+-----+-----+-----+-----+-----0
ALIAS 'Only_ten!!'
```

- SMP/E uses the alias value exactly as is. SMP/E does not try to enforce any rules the binder may be using as a result of the CASE execution parameter.

Warning to Packagers

Be extremely careful when creating the JCL and link-edit ALIAS control statements to be processed by SMP/E as JCLIN. When parsing the ALIAS control statements to derive alias values, SMP/E does not try to replicate binder processing that would result from a particular specification of the CASE execution parameter. Therefore, you must ensure that the values on the ALIAS control statement are exactly as desired and that the proper CASE value is used so that the link-edit utility produces the desired results.

End of Warning to Packagers

• Continuation records

- Column 72 of a given 80-byte record must be a nonblank character if the control statement is continued onto the next 80-byte record. The character in column 72 denotes only continuation and is never part of an alias value.
- The data for continuation records (80-byte records 2 through *n* of an ALIAS control statement) can start in column 2 or later and end anywhere up to and including column 71 (for example, if multiple aliases are being specified).

The data for a continuation record **must** start in column 2 if it is part of an alias value that is being continued from the previous 80-byte record. An alias value that is continued from one 80-byte record to another 80-byte record must do all of the following:

- Be enclosed in single apostrophes
- Extend through column 71 of the first 80-byte record
- Start in column 2 of the next 80-byte record
- Have a nonblank continuation character in column 72

Here is an example:

```

      1      2      3      4      5      6      7      8
-----+-----+-----+-----+-----+-----+-----+-----+-----0
ALIAS      'This_is_a_very_long_value_that_is_continued_to_the_next*
_card!'
```

• **Entry points**

A new format of the ALIAS statement supported for the binder allows an alternative entry point to be specified into a load module. If this new format is used, each alias name with an associated entry point must be specified on its own 80-byte record, with a separate ALIAS statement; no other aliases should be specified on that statement. If multiple alias values of this format are specified on a single ALIAS control statement, only the first is recognized; the rest are ignored.

Note: When this form of the ALIAS control statement is used, the alias value cannot be 64 characters long, because SMP/E requires the statement to be complete on one 80-byte record. When this form of the ALIAS control statement is used, the maximum length for an alias value is 61 characters.

Suppose that a load module has the following aliases: ALA1, ALA2, ALA3, and ALA4. ALA1 and ALA2 are associated with entry point names ENTRYPT1 and ENTRYPT2, respectively.

- Here are examples of how the aliases should be specified:

```

      1      2      3      4      5      6      7      8
-----+-----+-----+-----+-----+-----+-----+-----+-----0
ALIAS ALA1(ENTRYPT1)
ALIAS ALA2(ENTRYPT2)
ALIAS ALA3
ALIAS ALA4
or
ALIAS ALA3,ALA4
```

- Here are examples of how the aliases should *not* be specified:

```

      1      2      3      4      5      6      7      8
----+---0---+---0---+---0---+---0---+---0---+---0---+---0---+---0
ALIAS ALA1(ENTRYPT1),ALA2(ENTRYPT2),ALA3,ALA4
or
ALIAS ALA1(ENTRYPT1),ALA3
ALIAS ALA2(ENTRYPT2),ALA4

```

• **Multiple aliases**

- Multiple alias values can be specified on a single ALIAS control statement as long as they are not in the form alias(entrypoint). Multiple alias values must be separated by commas (“,”). Here is an example:

```

      1      2      3      4      5      6      7      8
----+---0---+---0---+---0---+---0---+---0---+---0---+---0---+---0
ALIAS ALIAS1,ALIAS2,ALIAS3,ALIAS4

```

- Multiple alias values can span multiple 80-byte records. When this occurs, there must be a nonblank character in column 72, and one of the following must be true:

- The last alias value on the 80-byte record that is being continued must be followed by a comma and one or more blanks (“, ...”). Here is an example:

```

      1      2      3      4      5      6      7      8
----+---0---+---0---+---0---+---0---+---0---+---0---+---0---+---0
ALIAS ALIAS1,ALIAS2,
ALIAS3,ALIAS4

```

- The last alias value on the 80-byte record that is being continued must be followed by a comma (“,”) in column 71. Here is an example:

```

      1      2      3      4      5      6      7      8
----+---0---+---0---+---0---+---0---+---0---+---0---+---0---+---0
ALIAS ALIAS1,ALIAS2,'A_relatively_long_ALIAS_but_not_quite_64_chars.',*
ALIAS4,ALIAS5

```

- The last alias value on the 80-byte record that is being continued can be enclosed in single apostrophes such that part of the alias value appears on the current 80-byte record and part appears on the next 80-byte record (see the rules for continuation records). Here is an example:

```

      1      2      3      4      5      6      7      8
----+---0---+---0---+---0---+---0---+---0---+---0---+---0---+---0
ALIAS ALIAS1,ALIAS2,'A_relatively_long_ALIAS.',ALIAS4,'Not_too_long_bu*
t_wraps.',ALIAS5,ALIAS6

```

- If a blank (X'40') follows an alias value, SMP/E assumes there are no more alias values for the current ALIAS control statement.

CHANGE statement

IBM Software Delivery Restrictions

For the **IBM Software Delivery Solutions** process to operate correctly, a product must not contain a CHANGE statement in a link-edit step.

End of IBM Software Delivery Restrictions

ENTRY statement

Each load module consisting of more than one distribution library module must have an ENTRY statement; otherwise, the entry point of the load module changes each time the load module is relinked by SMP/E.

EXPAND statement

Packaging Recommendations

EXPAND statements should not be used in JCLIN data, because they would be saved in the LMOD entry and would cause the load module to be expanded each time it is link-edited. This is not always desirable.

End of Packaging Recommendations

IDENTIFY statement

Packaging Recommendations

IDENTIFY statements should not be used in JCLIN data. They are produced as part of servicing a module. If found in the JCLIN, they are stored in the LMOD entry and can result in incorrect data being stored during the application of service.

End of Packaging Recommendations

INCLUDE *ddname(member,member...)* statement

INCLUDE statements are used to identify the modules in the load module. They are also used to identify utility input to be included when the load module is link-edited. This is denoted by the TYPE comment on the INCLUDE statement. The format of the TYPE comment on the INCLUDE statement is:

INCLUDE *ddname(member,member...)* TYPE=UTIN

There must be at least one blank between the closing parenthesis of the INCLUDE statement and the TYPE comment. If the TYPE comment is not specified, SMP/E assumes the INCLUDE statement identifies modules.

Processing Modules:

Each module in the load module must be specified as a member name on an INCLUDE statement.

The member names are assumed to be modules existing in distribution library *ddname*. SMP/E builds MOD entries for each member name specified and sets the DISTLIB value in each MOD entry to *ddname*. (An exception to this is when the *ddname* is SYSLMOD. In that case, no MOD entry is built for the INCLUDE statement.) SMP/E does not refer to the *ddname* DD statement to determine the actual library referred to. Therefore, all *ddnames* specified on INCLUDE statements must be the actual *ddnames* assigned to the products.

The INCLUDE statements are not saved in the LMOD entry, because they are not necessary when the load module is link-edited. All link-edits requested by SMP/E are CSECT-replaces; the load module is built from the new version of the updated CSECT and the existing load module from the target library.

The *ddnames* *SYSPUNCH* and *SMPOBJ* are reserved for inclusions of object decks produced by assembly steps that are not to be link-edited to a distribution library during ACCEPT processing. In both cases, the name stored in the MOD entry's DISTLIB subentry is SYSPUNCH.

Processing Utility Input:

Each utility input file must be specified as a member on an INCLUDE statement with the TYPE=UTIN comment.

The member names are assumed to be members of the library *ddname*. SMP/E builds a utility input subentry for each member name specified and stores it into the LMOD entry. Each utility input subentry contains the member name and the *ddname*. The utility input files will be included when link-editing the load module. These files may identify definition side decks containing link-edit control statements, or any other file to be included during a link-edit operation.

Note: For a product including modules not provided by that product, the INCLUDE statements must specify either a distribution library *ddname* or SYSPUNCH. (SYSPUNCH is used only for processing assembled modules.) If your product will be installed in the same target and distribution zones as the other product, see 9.6.4, "Cross-Product Load Modules for Products Installed in the Same Zone" on page 107 for more information. If you cannot be sure, or if you know that your product will be installed in different target and distribution zones from the other product, see 9.6.5, "Cross-Product Load Modules for Products Installed in Different Zones" on page 109 for more information.

INSERT and OVERLAY statements

If a load module is to be linked in overlay structure, you must supply an INSERT control statement for each CSECT in the load module, including INSERT statements for those CSECTs within the root segment. It is not sufficient to properly place the INCLUDE and OVERLAY control statements.

LIBRARY statement

Normally, LIBRARY statements should not be used in JCLIN data. An exception is when the CALLLIBS operand is specified on the JCLIN command or ++JCLIN MCS, or when //*CALLLIBS=YES is encountered after a job card preceding a link-edit step. JCLIN processing then allows for the LIBRARY state-

ment to be used to specify those modules (external references) that are to be excluded from the automatic library search during the following:

- The current linkage editor job step (restricted no-call function)
- Any subsequent linkage editor job step (never-call function)

A LIBRARY statement should be used only if a SYSLIB DD statement is also used. It should not be used to specify additional automatic call libraries; the SYSLIB DD statement should be used instead.

NAME *Imodname*(R) statement

When SMP/E encounters either the NAME control statement or the end of input with no NAME statement, SMP/E builds an LMOD entry. How SMP/E determines the name of the LMOD depends on the JCL being scanned:

- If the NAME statement is found, SMP/E gets the LMOD name from the *Imodname* field of the NAME statement.
- If no NAME statement is found and a SYSLMOD DD statement is present, SMP/E gets the LMOD name from the member name of the data set specified. If no member name is specified, SMP/E issues an error message identifying the JOBNAME and STEPNAME and the reason for the error.
- If no NAME and SYSLMOD DD statements are found, SMP/E searches for the MOD=*name* operand in the JCL and uses that name as the LMOD name. If no MOD=*name* operand is found, SMP/E issues an error message.

IBM Software Delivery Restrictions

A product that uses CALLLIBS must not use the RC= parameter on a NAME statement in its JCLIN unless one of the following is true:

1. The product using CALLLIBS identifies the product providing the CALLLIBS libraries as a requisite (REQ) in its SMPMCS.
2. The product using CALLLIBS ships stubs for the elements linked with CALLLIBS.

This is a restriction of the **IBM Software Delivery Solutions** process; the CALLLIBS libraries may not be available during order build processing unless one of the above criteria is met.

End of IBM Software Delivery Restrictions

ORDER statement

If a specific order of CSECTs within a load module is necessary, ORDER statements are required to define the load module structure. Simply ordering the INCLUDE statements is not sufficient, because SMP/E does CSECT replacements when relinking the load module and, therefore, changes the order of the CSECTs.

REPLACE statement

REPLACE statements are saved in the LMOD entry and are associated with the DLIB module name found on the next INCLUDE statement in the JCL. If the same INCLUDE statement is processed later by JCLIN, REPLACE state-

ments already in the LMOD entry associated with this INCLUDE statement are deleted and replaced by any associated REPLACE statements in the latest job. REPLACE statements are passed to the linkage editor only when the associated DLIB module is to be replaced in the load module.

SYSDEFSD DD statement

SMP/E uses the SYSDEFSD DD statement to determine the side deck library for the load module. SMP/E determines the ddname by using the lowest-level qualifier of the data set name specified in the SYSDEFSD DD statement. This ddname is saved as the side deck library subentry in the LMOD entry.

The side deck library will contain the definition side deck for the load module created by the link-edit utility. The definition side deck contains link-edit control statements describing the load module.

SYSLIB DD statement

Normally, SYSLIB DD statements should not be used in JCLIN data. However, they can be used for load modules needing to implicitly include modules from other products. Such load modules are commonly used by products that:

- Are written in a high-level language and, as a result, include modules from libraries (such as compiler libraries) that are owned by a different product
- Make use of a callable-services interface provided by another product
- Need to include stub routines or interface modules from different products that may reside in other zones

For such load modules, the SYSLIB DD statement should specify all the automatic call libraries SMP/E is to use when linking the load module. (These libraries should be target libraries.) The low-level qualifier of each data set specified in the SYSLIB concatenation is saved as a CALLLIBS subentry for the associated load module. For SMP/E to link implicitly-included modules from these libraries, the user must provide DDDEF entries for the libraries in the zone containing the LMOD entry.

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.

Implementation Notes:

- It is best to use this SYSLIB support when introducing a new version or release of your product, or when introducing a new load module for an existing version or release of your product.

Using this SYSLIB support for an existing load module in a current product is not recommended. However, if you need to make such a change, make sure to do the following in the SYSMOD introducing the change:

1. Supply a JCLIN link-edit step to redefine the load module. This step must specify the SYSLIB allocation needed for the load module.
2. Specify the CALLLIBS operand on the ++JCLIN statement to ensure that the SYSLIB DD statement is processed.
3. Supply all the modules that are explicitly included in the JCLIN link-edit step and that are owned either by this SYSMOD or by its FMID.

Modules that are explicitly included in the JCLIN link-edit step and that are not owned by this SYSMOD or its FMID are included by SMP/E through normal load module build processing.

If the existing load module had included cross-zone modules through the use of the LINK command, those modules are no longer included in the load module after the installation of the SYSMOD that redefined the load module. In this case, SMP/E issues warning messages. After the installation of the SYSMOD, the user must rerun the LINK command to include those cross-zone modules back into the load module. For more information about the LINK command, see the *OS/390 SMP/E Commands* manual or the *SMP/E Release 8.1 Reference* manual.

- When a load module is built using SYSLIB DD statements, SMP/E cannot completely service the load module because it does not know the content of the load module. Specifically, the load module is not automatically rebuilt when an implicitly-included module is serviced. However, users can run the REPORT CALLLIBS command to identify and relink such load modules. For more information about the REPORT CALLLIBS command, the *OS/390 SMP/E Commands* manual or the *SMP/E Release 8.1 Reference* manual.

Including Pathnames in a SYSLIB Concatenation: A DD statement in a SYSLIB concatenation can include the PATH operand to specify a pathname as an automatic call library. A LIBRARYDD comment statement must immediately follow this DD statement and specify the ddname to be associated with that pathname. SMP/E saves the ddname specified on the LIBRARYDD comment as part of the CALLLIBS list in the LMOD entry being updated or created. For an example, see 9.7.5, “JCLIN Data for Load Modules Residing in a Hierarchical File System” on page 119.

Notes:

1. If a DD statement in the concatenation comes between the DD statement specifying the PATH operand and the LIBRARYDD comment, the misplaced DD statement is ignored.
2. If the DD statement specifying the PATH operand is followed by a JCL statement other than a LIBRARYDD comment or a continuation DD statement for the SYSLIB concatenation, the LMOD entry is not updated or created. In addition, if the JCLIN was specified in a SYSMOD (instead of being processed by the JCLIN command), processing for that SYSMOD fails.

SYSLMOD DD statement

SMP/E uses either the SYSLMOD DD statement or the NAME statement to determine the target library for the load module, as follows:

- If a SYSLMOD DD statement is present, SMP/E determines the target library ddname by using the lowest-level qualifier of the data set name specified in the SYSLMOD DD statement.
- If no SYSLMOD DD statement is present, SMP/E determines the name by looking at the NAME=*dsname* option on the procedure statement. The ddname used is the lowest-level qualifier of the data set name specified in the NAME option.

- If no SYSLMOD DD statement or NAME=*dsname* value is found, SMP/E issues an error message.

The ddname of the target library is saved as the SYSLIB value in the LMOD entry for the load module.

A SYSLMOD DD statement can include the PATH operand to specify a pathname for installing a load module in a hierarchical file system. A LIBRARYDD comment statement must immediately follow this DD statement and specify the ddname to be associated with that pathname. SMP/E saves the ddname specified on the LIBRARYDD comment as a SYSLIB subentry in the LMOD entry being updated or created. For an example, see 9.7.5, “JCLIN Data for Load Modules Residing in a Hierarchical File System” on page 119.

Notes:

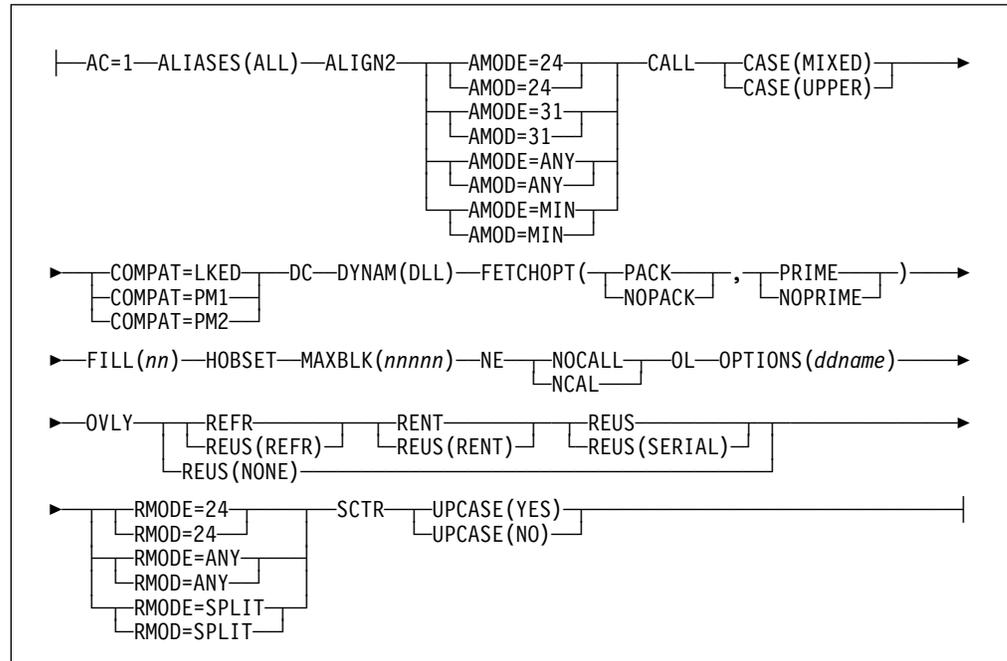
1. If the DD statement specifying the PATH operand is followed by a JCL statement other than a LIBRARYDD comment, the LMOD entry is not updated or created. In addition, if the JCLIN was specified in a SYSMOD (instead of being processed by the JCLIN command), processing for that SYSMOD fails.
2. An LMOD entry can have at most two SYSLIB subentries. If the LMOD entry already contains two SYSLIB subentries, SMP/E replaces the second SYSLIB ddname with the ddname found on the SYSLMOD DD statement, the NAME=*dsname* option, or the LIBRARYDD comment statement.

All other statements found in link-edit input

All other link-edit control statements found are saved in the LMOD entry in the order they are encountered, and are passed to the linkage editor whenever SMP/E needs to relink this load module.

9.6.3 Link-Edit Attribute Parameters

These are the link-edit attributes SMP/E recognizes in the PARM field and saves for future processing:



When none of the above attributes are found, the STD indicator is set in the LMOD entry to indicate that the load module should be link-edited without any particular attributes.

Notes:

1. The OPTIONS attribute is recognized and processed, but it is not saved as part of the LMOD entry or the MOD entry being processed. It is used as a pointer to an imbedded file containing additional option specifications, allowing the PARM string to exceed the 100-character limit.
2. For more information on which attributes you can use with a specific link-edit utility, see the reference manual for that utility.

9.6.4 Cross-Product Load Modules for Products Installed in the Same Zone

There are two basic reasons for products to require cross-product load modules:

- A load module for one function SYSMOD needs to include a module from another function SYSMOD.
- A function SYSMOD needs to include some of its own modules in a load module of another function SYSMOD.

9.6.4.1 Linking a Module from Another Function

If a load module for Product A needs to include a module from Product B, a link-edit step in the JCLIN data for Product A must do one of the following:

- **Explicitly define the modules:** To explicitly define Product B modules to be included, the INCLUDE statement must be used.
- **Implicitly define the modules:** To implicitly define Product B modules to be included, the SYSLIB statement must be used. (The LIBRARY statement can also be used, if any specific modules should not be included.)

For more information on this method, see 9.6.5.2, “Implicitly Defining the Modules” on page 110.

Table 16 briefly compares the two methods.

<i>Table 16. Comparison of Explicit versus Implicit Definition of Modules</i>		
Consideration	Explicit Definition	Implicit Definition
Modules are automatically serviced.	X	
Modules do not need to be specified individually.		X

The function SYSMOD for Product A will not contain a ++MOD statement for the Product B module. If the Product B module is installed and the Product A load module does not already exist, the module is automatically included in the Product A load module.

If a load module for your product (Product A) requires a module from another product, you should describe this in the installation documentation for Product A and mention any additional jobs the user should run.

If the module from Product B is deleted (such as if a new replacement release of Product B is installed), SMP/E keeps a record of the fact that the module had been a part of a load module in Product A. As a result, if the module is reintroduced by Product B (such as in the replacement release of the product), SMP/E automatically relinks the load module from Product A to include the module from Product B. If the module is not reintroduced but is still required in Product A, and a copy of the module is still available on the system, the user must use the JCLIN command to reprocess the JCLIN data for Product A and then rerun the postinstallation link-edit job.

On the other hand, a new release of Product A might delete the previous release of Product A and redefine the load module without including any of the borrowed modules. As part of installing the new release of Product A, SMP/E will first delete the old Product A modules from the load module, leaving a copy of the load module consisting solely of modules borrowed from other products. SMP/E will then use this copy of the load module (with the borrowed modules) as input when rebuilding the load module for the new release of Product A. To ensure that the new version of the load module does not include the borrowed modules, the new release of Product A must contain a ++DELETE MCS for the load module (to delete the previous version) in addition to the JCLIN needed to rebuild the new version of the load module.

9.6.4.2 Linking Modules into a Load Module for Another Function

If a function SYSMOD(1) needs to include any of its own modules in a load module of another function SYSMOD(2), you have two packaging options:

- If the load module already exists, and no link-edit control statements must be added or changed to add the modules to the load module, the ++MOD MCS for each module can specify the load module on the LMOD operand.

- If the load module does not exist, or if any link-edit control statements must be added or changed to add the module to the load module, the JCLIN data for function SYSMOD(1) must specify an INCLUDE statement for each of those modules followed by an INCLUDE SYSLMOD statement for the load module of function SYSMOD(2).

You can use these techniques to include a module for a dependent function in a load module for its parent base function, or to include a module for Product A in a load module for Product B. However, Product B must be installed before Product A.

A new release of Product A might no longer need to include its modules in a load module for Product B. However, because the new release of Product A deletes the previous release, SMP/E updates the LMOD entry for the Product B load module to track the modules that were deleted. As a result, if no action is taken, SMP/E relinks the Product A modules into the Product B load module when the new release of Product A is installed. You must make sure the installation documentation for your product tells the user how to avoid this problem. Here are the steps you need to describe:

1. Build a dummy function to delete Product A. (For an example, see the recommendations under 7.2.4, “Deleting SYSMODs (DELETE)” on page 57.)
2. Use UCLIN to remove the MODDEL subentries for the Product A modules from the Product B LMOD entry.
3. Install the new release of Product B.

9.6.5 Cross-Product Load Modules for Products Installed in Different Zones

Cross-product, cross-zone load modules can be created through one of the following methods:

- SMP/E LINK command (done after installation)
- Implicitly defining the modules (done in the JCLIN link-edit step)

Table 17 briefly compares the two methods. It is followed by more information about each method.

Consideration	SMP/E LINK Command	Implicit Definition
Good for products written in high-level languages or that use callable services.		X
Load modules can be automatically serviced.	X	
Modules do not need to be specified individually.		X
SMP/E tracks the cross-zone relationship.	X	

9.6.5.1 SMP/E LINK Command

This method is best when a load module needs to include a few specific modules from another product. To define the cross-zone relationship and create the cross-zone load modules, the LINK command and UCL statements are run by the user. No JCL statements are needed to add the modules to the cross-zone load modules.

If a load module for your product requires a module from another product that is likely to be installed in a different zone, you should describe this in the installation documentation for your product and describe the SMP/E LINK commands the user should run.

When this method is used, SMP/E tracks the cross-zone relationship between the load modules and modules. As a result, cross-zone processing for subsequent APPLY and RESTORE commands can automatically maintain the affected load modules.

The LINK command requires the modules it processes to be stand-alone modules. No assemblies are done by either the LINK command or by cross-zone processing for APPLY and RESTORE commands. Therefore, when packaging a module that you intend to be used as input to the LINK command, make sure it is installed in its target library as either a single-CSECT load module or as part of a totally copied library.

Note: There are times when the LINK command is not appropriate to use—generally, for products that are written in a high-level language and, as a result, include modules from libraries (such as compiler libraries) owned by a different product. In these cases, you should use SYSLIB DD statements to implicitly include the modules. For more information on when to use a SYSLIB DD statement, see 9.6.5.2, “Implicitly Defining the Modules” and the description of the SYSLIB DD statement under 9.6.2, “Link-Edit Control Statements” on page 97.

For an example of using the LINK command, see the *SMP/E User's Guide*. For details on the LINK and UCL commands, see the *OS/390 SMP/E Commands* manual or the *SMP/E Release 8.1 Reference* manual.

9.6.5.2 Implicitly Defining the Modules

This method is best used when a load module must include many modules from other products, and it is difficult and error-prone (and perhaps impossible) to define all the modules to be included.

To implicitly define modules to be included from another product, the SYSLIB statement must be used. (The LIBRARY statement can also be used, if any specific modules should not be included.) Inform the user that the libraries containing the modules must be defined by DDDEF entries in the zone for the product that is including the modules.

Unlike the LINK command, when this method is used, SMP/E does not track the cross-zone relationship between the load modules and modules. However, after a library specified in the SYSLIB DD statement has been updated, the REPORT CALLLIBS command can be used to identify and relink load modules that define a SYSLIB statement. For more information about the REPORT CALLLIBS command,

see the *OS/390 SMP/E Commands* manual or the *SMP/E Release 8.1 Reference* manual.

For more information on using the SYSLIB DD statement, see the description of that statement under 9.6.2, “Link-Edit Control Statements” on page 97.

9.6.6 Adding or Changing Load Modules in a PTF

If a PTF needs to add a new load module or change the structure of an existing load module, use the techniques listed below.

- **Adding a new module to an existing load module.** The PTF must ship all of the following:
 - Inline JCLIN describing the new load module structure
 - A ++MOD statement for the new module being added

When SMP/E installs the PTF, it updates the entries, then performs the link. As a result, the new module is included in the link, and the old load module is replaced.

- **Creating a new load module.** The PTF must ship all of the following:
 - Inline JCLIN describing the new load module structure
 - All of the modules (other than those from other products) that are part of the load module
 - A ++MOD statement for each of those modules (except those from other products)

When SMP/E installs the PTF, it updates the entries, then performs the link. As a result, the new load module is added to the target library.

- **Deleting a module from a load module (and from a product).** The PTF must ship all of the following:
 - Inline JCLIN describing the new load module structure
 - A ++MOD DELETE statement for the module being deleted

When SMP/E installs the PTF, it delinks the module from the load module, then removes all references to the deleted module.

- **Deleting a module from a load module (but leaving the module in the product).** The PTF must ship all of the following:
 - Inline JCLIN describing the new load module structure
 - A ++DELETE statement for the load module
 - All the modules (other than those from other products) that are still part of the load module
 - A ++MOD statement for each of those modules (except those from other products)

When SMP/E installs the PTF, it deletes the load module from the target library, updates the appropriate entries with the new load module definition, and relinks the load module using the modules from the PTF.

Note: In each of these cases, the PTF must contain a ++MOD statement for each module being added or deleted. If the PTF does not contain the ++MOD statement, SMP/E updates the entries but does not invoke the link-edit utility.

9.7 Examples of JCLIN Data

This section shows examples of JCLIN data to define the following:

- Copy, assembler, and link-edit steps for modules
- Copy steps for macros or source
- Assembler steps to create modules from source
- Link-edit steps that use the automatic library call function
- Link-edit steps for load modules residing in a hierarchical file system

9.7.1 JCLIN Data for Modules

The following are some sample job steps for providing SMP/E with the information it needs to copy, assemble, and link-edit modules.

```

/*****/
/*
/* Step C1 informs SMP/E that an entire distribution library
/* is copied to a target library. From the INDD operand SMP/E
/* determines the ddname of the distribution library (AMACLIB), and
/* from the OUTDD operand SMP/E determines the ddname of the target
/* library (MACLIB). SMP/E will use this information to determine
/* the target library for subsequent changes that specify an
/* element's distribution library as AMACLIB.
/*
/*
/* Although the copy step can be performed using JCLIN, the
/* preferred method is to specify the copy in the MCS.
/*
/*****/
//C1      EXEC PGM=IEBCOPY
//AMACLIB DD DSN=SYS1.AMACLIB,DISP=SHR
//MACLIB  DD DSN=SYS1.MACLIB,DISP=SHR
//SYSIN DD *
        COPY INDD=AMACLIB,OUTDD=MACLIB  TYPE=MAC

/*****/
/*
/* Step C2 shows elements that are selectively copied from a
/* distribution library to a target library. The module name,
/* JZZLMO DC, is defined by the SELECT MEMBER statement. The load
/* module name, JZZLMO DC, is simply the module name. The INDD
/* statement defines the distribution library as AOS14, and the OUTDD
/* statement defines the target library as LINKLIB. When the JCLIN
/* data is processed, SMP/E sets an indicator (COPY)--the COPY
/* indicator means that when the module is link-edited, the link-edit
/* attributes must be obtained by examining the target library.
/*
/*****/
//C2      EXEC PGM=IEBCOPY
//AOS14   DD DSN=SYS1.AOS14,DISP=SHR
//LINKLIB DD DSN=SYS1.LINKLIB,DISP=SHR
//SYSIN DD *
        COPY INDD=AOS14,OUTDD=LINKLIB  TYPE=MOD
        SELECT MEMBER=((JZZLMO DC,,R))
/*

```

```
/*
/* *****
/* Step A1 defines an assembled module named JZZAMOD1. The module
/* name is specified as the member name on the SYSPUNCH DD statement.
/*
/* It also defines a macro named JZZAMAC1. SMP/E will detect the
/* invocation of the macro in the assembler SYSIN data.
/*
/* NOTE: This method is used to introduce a new element, not to
/* service an existing element.
/*
/* This example should be used ONLY for supplying inline
/* assembler source, and should NOT be used for elements
/* shipped with ++SRC or ++MOD statements. The MOD entry
/* resulting from this technique will contain a DISTLIB of
/* SYSPUNCH, which might not be desirable if a ++MOD statement
/* is shipped and the element is installed in a real
/* distribution library.
/* *****
//A1 EXEC PGM=ASMA90
//SYSLIB DD DSN=SYS1.AMACLIB,DISP=SHR
//SYSPUNCH DD DSN=&&PUNCH(JZZAMOD1),
// SPACE=(TRK,(1,1,1)),DISP=(,PASS)
//SYSIN DD *
JZZAMOD1 CSECT
        JZZAMAC1 --- INVOKE MACRO
        END JZZAMOD1
/*
```

```

/*****/
/*
/* Step L1 shows how to link-edit the previous assembly. The
/* link-edit INCLUDE statement defines module JZZAMOD1. The module
/* name is determined from the member name operand on the INCLUDE
/* statement, and the distribution library, SYSPUNCH, is determined
/* from the INCLUDE statement's ddname.
/*
/*
/* Step L1 also defines a load module and its target library.
/* Load modules JZZLMOD1 is defined by the link-edit NAME
/* statement. The ddname of the target library, LPALIB, is defined
/* by SYSLMOD DD statement. The load module attribute RENT is saved
/* for use in subsequent link-edits of this load module; the
/* parameters LET and LIST are not saved.
/*
/*
/* NOTE: This method is used to introduce a new element, not to
/* service an existing element.
/*
/*
/* This example should be used ONLY for supplying inline
/* assembler source, and should NOT be used for elements
/* shipped with ++SRC or ++MOD statements. The MOD entry
/* resulting from this technique will contain a DISTLIB of
/* SYSPUNCH, which might not be desirable if a ++MOD statement
/* is shipped and the element is installed in a real
/* distribution library.
/*
/*
/*****/
//L1 EXEC PGM=IEWL,PARM='LET,LIST,NCAL,RENT'
//SYSLMOD DD DSN=SYS1.LPALIB,DISP=SHR
//SYSPUNCH DD *.A1.SYSPUNCH,DISP=(SHR,PASS)
//SYSLIN DD *
INCLUDE SYSPUNCH(JZZAMOD1)
NAME JZZLMOD1(R)
/*

/*****/
/*
/* Step L2 defines two modules and one load module
/* to SMP/E. Modules JZZAMOD2 and JZZAMOD3 are defined by the
/* link-edit INCLUDE statements; the distribution library for each of
/* these is defined as AOS12. The load module is defined as
/* JZZLMOD2, with LINKLIB as the target library. The parameters LET
/* and LIST are not saved.
/*
/*
/*****/
//L2 EXEC PGM=IEWL,PARM='LET,LIST,NCAL'
//SYSLMOD DD DSN=SYS1.LINKLIB,DISP=SHR
//AOS12 DD DSN=SYS1.AOS12,DISP=(SHR,PASS)
//SYSLIN DD *
INCLUDE AOS12(JZZAMOD2)
INCLUDE AOS12(JZZAMOD3)
ENTRY JZZAMOD2
NAME JZZLMOD2(R)
/*

```

```

/*****/
/*
/* Step L3 shows an example of using the OPTIONS option.
/* The OPTNAME DD statement allows SMP/E to process the PARM string
/* even though the options exceed the 100-character limit.
/*
/*
/*****/
//L3      EXEC PGM=IEWBLINK,PARM='OL,AMODE=31,...,OPTIONS(OPTNAME)'
//SYSLMOD DD DSN=SYS1.LINKLIB,DISP=SHR
//AOS12   DD DSN=SYS1.AOS12,DISP=SHR
//OPTNAME DD *
          FETCHOPT(PACK,PRIME),RMODE=24
          MAXBLK(256)
/*
//SYSLIN DD *
          INCLUDE AOS12(GIMMPDRV,GIMMPDR1,...)
          ENTRY GIMMPDRV
          SETCODE AC(1)
          NAME GIMMPP(R)
/*

/*****/
/*
/* Step L4 shows an example of using a SYSLIB concatenation in a
/* link-edit step to implicitly include modules from libraries for
/* other products.
/*
/*
/* Modules MOD00004 and MOD00005 are defined by a link-edit INCLUDE
/* statement; the distribution library for each of these modules is
/* defined as AOS12. The load module is defined as LMOD04, with
/* APPLoad as the target library. If the CALLLIBS operand is
/* specified on the JCLIN command or ++JCLIN MCS, the low-level
/* qualifiers of the data sets specified in the SYSLIB concatenation
/* (PLIBASE and APPBASE) are saved as CALLLIBS subentries in the
/* LMOD entry for LMOD04.
/*
/*
/*****/
//L4      EXEC PGM=IEWBLINK,PARM='CALL,RENT,REUS'
//SYSLMOD DD DSN=SYS1.APPLoad,DISP=SHR
//AOS12   DD DSN=SYS1.AOS12,DISP=SHR
//SYSLIB  DD DSN=SYS1.PLIBASE,DISP=SHR
//        DD DSN=SYS1.APPBASE,DISP=SHR
//SYSLIN DD *
          INCLUDE AOS12(MOD00004,MOD00005)
          ENTRY MOD00004
          SETCODE AC(1)
          NAME LMOD04(R)
/*

```

9.7.2 JCLIN Data for Macros and Source

Here is a sample job step for providing SMP/E with the information it needs to copy macros and source:

```
//STEP1 EXEC PGM=IEBCOPY
//AMACLIB DD DSN=SYS1.AMACLIB,DISP=SHR
//MACLIB DD DSN=SYS1.MACLIB,DISP=SHR
//AJZSRC DD DSN=SYS1.AJZSRC,DISP=SHR
//JZSRC DD DSN=SYS1.JZSRC,DISP=SHR
//SYSIN DD *
COPY INDD=AMACLIB,OUTDD=MACLIB TYPE=MAC
S M=(MAC01,MAC02,MAC03)
S M=(MAC11) ALIAS OF MAC01
COPY INDD=AJZSRC,OUTDD=JZSRC TYPE=SRC
S M=(SRC04,SRC05)
/*
```

This JCLIN data defines the following to SMP/E:

Element Type	Element Name	Distribution Library	Target Library
Macro	MAC01	AMACLIB	MACLIB
Macro	MAC02	AMACLIB	MACLIB
Macro	MAC03	AMACLIB	MACLIB
Macro	MAC11 (alias)	AMACLIB	MACLIB
Source	SRC04	AJZSRC	JZSRC
Source	SRC05	AJZSRC	JZSRC

Remember, if the element is fully defined (both the DISTLIB and the SYSLIB are specified on the element MCS), this JCLIN data is not needed.

9.7.3 JCLIN Data for an Assembler Step to Create a Module from Source

Here is a sample job step for providing SMP/E with the information it needs to create a module by assembling source:

```
//STEP1 EXEC PGM=ASMA90
//SYSLIB DD DSN=SYS1.MACLIB,DISP=SHR
//SYSPUNCH DD DSN=&&PUNCH(SRCA),DISP=SHR
//SYSIN DD DSN=SYS1.AJZSRC(SRCA),DISP=SHR
```

This defines a source module named SRCA, which resides in distribution library AJZSRC.

9.7.4 JCLIN for Using the Link-Edit Automatic Library Call Function

SMP/E provides support for load modules that need to use the link-edit automatic library call function, which enables the load modules to contain modules from multiple products without explicitly specifying those modules on INCLUDE statements in link-edit steps. SMP/E's support for load modules that use the link-edit automatic library call function is called *CALLLIBS support*.

9.7.4.1 Overview of CALLLIBS Support

SMP/E's CALLLIBS support uses the link-edit CALL parameter and a SYSLIB allocation when invoking the link-edit utility to resolve external references in load modules. CALLLIBS support can be useful for a variety of products, including those that:

- Are written in a high-level language and, as a result, include modules from libraries (such as compiler libraries) that are owned by a different product
- Make use of a callable-services interface provided by another product
- Need to include stub routines or interface modules from different products that may reside in other zones

To package a load module that needs to use the automatic library call function, follow these steps:

1. Specify the CALLLIBS operand on the ++JCLIN MCS. CALLLIBS tells SMP/E to:
 - Save the SYSLIB allocation defined by the JCLIN link-edit step in the LMOD entry for the load module. This information is recorded in the new CALLLIBS subentry list.
 - Pass the SYSLIB allocation and the CALL parameter to the link-edit utility for linking the load module.

Here is an example of the ++JCLIN MCS:

```
++JCLIN ... CALLLIBS.
```

Note: If CALLLIBS is not specified, the SYSLIB allocation in the link-edit step is ignored and the NCAL parameter is used when invoking the link-edit utility.

2. Provide link-edit JCLIN that defines the SYSLIB allocation for the libraries containing the modules to be implicitly included by the link-edit automatic library call function.

SMP/E will save the low-level qualifiers of the data sets in the SYSLIB allocation as a CALLLIBS subentry list in the LMOD entry for the load module.

Here is an example of link-edit JCLIN that defines a SYSLIB allocation for a load module that needs to use the link-edit automatic library call function.

```
//STEP1 EXEC PGM=IEWBLINK,PARM='RENT,REUS'
//SYSLMOD DD DSN=SYS1.APPLoad,DISP=OLD
//AOS12 DD DSN=SYS1.AOS12,DISP=SHR
//SYSLIB DD DSN=SYS1.PLIBASE,DISP=SHR
// DD DSN=SYS1.APPBASE,DISP=SHR
//SYSLIN DD *
    INCLUDE AOS12(MOD00001,MOD00002)
    ENTRY MOD00001
    SETCODE AC(1)
    NAME LMOD01(R)
/*
```

3. Inform your users of special requirements for installing the SYSMOD.

- Before installing the SYSMOD, users must define DDDEF entries in the target zone that will be used to apply the SYSMOD. DDDEF entries are required for:
 - Each of the data sets in the load module's SYSLIB allocation
 - The SMPLTS data set, which is used to link the implicitly-included modules into the load module

9.7.4.2 Example of a SYSMOD That Implements CALLLIBS Support

The following is a part of a sample function SYSMOD with a load module that needs to use the link-edit automatic library call function. The numbers associate items in the SYSMOD with the steps listed in 9.7.4.1, "Overview of CALLLIBS Support" on page 117.

```
++FUNCTION(HXY1100) FILES(3).
++VER(Z038).
1 ++JCLIN CALLLIBS.
...
//STEP1 EXEC PGM=IEWBLINK,PARM='RENT,REUS'
//SYSLMOD DD DSN=SYS1.APPLoad,DISP=OLD
//AOS12 DD DSN=SYS1.AOS12,DISP=SHR
2 //SYSLIB DD DSN=SYS1.PLIBASE,DISP=SHR
// DD DSN=SYS1.APPBASE,DISP=SHR
//SYSLIN DD *
    INCLUDE AOS12(MOD00001,MOD00002)
    ENTRY MOD00001
    SETCODE AC(1)
    NAME LMOD01(R)
/*
...
++MOD(MOD00001) RELFILE(2) DISTLIB(AOS12).
++MOD(MOD00002) RELFILE(2) DISTLIB(AOS12).
...

```

The user needs to define DDDEF entries for the data sets specified in the SYSLIB allocation (PLIBASE and APPBASE), as well as for the SMPLTS data set, which SMP/E will use to link-edit the load module. (For details on the SMPLTS data set, see the *SMP/E Reference* and (for OS/390) *OS/390 SMP/E Commands* manuals.) Here are examples of defining the DDDEF entries, assuming that the function will be applied to target zone TGT1.

```

3 SET BDY(TGT1).           /* Set to target zone.   */
    UCLIN.                  /*                          */
    ADD DDDEF(PLIBASE)      /* Define PLIBASE.       */
        DA(SYS1.V2R2M0.PLIBASE) /* Data set is cataloged. */
        SHR.                /* SHR for read.         */
    ADD DDDEF(APPBASE)     /* Define APPBASE.       */
        DA(SYS1.V2R2M0.APPBASE) /* Data set is cataloged. */
        SHR.                /* SHR for read.         */
    ADD DDDEF(SMPLTS)      /* Define SMPLTS.        */
        DA(SYS1.SMPLTS)      /* Data set is cataloged. */
        SHR.                /* SHR for read.         */
    ENDUCL.

```

9.7.4.3 Restrictions in CALLLIBS Support

CALLLIBS support puts restrictions on the following:

- **Use of the CALL and NCAL parameters.** Processing of the CALL and NCAL parameters in SMP/E Release 8 is different from processing of those parameters in previous SMP/E releases.

Before, NCAL was a default parameter passed to the link-edit utility. However, you could use the link-edit UTILITY entry to pass the CALL parameter instead.

With CALLLIBS support, there is no longer any way to directly tell SMP/E to pass the NCAL or CALL parameter. SMP/E ignores any specification of NCAL or CALL, and instead checks for the CALLLIBS subentry in the load module's LMOD entry to determine which parameter to pass to the link-edit utility when linking the load module.

- **Sharing zones between different releases of SMP/E.** Users cannot share zones between SMP/E Release 8 (or later) and previous releases of SMP/E. This is because in SMP/E Release 8, the structure of the LMOD entries has changed to support the new CALLLIBS subentry list. (LMOD entries are typically updated when JCLIN that defined the load module is processed.)

9.7.5 JCLIN Data for Load Modules Residing in a Hierarchical File System

A load module can reside in a hierarchical file system (HFS). To determine where the load module resides, SMP/E uses the following information, in addition to the usual JCL statements needed for load modules:

- The PATH operand on the SYSLIB or SYSLOAD statement associated with the load module. The PATH operand alerts SMP/E to the fact that the load module resides in a hierarchical file system (HFS); however, the PATH value specified is ignored.
- The LIBRARYDD comment statement immediately following the statement with the PATH operand. This comment statement specifies the ddname to be associated with the PATH value on the previous DD statement.
- The user-provided DDDEF entry whose name matches the ddname on the LIBRARYDD comment statement. The DDDEF entry specifies the directory portion of the pathname identified by the ddname. SMP/E uses the PATH value specified in the DDDEF entry to allocate the pathname, and does not check whether this value matches the PATH value specified on the SYSLIB or SYSLOAD DD statement associated with the LIBRARYDD comment.

Following are examples of job steps containing SYSLMOD and SYSLIB DD statements that use the PATH operand.

```

//STEP1 EXEC PGM=IEWBLINK,PARM='RENT,REUS'
1 //SYSLMOD DD PATH='/path_name1/'
2 //*LIBRARYDD=BPXLOAD1
//AOS12 DD DSN=SYS1.AOS12,DISP=SHR
//SYSLIN DD *
    INCLUDE AOS12(MOD00001)
    INCLUDE AOS12(MOD00002)
    ENTRY MOD00001
    NAME LMOD01(R)
/*
//STEP2 EXEC PGM=IEWBLINK,PARM='CALL,RENT,REUS'
//SYSLMOD DD PATH=SYS1.LINKLIB,DISP=OLD
//AOS12 DD DSN=SYS1.AOS12,DISP=SHR
3 //SYSLIB DD PATH='/path_calllib3/'
4 //*LIBRARYDD=BPXCALL3
4 // DD DSN=SYS1.PLIBASE,DISP=SHR
3 // DD PATH='/path_calllib4/'
4 //*LIBRARYDD=BPXCALL4
//SYSLIN DD *
    INCLUDE AOS12(MOD00005)
    INCLUDE AOS12(MOD00006)
    ENTRY MOD00005
    NAME LMOD03(R)
/*

```

- 1** Because the SYSLMOD statement specifies a PATH operand, SMP/E expects the next statement to be a LIBRARYDD comment statement.
- 2** Using the ddname on the LIBRARYDD comment, SMP/E updates the LMOD entry for LMOD01 to specify a SYSLIB value of BPXLOAD1. The user needs to provide a DDDEF entry for BPXLOAD1, specifying the appropriate pathname.
- 3** The SYSLIB DD statement is a concatenation of three DD statements. Two of the DD statements specify the PATH operand.
- 4** Using the ddnames on the LIBRARYDD comments and the low-level qualifier of the data set specified on the DSN operand, SMP/E updates the LMOD entry for LMOD03 to specify a CALLLIBS subentry list with the values BPXCALL3, PLIBASE, and BPXCALL4. The user needs to provide DDDEF entries for BPXCALL3 and BPXCALL4, specifying the appropriate pathnames. Likewise, the user needs to define PLIBASE with a DDDEF entry.

Chapter 10. Naming Conventions

This section explains the naming conventions used for the following:

- Component codes (prefix for element and load module names)
- SYSMOD IDs for functions
- Element and load module names
- Library names

10.1 Component Codes

In the SMP/E environment, code for one product must be uniquely distinguishable from code for other products. The best way to keep your code unique is to start the names of all the elements and load modules for that product with a single, unique 3-character identifier. This identifier is called the *component code*. (For more information about component codes, see 10.3, “Element, Alias, and Load Module Names” on page 122.) IBM is offering to register the component codes for your products. This registration ensures that your component code is not used by any other MVS-based, SMP/E-installable products distributed by IBM. To register your component code, do one of the following:

- If you have access to the IBM Information Network (IIN), send a note to user ID USIB34FD at IBMMAIL.
- If you have access to the internet, send a note to bobraz@us.ibm.com.
- Ask your IBM representative to contact IBM Poughkeepsie, Department 31RA (user ID ELEMENT at node KGNVMC).

10.2 SYSMOD IDs for Functions

The SYSMOD ID of a function is called the function modification identifier (FMID). The FMID is a 7-character identifier that needs to be unique to distinguish one product from another. One way to help ensure this uniqueness is to follow the naming convention *tcccrrr*, as described below:

t

is an alphabetic character used to indicate the type of function. Avoid the values used by IBM (A, B, C, D, E, F, G, H, I, J, and K).

ccc

is the product version code. In order to avoid conflicts with IBM service identifiers, the first two characters must be alphabetic. The third character can be alphanumeric. You can help guarantee the uniqueness of the product version code by using the three-character component code you registered with IBM. For details, see 10.1, “Component Codes.”

rrr

is the release value. These three characters identify a specific release of a product function and must all be alphanumeric. The *rrr* value must be unique for all function SYSMODs within a given product version code. This *rrr* value is used in IBM's service system (RETAIN) to track each product uniquely.

10.3 Element, Alias, and Load Module Names

Element names are assigned to each discrete piece of a product, such as macros, modules, source codes, and panels.

It is important to maintain the uniqueness of element and load module names to ensure that:

- Each element can be identified by its owning product
- Elements are not unintentionally overlaid
- Each element can be serviced correctly

Packaging Rules (Element and Load Module Names)

- 14300. All products must use the assigned, unique three-character component code as the first three characters of the element names. The first character of the component code follows the conventions shown below to avoid naming conflicts with elements provided by IBM or by other user-written software.

Value Meaning

A–I When used by IBM, all three characters of the prefix are generally alphabetic (with some exceptions).

Can be used by non-IBM products only if the prefix includes at least one numeric or national character.

J–Z Available only for non-IBM products. The prefix can be all alphabetic or can include numeric or national characters.

- 14400. Two elements with the same element type cannot have the same name—element names must be unique. This is true regardless of whether the elements are in the same product or in different products. For more information, see 6.1, “General Packaging Rules, Restrictions, and Recommendations for Elements” on page 44.
- 14500. Load modules should have unique names, which should begin with the product's assigned three-character prefix. However, the same load modules having the same attributes can be defined to two load libraries.
- 14600. Like-named elements, including aliases, must be in separate target and distribution libraries. These libraries must be in separate RELFILES. This prevents unintentional overlaying of elements.
 - See 6.1, “General Packaging Rules, Restrictions, and Recommendations for Elements” on page 44 for information about restrictions on like-named elements.
 - See 3.1.1, “Format and Contents of the RELFILE Tape” on page 12 for additional rules and requirements concerning RELFILES.
- 14700. If more than one version of a product is intended to coexist in the same zone, the element and load module names must be unique for each version.

The recommended approach is to use a unique, 3-character component code as the first three characters of the element names, as mentioned above.

_____ End of Packaging Recommendations _____

When two different elements have the same name and type, the installation process becomes more complicated because each of these elements must be installed in a different zone. You can avoid this predicament by giving each element a unique name, a unique element type, or both.

10.3.1 NLS Considerations for Element Types

Translated elements should use the appropriate data element type, followed by a 3-character national language identifier as a suffix for the element type (for example, ++PNLENU, and ++PNLFRA). Elements that are not translated should not use the national language suffix (for example, ++SKL).

See Table 18 on page 129 for a list of the national language abbreviations, and 6.4, “Data Element Types” on page 45 for a list of the element types.

10.3.2 Elements with the Same Name

Element names must be unique; two elements with the same type cannot have the same name. However, elements that have different types can have the same name provided that they are contained in different FMIDs. For example, ++PNLENU(ABCPANEL) and ++PNLFRA(ABCPANEL) would be valid.

If you need to define elements with the same name (such as HELP) for programming access, you should use HELP as the alias and assign a unique name, in accordance with the corporate naming standard, as the actual element name. For more information, see 10.3.3, “Alias Names” and Chapter 11, “Packaging for National Language Support (NLS)” on page 127.

10.3.3 Alias Names

Alias names can be assigned to elements or load modules. Alias names are defined on the ALIAS, TALIAS, or MALIAS operand of a ++element statement, or during load module creation. Alias names do not need to begin with the 3- or 4-character component codes. Alias names do not need to be unique within an FMID, but they must be unique within a RELFILE or target or distribution library partitioned data set (PDS). For information about rules for aliases, refer to 10.3, “Element, Alias, and Load Module Names” on page 122.

Note: Macros that are externally invoked can have meaningful alias names; however, the actual name of the macro must conform to the corporate naming standard.

If an alias name is assigned to an element, the RELFILE tape must contain both the element and the alias in a RELFILE. (Alias members must be created using an appropriate utility, such as IEBGENER.)

10.4 Library Names

Whenever possible, elements must be assigned to existing distribution libraries and target libraries (which are specified on the DISTLIB and SYSLIB operands of the element MCS statements). Otherwise, libraries must follow packaging rules for library names.

Packaging Rules (Library Names)

- 14900. The low-level qualifier of the name of a new distribution or target library must be unique. You must use your registered component code (*ccc*) in your library name. You should also distinguish between your distribution libraries and your target libraries to make it easier for customers to identify your libraries. One way to do this is to use the format *xccczzzz*, where:
 - *x* is the letter for a distribution library or a target library.
 - *ccc* is the component code (the first three characters assigned to the elements).
 - *zzzz* is whatever the product developer chooses to use, to keep the name unique.

Exception: A data set name need not conform to this format if *all* of the following are true:

1. The data set name is required to have a non-conforming low-level qualifier for unavoidable technical reasons. (For example, C language header file data sets are required by the C compiler to use the low-level qualifier of "H.")
2. The data set is *not* specified as a target library in any JCLIN data, either on a SYSLMOD DD statement or on an EXEC statement.
3. The data set name is *not* specified in a SYSLIB concatenation in any JCLIN data.

Data sets that qualify under this exception must still use ddnames with the format *xccczzzz* or *xcccczzzz* as defined above, to comply with rule14910.

- 14910. Every target and distribution library must have a unique ddname.
- 15000. See rule 119 in 9.3, "General Packaging Rules for JCLIN Data" on page 89.

Packaging Rules (Library Names)

- 15010. If a data set whose name does not use the *xccczzzz* format is renamed for any reason, the low-level qualifier of the new data set name must use the *xccczzzz* format, and the data set's new ddname must match the new low-level qualifier.

Exception: A data set name need not conform to this format if *all* of the following are true:

1. The data set name is required to have a non-conforming low-level qualifier for unavoidable technical reasons. (For example, C language header file data sets are required by the C compiler to use the low-level qualifier of "H.")
2. The data set is *not* specified as a target library in any JCLIN data, either on a SYSLMOD DD statement or on an EXEC statement.
3. The data set name is *not* specified in a SYSLIB concatenation in any JCLIN data.

Data sets that qualify under this exception must still use ddnames with the format *xccczzzz* or *xcccczzz* as defined above, to comply with rule 14910.

- 15100. A product's execution must not depend on the high-level qualifier of any data set names. Product code should refer only to ddnames.

A process that depends on a specific data set name may restrict customer processes or naming conventions. You should design your product to rely only on specific ddnames.

Using the low-level qualifier as the ddname for a data set ensures that the ddname will be unique in the SMP/E zone. An advantage of a unique ddname is that the SMP/E DDDEF for that data set is also guaranteed to be unique. If either the ddname or the DDDEF is not unique, products might unnecessarily prevent other products from being installed in the same zone.

 Packaging Recommendations

The variable portion of the library name should be used to describe the library. For example, the type of elements found in the library could be indicated by MOD, MAC, or PNL, or the national language of the library could be indicated by identifiers such as ENU, FRA, or ESP. Table 18 on page 129 lists the national language identifiers.

 End of Packaging Recommendations

Chapter 11. Packaging for National Language Support (NLS)

There are packaging rules and considerations for products that have elements that require translation for *national language support (NLS)*. This section shows several variations of base and additive dependent function SYSMODs and how they are packaged with their language-support dependent function SYSMODs.

Notes:

1. For more information on NLS, see the *National Language Information and Design Guide*, SBOF-3101 (series of books).
2. Refer to 7.2.4, "Deleting SYSMODs (DELETE)" on page 57 for information about language-support dependent functions not deleting additive dependent functions.

Packaging Rules (Language-Sensitive Elements)
<ul style="list-style-type: none"><input type="checkbox"/> 15200. This rule has been changed to a recommendation (see below).<input type="checkbox"/> 15350. Language variants of an element may have the same name for programming access. In this case, package each language variant as a different element type using the same name for each variant. However, because the names are the same, you must assign the elements to different libraries. See Figure 6 on page 47 for an example.<input type="checkbox"/> 15410. This rule has been deleted.

Packaging Recommendations

Each language-support dependent function should have its own unique FMID.

Each supported language should be individually orderable. Each package should ship everything needed to install the function and the language, including all required functions and installation publications.

Languages can be packaged in a number of ways, including:

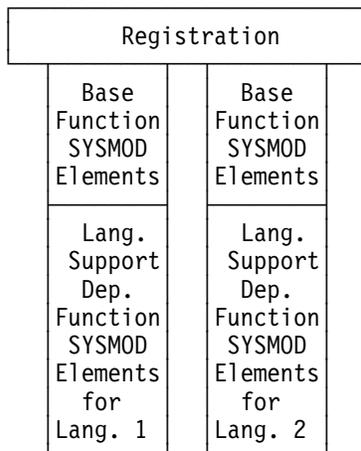
- Each language has a separate FMID
- One language is included in the base function and the rest have separate FMIDs
- All languages are packaged in the base FMID

The decision should be based on such factors as:

- If the language functions are large, separate FMIDs permit customers to save space by only installing the languages they wish to use
- If most customers will want most or all of the languages, using a single FMID makes installation easier without wasting space
- How many tapes will be required to ship the various combinations of functions?

End of Packaging Recommendations

Following is an overview of how to package NLS support for a single base function.



In this example, the elements that do not have to be translated are in a single base function SYSMOD, and the translated elements for each language are in a separate language-support dependent function SYSMOD, one for each language. For each supported language, the base function SYSMOD is packaged with the appropriate language-support dependent function SYSMOD.

For more detailed examples of packaging language-sensitive elements, see Chapter 13, “SYSMOD Packaging Examples” on page 137 and 13.4, “Example 3: Dependent Functions” on page 145.

11.1 Element Types for Translated Elements

Packaging Rules (Language Abbreviations)

- 15500. When the data element or hierarchical file system (HFS) element MCS indicates the language being supported, use one of the national language identifiers shown in Table 18 on page 129 as the three-character suffix for the element type.
- 15600. Each language variant of an element type constitutes a distinct element type, and rules applying to element types apply to every such variant. For example, ++PNLENU and ++PNLDEU are two different element types.

You should not use a national language identifier for a data element that was not translated.

Table 18. National Language Identifiers Used for Language-Unique Elements. See 6.4, “Data Element Types” on page 45 and 6.5, “Hierarchical File System (HFS) Element Types” on page 47 for a list of element MCS you can use these identifiers with.

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

Table 18 might not reflect the most currently supported values. For the latest information on national language identifiers, see the *SMP/E Reference* manual.

Chapter 12. Packaging for Special Situations

This chapter offers packaging suggestions to accommodate the following:

- High-level languages (HLL)
- C language prelinker
- Workstation code
- hierarchical file system (HFS)

12.1 High-Level Languages

Because SMP/E supports the automatic library call facility through the use of SYSLIB DD statements, products now have an alternative to postinstallation link-edit jobs or explicitly defining all the modules to be included in load modules. This section contains two parts: one for packagers who can take advantage of the support in SMP/E Release 8 and later, and one for those who cannot.

12.1.1 Support in SMP/E Release 8 and Later for the Automatic Library Call Facility

If you require SMP/E Release 8 or later as the minimum level of SMP/E for installing your product, you can use SYSLIB DD statements and the automatic library call facility to implicitly include modules. For more information about using the SYSLIB DD statement for such products, see the description of that DD statement in 9.6.2, “Link-Edit Control Statements” on page 97.

Packaging Rules (Libraries)
<ul style="list-style-type: none">□ 15700. This rule has been deleted.□ 15800. This rule has been deleted.□ 15810. If a ++PROGRAM element is prebound with parts from another product, it must use the lowest supported level of the borrowed parts, and must require that level or higher as a functional (noninstallation) requisite. This will avoid problems in customer environments with varying levels of the product.

12.1.2 If You Cannot Use the Automatic Library Call Facility

SMP/E does not exploit the automatic library call option when the link-edit utility is invoked; however, this automatic library call option is used by most high-level languages to include the resident library routines in the load module.

There are two ways to address this problem:

- Use a postinstallation link-edit job. This is the most flexible method.
- Use JCLIN to explicitly identify all the library routines to SMP/E.

Each of these options requires extra packaging and installation steps. There is no complete or easy solution.

12.1.2.1 Using a Postinstallation Link-Edit Job

For this approach, you do not identify the resident libraries to SMP/E. The typical SMP/E link-edit options are NCAL and LET. The link-edit utility issues messages indicating unresolved external references, but these can be ignored. You can provide the user with a link-edit automatic library call option job to be run after installation in order to include the required library routines.

Considerations

1. This requires an additional, manual step to complete the installation. In such cases, errors are likely.
2. SMP/E will not know about this extra link-edit and the resident libraries routines that now are in the load module. This may not be a problem, because SMP/E always includes the old load module when creating a new one. In this case, as long as no changes are required to the resident library routines, those routines continue to exist in the new load module as before.
3. SMP/E does not relink the load module to incorporate maintenance or product-level changes processed for the resident libraries.
4. It is difficult to install code changes affecting the resident libraries. One approach is to rerun the postinstallation job. This is a manual process, and PTFs requiring it need to be held for an ACTION reason ID. Another disadvantage is that the input load module already contains all the resident library routines previously added (from the prior invocations of the job). The automatic library call option works only for unresolved references left *after* the inclusion of the existing load module; only net additions can be processed. Changes or deletions are not done unless you can return to the original point of including only your pieces in the link-edit. Similarly, you cannot use the postinstallation link-edit to bring maintenance to current levels for the resident libraries, because the existing versions are always included first.

There are some alternatives to deal with the problems described in items 3 and 4. You can do a postinstallation link-edit with only the modules that the product owns. The automatic library call option would then include the resident library routines. Do one of the following:

- Code the postinstallation link-edit job to include from the distribution libraries rather than the target load module. This forces complete processing of the automatic library call option, because the distribution library data sets have only your product's code in them.

These are some of the problems with this method:

- The JCL is more complex. You may have to include multiple parts instead of just including the target load module.
 - The user must accept the product into the distribution libraries before running the link-edit job.
 - Because the user must accept all required maintenance before rerunning postinstallation link-edit, the SMP/E RESTORE command cannot be used for recovery.
- Provide two copies of each affected load module: one to be the target of the postinstallation link-edit, and one to be the source.

The load modules would be the same from an SMP/E standpoint and would have the same contents. They could be defined to SMP/E as either two load modules in the same library with different names or as the same load module in two different target libraries. SMP/E can handle either case. A consideration is the amount of space already used in the target library. If the number of load modules is small, you may opt to have two copies in the same target library: one executable, and one not. The alternative creates additional complexity by using a new target library.

In either case, the postinstallation link-edit must be set up to include the second load module to relink the first (executable) load module. In this way, whenever PTFs for your product are installed, SMP/E automatically includes the changes in both load modules. If a PTF does not change the resident libraries, nothing needs to be done. However, if a PTF does change the resident libraries, a hold reason ID of ACTION can be specified for the PTF to indicate that a postinstallation link-edit job needs to be rerun; the automatic library call option includes the correct set of resident libraries. Because all of this is done against target libraries, the RESTORE command can be used to remove bad maintenance and can be followed by the link-edit job again, if necessary.

12.1.2.2 Using JCLIN to Identify Library Routines

You can define to SMP/E all the library routines used by the link-edit utility. You must use JCLIN to identify all the INCLUDE statements for the library routines needed.

If all the following conditions are true, SMP/E can correctly construct the load module without using the automatic library call option.

- The target load module is *new* (not preexisting).
- The required resident library routine is installed in the *same* target/DLIB set of SMP/E zones as that used for the new product.
- You have supplied correct JCLIN describing the routines needed. In this case, SMP/E uses its load module build function to generate the correct INCLUDEs in the link-edit input to build the load module.

Subsequent PTF maintenance to either your product or to the resident library routines causes the load module to be relinked with the updated parts, because SMP/E knows where to find all the load modules that must include the part. This link-edit process includes the *old* load module when the *new* load module is created; SMP/E ensures that no parts are lost.

Considerations

1. The JCLIN must be accurate so that no required resident routines are missed.
2. The load module must be *new*. If the load module is already known to SMP/E, SMP/E just includes the old copy without invoking load module build.
3. Changes in the required resident routines are difficult. For example, what if you change the source code such that a different library routine is required? If nothing else is done, SMP/E uses the JCLIN as you supply it; however, the load module build function is not called, because the load module already exists. Additional routines are not included as required.

4. Installation of new releases also have complications for similar reasons. Typically, the new release uses SMP/E ++VER DELETE processing to remove the old level. SMP/E tries to delete the load modules owned by the old FMID before applying the new release. However, because SMP/E knows that the load modules contain pieces belonging to other FMIDs (the resident library routines), it does not do a total delete. SMP/E deletes the old pieces but leaves the load module in place with the associated resident library routines still there.

When the new product is installed, SMP/E knows that the load module still exists, so load module build is not used, and SMP/E includes the old load module. This might not cause a problem if the exact same set of resident routines is required. If there are any changes to the resident routines, however, the load module will not be correct.

5. There can be drawbacks to automatically relinking a product's load modules whenever there is a maintenance or product-level change to the resident libraries:
 - There may be a problem in the new level of subroutine because of code problems and interface changes. This can cause problems, even though you did not change anything.
 - If there is a product change, the situation is worse if the new level of the resident library deletes the old level.

In this case, SMP/E does the following:

- a. Deletes the old pieces of the resident library wherever they occur. This means it removes them from the load modules.
- b. Deletes the SMP/E information about the old pieces from the SMP/E zone. This includes deleting the links to your load modules.
- c. Installs the new pieces of the resident library. The load modules are not updated with the equivalent new parts.

However, SMP/E maintains a record of any modules from a deleted product that were included in a load module of another product. If the deleted modules are reintroduced, SMP/E automatically link-edits the load module to include the borrowed modules. This can be helpful but, depending on the products involved, SMP/E may try to include modules that no longer exist, and it might not include all the modules you need.

12.2 Using the C Language Prelinker

SMP/E does not invoke the C Prelinker. The C Prelinker is needed for:

- Reentrancy

Some products have avoided use of the C Prelinker by writing the code in a naturally reentrant format.

Note: If your product has already been developed, this option may not work for you.

- Support of long names

There are instances where using the C Prelinker cannot be avoided. The following example explains how a product can avoid a packaging problem if the C Prelinker must be used.

12.2.1 Example of a Product Requiring the C Prelinker

Product A, which is written in C, includes:

- **Load module ABCLMOD**, which contains these CSECTs used as input to the Prelinker (shown in Figure 7):
 - **CSECT ABCM1**
 - **CSECT ABCM2**
 - **CSECT ABCM3**
- **Text deck ABCT1**, which is the Prelinker output from ABCM1, ABCM2, and ABCM3. ABCT1 is shipped as a module in product A.

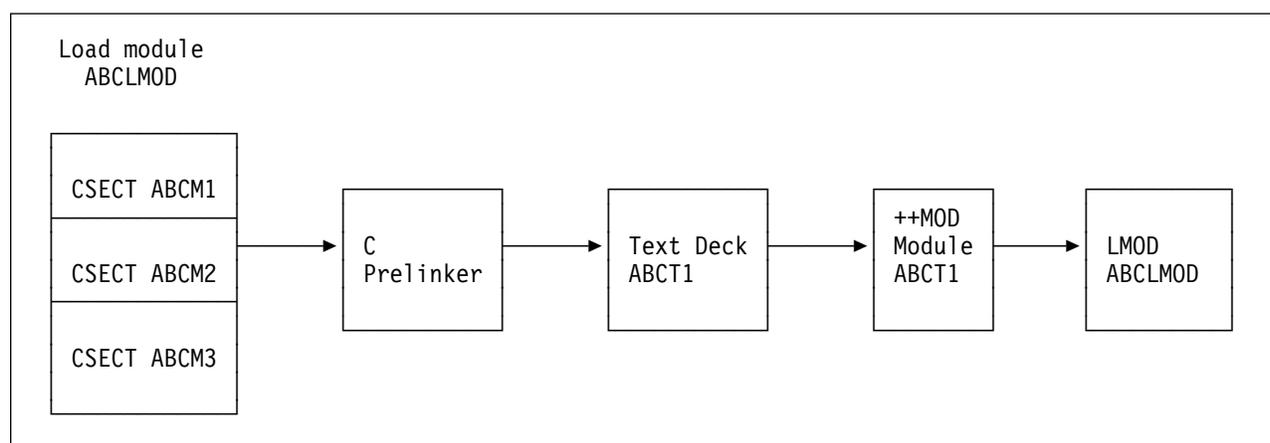


Figure 7. Using the C Prelinker to Create Load Module ABCLMOD

The MCS statement that describes module ABCT1 is:

```
++MOD(ABCT1) CSECT(ABCM1,ABCM2,ABCM3) DISTLIB(nnnnnn) RELFILE(n).
```

The product tape for product A contains module ABCT1, which must be in link-edit format, as required for modules on RELFILE tapes. You do not need MCS statements for CSECTs ABCM1, ABCM2, or ABCM3. You also need to provide JCLIN to indicate that load module ABCLMOD contains module ABCT1.

Suppose that an error is later discovered in CSECT ABCM1. The service process supplies an updated copy of CSECT ABCM1, in addition to CSECTs ABCM2 and ABCM3, which are at the same level as were shipped on the product tape. All three CSECTs must be shipped so that an updated text deck ABCT1 can be created. Module ABCT1 is shipped as an inline module replacement PTF. (It is a service requirement for PTFs that modules be in inline format.) The MCS statement for this new level of ABCT1 is:

```
++MOD(ABCT1) CSECT(ABCM1,ABCM2,ABCM3) DISTLIB(nnnnnn).
```

When the PTF is installed, SMP/E invokes the link-edit utility to link-edit module ABCT1 into load module ABCLMOD. Because the structure of load module ABCLMOD has not changed, no JCLIN is required.

Servicing modules shipped in this manner can have some complications. Large PTFs may result because both updated CSECTs and unchanged CSECTs for a module must be shipped when servicing that module. The advantage of this method is that the product can use the normal service process, and no special customer action is required when installing service.

12.3 Packaging Workstation Code to Be Installed on the Host

There may be instances where workstation code needs to be installed on the host and downloaded to workstations. One of the advantages of delivering workstation code to an MVS host is that it can be maintained under SMP/E control; central service can be used to supply updates. Because such code is MVS-installable, it must also comply with the packaging rules.

12.4 Hierarchical File System (HFS)

Packaging Rules (Hierarchical File System)
<ul style="list-style-type: none">□ 18810. Symbolic links must not exist in the /tmp, /dev, /var or /etc directories.□ 18820. Products must not install anything directly into the /etc directory during APPLY processing; the /etc directory is used only for customization data. Shell scripts invoked by SMP/E must not install or change files in the /etc directory.□ 18830. Permission bits for every file or directory in the hierarchical file system (HFS) must be User >=6, Group >= 4, and Other >= 4. Directories containing SMP/E-installed files must be User=7, Group=5, Other=5.

Packaging Recommendations

- Products should not provide jobs, execs, scripts, or instructions to create files or directories under the /var, /tmp, or /dev directories. If a product needs one of these directories for execution, it should be created dynamically by the product during execution.
- The permission bits for HFS files should be User=7, Group=5, Other=5 for executables, and User=6, Group=4, Other=4 for all other files. (NOTE: there may be some exceptions for daemons, started tasks, and other setuid 0 programs.)
- Products should not require a product-specific HFS. Instead, document the amount of space needed for the product, and allow the installer decide whether or not to install in the root HFS.

End of Packaging Recommendations

Chapter 13. SYSMOD Packaging Examples

This chapter illustrates relationships when developing and servicing the following sample products:

- Product A: defining a stand-alone base function (with support for only U.S. English elements)
- Products B and C: defining corequisite base functions (with support for only U.S. English elements)
- Products B and C: defining dependent functions (with support for only U.S. English elements)
- Products C, D, and E: defining base functions with prerequisites (with support for only U.S. English elements)
- Product E: defining mutually exclusive dependent functions (with support for only U.S. English elements)
- Products X and Y: defining functions that support more than one language
- Products K, L, and M: changing the contents of products

13.1 Conventions Used in This Chapter

Please Note

This chapter contains “skeleton” SYSMOD packaging examples to highlight packaging concepts. The syntax used in the examples is not complete. For example, certain operands such as DISTLIB and SYSLIB are not shown, to focus on other operands whose use is being demonstrated.

To make the SYSMOD packaging examples easier to read, shortened forms of product names and SYSMOD IDs are used. For example:

Product A single letter is used, such as Product A or Product B.

Function Each base function FMID starts with “W,” and each dependent function FMID starts with “X.” This is followed by the product letter and a number (for example, Function WA0 or Function XB1). (For language-support dependent functions, an abbreviation indicating the language is also included—for example, XA0ENU.)

Program temporary fix, or PTF

The letter P, a letter associated with the product, and a single number are used, such as PTF(PA1) or PTF(PA2).

APAR fix The letter R, a letter associated with the product, and a single number are used, such as APAR(RA1).

SYSMOD in general

A single number is used, such as SYSMOD(1) or SYSMOD(2).

Element A single letter is used, such as MOD(J) or MOD(K). (For language-sensitive elements, an abbreviation identifying the product is also included—for example, AP to identify an element for product A.)

13.2 Example 1: A Stand-Alone Function

Suppose you have developed a new product (J) that has no dependencies on other SYSMODs that may be installed on the same system, and that supports only U.S. English in its dialog panels and messages. These are some of the SYSMODs you may define in the course of developing and servicing product A:

- The initial release of the product. This release consists of the:
 - Base function
 - Language-support dependent function for English
- PTF service for the initial release
- PTF service that depends on previous service
- Replacing the initial release
- Ensuring that a fix for a previous release is not lost
- Integrating PTF service for a service update

13.2.1 Initial Release

The first release of product A is packaged as FMID WA0 (a base function) and FMID XA0ENU (its language-support dependent function). Because function WA0 is the base function, it is an **unconditional prerequisite** for function XA0ENU. This relationship is defined by the FMID operand on the ++VER statement for XA0ENU, as shown in Figure 8. However, because these functions have no relationships with other SYSMODs, no other requisites need to be specified on their ++VER statements.

Product A
<pre> ++FUNCTION(WA0) REWORK(1989260). ++VER(Z038). ++JCLIN RELFILE(1). ++MOD(J). ++MOD(K). </pre>
<pre> ++FUNCTION(XA0ENU) REWORK(1989260). ++VER(Z038) FMID(WA0). ++JCLIN RELFILE(1). ++PNLENU(AP). ++MSGENU(AM). </pre>

Figure 8. Initial Release

13.2.2 PTF Service for the Initial Release

Suppose a user has reported an error in function WA0, and you have packaged the fix as an APAR (RA1) to correct the problem on that user's system. Now it appears that the problem may occur on all users' systems. To distribute the fix as service to all users, you do additional testing on the correction and package it as PTF(PA1). The fix is applicable to function WA0, so WA0 is an **unconditional prerequisite** for PTF(PA1). You define this relationship by coding a ++VER statement that specifies function WA0 as the FMID, as shown in Figure 9 on page 139. To ensure that the APAR cannot be installed on top of the PTF, and thus regress the changes, you should also have the PTF **supersede** the APAR.

Product A
<pre> ++FUNCTION(WA0) REWORK(1989260). ++VER(Z038). ++JCLIN RELFILE(1). ++MOD(J). ++MOD(K). </pre>
<pre> ++APAR(RA1). ++VER(Z038) FMID(WA0). ++MOD(J). </pre>
<pre> ++PTF(PA1). ++VER(Z038) FMID(WA0) SUP(RA1). ++MOD(J). </pre>

Figure 9. PTF Service for the Initial Release

13.2.3 PTF Service That Depends on Previous Service

After a while you have some service for function WA0. This fix depends on some of the changes made by PTF(PA1). You package the fix as a PTF, PA2, for product A. The fix is applicable to function WA0, so WA0 is an **unconditional prerequisite** for PTF(PA2). You define this relationship by coding a ++VER statement that specifies function WA0 as the FMID, as shown in Figure 10. In addition, because PA2 depends on changes made by PA1, PA1 is also an **unconditional prerequisite** for PTF(PA2). You define this relationship by coding a ++VER statement that specifies PTF(PA1) as a prerequisite.

Product A
<pre> ++FUNCTION(WA0) REWORK(1989260). ++VER(Z038). ++JCLIN RELFILE(1). ++MOD(J). ++MOD(K). </pre>
<pre> ++APAR(RA1). ++VER(Z038) FMID(WA0). ++MOD(J). </pre>
<pre> ++PTF(PA1). ++VER(Z038) FMID(WA0) SUP(RA1). ++MOD(J). </pre>
<pre> ++PTF(PA2). ++VER(Z038) FMID(WA0) PRE(PA1). ++MOD(K). </pre>

Figure 10. PTF Service That Depends on Previous Service

Note: To make the rest of the examples in this section easier to read, none of them show APAR fixes being superseded by PTFs. (APAR fixes are normally superseded by the PTFs that include them.)

13.2.4 Ensuring That a Fix for a Previous Release Is Not Lost

Suppose you are completing base function WA2, the second release of product A, and have included all the PTFs that were issued for WA1. After doing this, however, you had to add another PTF(PA8) to fix module B in function WA1. But, because the development cycle for function WA2 has passed the APAR cutoff point, the fix cannot be included in WA2. You want to make sure that any users who install PTF(PA8) do not lose those corrections when they install function WA2.

To do this, you must also code PTF(PA9), which fixes the same problem as PA8, only for function WA2. You must ensure that users who had installed PA8 will install PA9 along with function WA2. PTF(PA9) is, therefore, a **conditional corequisite** for function WA2. You define this relationship by coding an ++IF statement in PTF(PA8), as shown in Figure 11.

Product A	
Old Release	New Release
<pre> ++FUNCTION(WA1) REWORK(1990050). ++VER(Z038). ++JCLIN RELFILE(1). ++MOD(J). ++MOD(K). </pre>	<pre> ++FUNCTION(WA2) REWORK(1990100). ++VER(Z038) DELETE(WA1) SUP(WA1). ++JCLIN RELFILE(1). ++MOD(J). ++MOD(K). </pre>
<pre> ++PTF(PA8). ++VER(Z038) FMID(WA1). ++IF FMID(WA2) REQ(PA9). ++MOD(K). </pre>	<pre> ++PTF(PA9). ++VER(Z038) FMID(WA2). ++MOD(K). </pre>

Figure 11. Ensuring That a Fix for a Previous Release Is Not Lost

When a user tries to install PTF(PA8), SMP/E does one of two things:

- If function WA2 is already installed, SMP/E cannot install PTF(PA8) and does not know that PA9 is required for WA2. Because PA8 was never installed on function WA1, the system is not at a lower level when function WA2 is installed without PA9. PA9 is eventually installed when the user processes service for WA2, and the problem is fixed.
- If function WA2 is not yet installed, SMP/E notes that PA9 is needed for WA2, and saves this information. Later, if the user tries to install function WA2, SMP/E makes sure PTF(PA9) is also installed. This ensures that the corrections from PA8 are not lost.

13.2.5 Replacing the Initial Release

Suppose there are a number of improvements you want to make in product A, so you are thinking of packaging a new release. This new release could delete the initial release, supersede it, or both. (See Table 10 on page 28 if you need to review the differences.)

In this example, base function WA1 supersedes and deletes base function WA0. Function WA1 is, therefore, an unconditional replacement for WA0. You define this relationship by coding the SUP and DELETE operands on the ++VER statement for function WA1, as shown in Figure 12.

A new release of the language-support dependent function is required for function WA1. This new release (XA1ENU) must supersede the previous release (XA0ENU), as shown in Figure 12.

Note: There is no need for WA1 or XA1ENU to delete XA0ENU, because XA0ENU is automatically deleted when WA0 is deleted. However, an explicit deletion is recommended for purposes of documentation.

Product A	
Old Release	New Release
<pre> ++FUNCTION(WA0) REWORK(1989260). ++VER(Z038). ++JCLIN RELFILE(1). ++MOD(J). ++MOD(K). </pre>	<pre> ++FUNCTION(WA1) REWORK(1990050). ++VER(Z038) DELETE(WA0) SUP(WA0). ++JCLIN RELFILE(1). ++MOD(J). ++MOD(K). </pre>
<pre> ++FUNCTION(XA0ENU) REWORK(1989260). ++VER(Z038) FMID(WA0). ++JCLIN RELFILE(1). ++PNLENU(AP). ++MSGENU(AM). </pre>	<pre> ++FUNCTION(XA1ENU) REWORK(1990050). ++VER(Z038) FMID(WA1) SUP(XA0ENU). ++JCLIN RELFILE(1). ++PNLENU(AP). ++MSGENU(AM). </pre>

Figure 12. Replacing the Initial Release

Suppose the previous example left the ++IF statement out of PTF(PA8). This creates the possibility of customers regressing their systems by installing WA2 without PTF(P9).

To avoid this problem, the product owner could change the packaging for WA2 and define PA9 as an **unconditional requisite**. This is done by specifying the REQ operand on the ++VER statement, as shown in Figure 13 on page 142.

Product A	
Old Release	New Release
<pre> ++FUNCTION(WA1) REWORK(1990050). ++VER(Z038) DELETE(WA0) SUP(WA0). ++JCLIN RELFILE(1). ++MOD(J). ++MOD(K). </pre>	<pre> ++FUNCTION(WA2) REWORK(1990100). ++VER(Z038) DELETE(WA0,WA1) SUP(WA0,WA1). REQ(PA9). ++JCLIN RELFILE(1). ++MOD(J). ++MOD(K). </pre>
<pre> ++PTF(PA8). ++VER(Z038) FMID(WA1). ++MOD(K). </pre>	<pre> ++PTF(PA9). ++VER(Z038) FMID(WA2). ++MOD(K). </pre>

Figure 13. Correcting an Erroneous Post-Cutoff PTF

13.3 Example 2: Corequisite Base Functions

Suppose you have developed a new function that involves elements from two different products, B and C. Each product provides specific aspects of the function, but the code works properly only if the two products are installed together. Both products support only U.S. English in their dialog panels and messages. These are some of the SYSMODs you might have to define in the course of developing and servicing products B and C:

- The initial releases of the products. These consist of:
 - The base functions
 - The language-support dependent functions
- PTF service for one of the base functions
- Cross-product service between the base functions
- Deleting and superseding one of the base functions

13.3.1 Initial Releases of Corequisite Functions

B and C are products that, together, provide a new function. These products could be packaged as base-function SYSMODs that are **unconditional corequisites**. This relationship is defined by the REQ operand on the ++VER statement for each base function, as shown in Figure 14 on page 143.

In addition, a language-support dependent function is provided for each base function. Each base function is an **unconditional prerequisite** for its corresponding language-support dependent function. This relationship is defined by the FMID operand on the ++VER statement for each dependent function, as shown in Figure 14 on page 143.

Product B	Product C
<pre> ++FUNCTION(WB0) REWORK(1990020). ++VER(Z038) REQ(WC0). ++JCLIN RELFILE(1). ++MOD(C). ++MOD(D). ++MOD(E). ++MOD(F). ++MOD(G). </pre>	<pre> ++FUNCTION(WC0) REWORK(1990020). ++VER(Z038) REQ(WB0). ++JCLIN RELFILE(1). ++MOD(M). ++MOD(N). </pre>
<pre> ++FUNCTION(XB0ENU) REWORK(1990020). ++VER(Z038) FMID(WB0). ++JCLIN RELFILE(1). ++PNLENU(BP). ++MSGENU(BM). </pre>	<pre> ++FUNCTION(XC0ENU) REWORK(1990020). ++VER(Z038) FMID(WC0). ++JCLIN RELFILE(1). ++PNLENU(CP). ++MSGENU(CM). </pre>

Figure 14. Initial Releases of Corequisite Functions

13.3.2 PTF Service for One of the Base Functions

Suppose you need to provide service for module C in function WB0. The fix is applicable to function WB0, so WB0 is an **unconditional prerequisite** for PTF(PB1). You define this relationship by coding a ++VER statement that specifies function WB0 as the FMID, as shown in Figure 15.

Product B
<pre> ++FUNCTION(WB0) REWORK(1990020). ++VER(Z038) REQ(WC0). ++JCLIN RELFILE(1). ++MOD(C). ++MOD(D). ++MOD(E). ++MOD(F). ++MOD(G). </pre>
<pre> ++PTF(PB1). ++VER(Z038) FMID(WB0). ++MOD(C). </pre>

Figure 15. PTF Service for One of the Base Functions

13.3.3 Cross-Product Service between Corequisite Base Functions

Suppose you need to provide service that affects module D and module M. Module D is owned by function WB0, and module M is owned by function WC0. To fix the problem, you need two PTFs, one for each module. PB2 fixes module D, and PC1 fixes module M. These PTFs are **conditional corequisites**. (This also ensures that each PTF can still be installed if the requisite product is deleted by a new release, superseded by a new release, or both.) You define this relationship by

coding the FMID and REQ operands on each PTF's ++IF statement, as shown in Figure 16 on page 144.

Product B	Product C
<pre> ++FUNCTION(WB0) REWORK(1990020). ++VER(Z038) REQ(WC0). ++JCLIN RELFILE(1). ++MOD(C). ++MOD(D). ++MOD(E). ++MOD(F). ++MOD(G). </pre>	<pre> ++FUNCTION(WC0) REWORK(1990020). ++VER(Z038) REQ(WB0). ++JCLIN RELFILE(1). ++MOD(M). ++MOD(N). </pre>
<pre> ++PTF(PB2). ++VER(Z038) FMID(WB0). ++IF FMID(WC0) REQ(PC1). ++MOD(D). </pre>	<pre> ++PTF(PC1). ++VER(Z038) FMID(WC0) ++IF FMID(WB0) REQ(PB2). ++MOD(M). </pre>

Figure 16. Cross-Product Service between Corequisite Base Functions

13.3.4 Deleting and Superseding a Base Function

Suppose there are a number of improvements you want to make in product C, so you are thinking of packaging a new release. The new release could delete the initial release, supersede it, or both. (See Table 10 on page 28 if you need to review the differences.)

In this case, you have decided that function WC1 will **unconditionally delete and supersede** WC0. This is to ensure that requisites specified by function WB0 are satisfied by both releases of product C. You define this relationship by coding the DELETE and SUP operands on the ++VER statement for function WC1, as shown in the figure below.

You must also provide a new release of the language-support dependent function. This new release (XC1ENU) must supersede the previous release (XC0ENU), as shown in Figure 17 on page 145.

Note: There is no need for WC1 or XC1ENU to delete XC0ENU. XC0ENU will automatically be deleted when WC0 is deleted. However, an explicit DELETE is recommended for documentation purposes.

Product C	
Old Release	New Release
<pre> ++FUNCTION(WC0) REWORK(1990020). ++VER(Z038) REQ(WB0). ++JCLIN RELFILE(1). ++MOD(M). ++MOD(N). </pre>	<pre> ++FUNCTION(WC1) REWORK(1990120). ++VER(Z038) REQ(WB0) DELETE(WC0,XC0ENU) SUP(WC0). ++JCLIN RELFILE(1). ++MOD(M). ++MOD(N). </pre>
<pre> ++FUNCTION(XC0ENU) REWORK(1990020). ++VER(Z038) FMID(WC0). ++JCLIN RELFILE(1). ++PNLENU(CP). ++MSGENU(CM). </pre>	<pre> ++FUNCTION(XC1ENU) REWORK(1990120). ++VER(Z038) FMID(WC1) SUP(XC0ENU). ++JCLIN RELFILE(1). ++PNLENU(CP). ++MSGENU(CM). </pre>

Figure 17. Deleting and Superseding a Base Function

If WC1 had only deleted WC0, instead of deleting and superseding it, any function or service that needed WC0 could not be installed without special processing. For example, users would need to have SMP/E bypass requisite checking to install the function or service. Because you know that WB0 has dependencies on WC0, you want to avoid this problem by having WC1 both delete and supersede WC0.

13.4 Example 3: Dependent Functions

In the course of developing products B and C you may decide to provide some optional enhancements that add to, but do not replace, the initial base functions. These enhancements would be packaged as dependent functions for the parent base functions, and would have their own language-support dependent functions. These are some of the SYSMODs you might have to define in the course of developing and servicing dependent functions for products B and C:

- The initial release of a dependent function. This release consists of:
 - The additive dependent function itself
 - The associated language-support dependent function
- PTF service for a dependent function
- Corequisite PTFs with an element common to the base and dependent functions
- Corequisite PTFs with no elements common to the base and dependent functions
- Repackaging a dependent function for a new release of the parent base function
- Deleting a dependent function
- Establishing the order of additional dependent functions
- Corequisite dependent functions

13.4.1 Initial Release of a Dependent Function

Suppose you have decided to provide an optional enhancement for product B. You package it as XB1, a dependent function for the parent base function WB0. Because function WB0 is the base function, it is an **unconditional prerequisite** for function XB1. You define this relationship by coding the FMID operand on the dependent function's ++VER statement, as shown in Figure 18.

Product B	
Base Function and Language Support	Dependent Function and Language Support
<pre> ++FUNCTION(WB0) REWORK(1990020). ++VER(Z038) REQ(WC0). ++JCLIN RELFILE(1). ++MOD(C). ++MOD(D). ++MOD(E). ++MOD(F). ++MOD(G). </pre>	<pre> ++FUNCTION(XB1) REWORK(1990070). ++VER(Z038) FMID(WB0). ++JCLIN RELFILE(1). ++MOD(F). ++MOD(H). ++MOD(J). </pre>
<pre> ++FUNCTION(XB0ENU) REWORK(1990020). ++VER(Z038) FMID(WB0). ++JCLIN RELFILE(1). ++PNLENU(BP). ++MSGENU(BM). ++CLISTENU(BC). </pre>	<pre> ++FUNCTION(XB1ENU) REWORK(1990070). ++VER(Z038) FMID(WB0) PRE(XB1,XB0ENU) VERSION(XB0ENU). ++JCLIN RELFILE(1). ++PNLENU(BP). ++MSGENU(BM). </pre>

Figure 18. Initial Release of a Dependent Function

13.4.2 PTF Service for a Dependent Function

Suppose you need to provide service for module H in function XB1. The fix is applicable to function XB1, so XB1 is an **unconditional prerequisite** for PTF(PB3). You define this relationship by coding a ++VER statement that specifies function XB1 as the FMID, as shown in Figure 19.

Product B
<pre> ++FUNCTION(XB1) REWORK(1990070). ++VER(Z038) FMID(WB0). ++JCLIN RELFILE(1). ++MOD(F). ++MOD(H). ++MOD(J). </pre>
<pre> ++PTF(PB3). ++VER(Z038) FMID(XB1). ++MOD(H). </pre>

Figure 19. PTF Service for a Dependent Function

13.4.3 Corequisite PTFs with an Element Common to the Base and Dependent Functions

Suppose you need to provide service that affects module F and module G. Module G is owned by base function WB0, and module F exists in WB0 and in its dependent function XB1. To fix the problem, you need two PTFs, one for each function. Because the dependent function may be installed with the base function, the PTF for the dependent function is a conditional requisite in the PTF for the base function. However, because the base function must be installed if the dependent function is installed, the PTF for the base function is an unconditional requisite in the PTF for the dependent function. You define these relationships by coding the FMID and REQ operands on the ++IF statement for the base function PTF, and by coding the REQ operand on the ++VER statement for the dependent function PTF, as shown in Figure 20.

Product B	
Base Function and Language Support	Dependent Function and Language Support
<pre> ++FUNCTION(WB0) REWORK(1990020). ++VER(Z038) REQ(WC0). ++JCLIN RELFILE(1). ++MOD(C). ++MOD(D). ++MOD(E). ++MOD(F). ++MOD(G). </pre>	<pre> ++FUNCTION(XB1) REWORK(1990070). ++VER(Z038) FMID(WB0). ++JCLIN RELFILE(1). ++MOD(F). ++MOD(H). ++MOD(J). </pre>
<pre> ++PTF(PB4). ++VER(Z038) FMID(WB0). ++IF FMID(XB1) REQ(PB5). ++MOD(F). ++MOD(G). </pre>	<pre> ++PTF(PB5). ++VER(Z038) FMID(XB1) REQ(PB4). ++MOD(F). </pre>

Figure 20. Corequisite PTFs with an Element Common to the Base and Dependent Functions

Suppose functions WB0 and XB1 were released and a new dependent function XB2 is being packaged that deletes XB1. Figure 21 on page 148 shows how function XB2 and its related language-support dependent function (XB2ENU) are then packaged.

Product B	
Base Function and Language Support	Dependent Function and Language Support
<pre> ++FUNCTION(WB0) REWORK(1990020). ++VER(Z038) REQ(WC0). ++JCLIN RELFILE(1). ++MOD(C). ++MOD(D). ++MOD(E). ++MOD(F). ++MOD(G). </pre>	<pre> ++FUNCTION(XB1) REWORK(1990070). ++VER(Z038) FMID(WB0). ++JCLIN RELFILE(1). ++MOD(F). ++MOD(H). ++MOD(J). </pre>
<pre> ++FUNCTION(XB0ENU) REWORK(1990020). ++VER(Z038) FMID(WB0). ++JCLIN RELFILE(1). ++PNLENU(BP). ++MSGENU(BM). ++CLISTENU(BC). </pre>	<pre> ++FUNCTION(XB1ENU) REWORK(1990070). ++VER(Z038) FMID(WB0) PRE(XB1,XB0ENU) VERSION(XB0ENU). ++JCLIN RELFILE(1). ++PNLENU(BP). ++MSGENU(BM). </pre>
<pre> ++PTF(PB4). ++VER(Z038) FMID(WB0). ++IF FMID(XB1) REQ(PB5). ++MOD(F). ++MOD(G). </pre>	<pre> ++PTF(PB5). ++VER(Z038) FMID(XB1) REQ(PB4). ++MOD(F). </pre>
	<pre> ++FUNCTION(XB2) REWORK(1990110). ++VER(Z038) FMID(WB0) DELETE(XB1,XB1ENU) SUP(XB1,PB4,PB5). ++JCLIN RELFILE(1). ++MOD(F). ++MOD(G). ++MOD(H). </pre>
	<pre> ++FUNCTION(XB2ENU) REWORK(1990110). ++VER(Z038) FMID(WB0) PRE(XB2,XB0ENU) VERSION(XB0ENU) SUP(XB1ENU). ++JCLIN RELFILE(1). ++PNLENU(BP). ++MSGENU(BM). </pre>

Figure 21. New Releases of the Base and Dependent Functions

Notes:

1. Function XB2 deletes and supersedes the previous release of the dependent function (XB1). It also deletes the language-support dependent function (XB1ENU) associated with that previous release.
2. Function XB2 does not have to refer to PTF(PB5) because function XB1 is deleted. However, XB2 does have to supersede PTF(PB4) to make sure that PB4 is not reprocessed by SMP/E.
3. Function XB2ENU supersedes the previous release of the language-support dependent function (XB1ENU).
4. If PTF(PB4) and PTF(PB5) affect two different elements, and the corequisite relationship is still required, the logic is the same.

13.4.4 Corequisite PTFs with All Elements Common to Base and Dependent Functions

Suppose you need to provide service that affects module F, which is present in both base function WB0 and dependent function XB1. To fix the problem, you need two PTFs, one for each function. Because the dependent function can be installed with the base function, the PTF for the dependent function is a **conditional corequisite** of the PTF for the base function.

If the user has the dependent function installed, the PTF for the base function really is not necessary, because the PTF for the dependent function provides a higher level of the element. However, it is important to prevent the user from accidentally installing the PTF for the base function and later downleveling the dependent function's version of the element. Therefore, the PTF for the base function is an **unconditional corequisite** of the PTF for the dependent function.

You define these relationships by coding the FMID and REQ operands on the ++IF statement for the base function PTF, and by coding either the REQ operand or the SUP operand on the ++VER statement for the dependent function PTF, as shown in Figure 22 on page 150.

Product B	
Base Function and Language Support	Dependent Function and Language Support
<pre> ++FUNCTION(WB0) REWORK(1990020). ++VER(Z038) REQ(WC0). ++JCLIN RELFILE(1). ++MOD(C). ++MOD(D). ++MOD(E). ++MOD(F). ++MOD(G). </pre>	<pre> ++FUNCTION(XB1) REWORK(1990070). ++VER(Z038) FMID(WB0). ++JCLIN RELFILE(1). ++MOD(F). ++MOD(H). ++MOD(J). </pre>
<pre> ++FUNCTION(XB0ENU) REWORK(1990020). ++VER(Z038) FMID(WB0). ++JCLIN RELFILE(1). ++PNLENU(BP). ++MSGENU(BM). </pre>	<pre> ++FUNCTION(XB1ENU) REWORK(1990070). ++VER(Z038) FMID(WB0) PRE(XB1,XB0ENU) VERSION(XB0ENU). ++JCLIN RELFILE(1). ++PNLENU(BP). ++MSGENU(BM). </pre>
<pre> ++PTF(PB6). ++VER(Z038) FMID(WB0). ++IF FMID(XB1) REQ(PB7). ++MOD(F). </pre>	<pre> ++PTF(PB7). ++VER(Z038) FMID(XB1) REQ(PB6). ++MOD(F). </pre>

Figure 22. Corequisite PTFs with All Elements Common to Base and Dependent Functions

Suppose functions WB0 and XB1 were released and a new dependent function XB2 is being packaged that deletes XB1, as shown in Figure 23 on page 151.

Product B	
Base Function and Language Support	Dependent Function and Language Support
<pre> ++FUNCTION(WB0) REWORK(1990020). ++VER(Z038) REQ(WC0). ++JCLIN RELFILE(1). ++MOD(C). ++MOD(D). ++MOD(E). ++MOD(F). ++MOD(G). </pre>	<pre> ++FUNCTION(XB1) REWORK(1990070). ++VER(Z038) FMID(WB0). ++JCLIN RELFILE(1). ++MOD(F). ++MOD(H). ++MOD(J). </pre>
<pre> ++FUNCTION(XB0ENU) REWORK(1990020). ++VER(Z038) FMID(WB0). ++JCLIN RELFILE(1). ++PNLENU(BP). ++MSGENU(BM). </pre>	<pre> ++FUNCTION(XB1ENU) REWORK(1990070). ++VER(Z038) FMID(WB0) PRE(XB1,XB0ENU) VERSION(XB0ENU). ++JCLIN RELFILE(1). ++PNLENU(BP). ++MSGENU(BM). </pre>
<pre> ++PTF(PB6). ++VER(Z038) FMID(WB0). ++IF FMID(XB1) REQ(PB7). ++MOD(F). </pre>	<pre> ++PTF(PB7). ++VER(Z038) FMID(XB1) REQ(PB6). ++MOD(F). </pre>
	<pre> ++FUNCTION(XB2) REWORK(1990110). ++VER(Z038) FMID(WB0) DELETE(XB1,XB1ENU) SUP(XB1) SUP(PB6). ++JCLIN RELFILE(1). ++MOD(F). ++MOD(H). ++MOD(J). ++MOD(K). </pre>
	<pre> ++FUNCTION(XB2ENU) REWORK(1990110). ++VER(Z038) FMID(WB0) PRE(XB2,XB0ENU) VERSION(XB0ENU) SUP(XB1ENU). ++JCLIN RELFILE(1). ++PNLENU(BP). ++MSGENU(BM). </pre>

Figure 23. New Releases of the Base and Dependent Functions

Note: Function XB2 does not have to refer to PTF(PB7), because function XB1 is deleted. However, XB2 does have to supersede PTF(PB6) to prevent PB6 from being reprocessed.

13.4.5 Deleting a Dependent Function Without Superseding It

Suppose function WB7 is a new release of dependent function WB6. WB7 changes the external interface of the dependent function so it is no longer compatible with prior releases. WB7 would, therefore, **unconditionally delete** WB6 but must not supersede WB6. You define this relationship by coding the DELETE operand on the ++VER statement for function WB7, as shown in Figure 24.

Product B	
Old Release	New Release
<pre> ++FUNCTION(WB6) REWORK(1990210). ++VER(Z038). ++JCLIN RELFILE(1). ++MOD(F). ++MOD(H). ++MOD(J). ++MOD(K). </pre>	<pre> ++FUNCTION(WB7) REWORK(1990240). ++VER(Z038) DELETE(WB6). ++JCLIN RELFILE(1). ++MOD(F). ++MOD(H). ++MOD(J). ++MOD(K). </pre>

Figure 24. Deleting a Dependent Function

13.4.6 Establishing the Order of Additional Dependent Functions

Suppose you have added an optional enhancement for product B. It is packaged as function XB8, a dependent function. XB8 does not delete or supersede any other dependent functions. However, because it has a requirement for modules in XB7, another dependent function, you must establish which of the dependent functions depends on the other. For example, if function XB8 is functionally higher than function XB7, function XB7 is an **unconditional prerequisite** for function XB8. You define this relationship by coding the PRE operand on function XB8's ++VER statement, as shown in Figure 25.

Product B	
Lower Level	Higher Level
<pre> ++FUNCTION(XB7) REWORK(1990240). ++VER(Z038) FMID(WB5). ++JCLIN RELFILE(1). ++MOD(F). ++MOD(H). ++MOD(J). ++MOD(K). </pre>	<pre> ++FUNCTION(XB8) REWORK(1990260). ++VER(Z038) FMID(WB5) PRE(XB7) VERSION(XB7). ++MOD(F). </pre>

Figure 25. Establishing the Order of Additional Dependent Functions

Note: The VERSION operand is required in functions XB8 and XB8ENU to change ownership of the common elements MOD(F) and PNLENU(BP). See 7.2, “++VER Statement” on page 56 for more information.

13.4.7 Conditional Corequisite Dependent Functions

Suppose you have developed a new user function that involves elements from dependent functions XB9 and XC2. XB9 is a dependent function for base function WB5, and XC2 is a dependent function for base function WC1. The code works properly only if the two dependent functions are installed together. These dependent functions are **conditional corequisites**. (This also ensures that either dependent function can still be installed if the other dependent function's parent base function is deleted by a new release, superseded by a new release, or both.) You define this relationship by coding the FMID and REQ operands on each dependent function's ++IF statements, as shown in Figure 26.

Product B	Product C
<pre> ++FUNCTION(XB9) REWORK(1990300). ++VER(Z038) FMID(WB5). ++IF FMID(WC1) REQ(XC2). ++MOD(L). </pre>	<pre> ++FUNCTION(XC2) REWORK(1990300). ++VER(Z038) FMID(WC1). ++IF FMID(WB5) REQ(XB9). ++MOD(N). </pre>

Figure 26. Corequisite Dependent Functions

13.5 Example 4: Base Functions with Prerequisites

Functions may depend on other functions as prerequisites, or they may depend on service provided for another function. Products C, D, and E are examples of these. Product D depends on product C; product E depends on service for product D. These are some of the relationships you may define in the course of developing and servicing these products:

- The initial release of a base function with a functional prerequisite. This release consists of:
 - The base function itself
 - The associated language-support dependent function
- Dependency on an SPE or service for another base function
- Cross-product service for a base function with a prerequisite

13.5.1 Initial Release of a Base Function with a Functional Prerequisite

Suppose your base function WC0 for product C provides the minimum level of support for base function WD0, the first release of product D. Function WC0 is, therefore, an **unconditional prerequisite** for function WD0. The owner of function WD0 defines this relationship by specifying the REQ operand on the ++VER statement for WD0, as shown in Figure 27 on page 154.

Product C	Product D
<pre> ++FUNCTION(WC0) REWORK(1990020). ++VER(Z038). ++JCLIN RELFILE(1). ++MOD(M). ++MOD(N). </pre>	<pre> ++FUNCTION(WD0) REWORK(1990060). ++VER(Z038) REQ(WC0). ++JCLIN RELFILE(1). ++MOD(P). ++MOD(Q). </pre>
<pre> ++FUNCTION(XC0ENU) REWORK(1990020). ++VER(Z038) FMID(WC0). ++JCLIN RELFILE(1). ++PNLENU(CP). ++MSGENU(CM). </pre>	<pre> ++FUNCTION(XD0ENU) REWORK(1990060). ++VER(Z038) FMID(WD0). ++JCLIN RELFILE(1). ++PNLENU(DP). ++MSGENU(DM). </pre>

Figure 27. Initial Release of a Base Function with a Functional Prerequisite

Suppose you come out with a new release of the base function, WC1. If WC0 is both deleted and superseded by WC1, as shown in Figure 28, WD0 does not need to be repackaged to work with both releases of product C.

Product C		Product D
Old Release	New Release	
<pre> ++FUNCTION(WC0) REWORK(1990020). ++VER(Z038) REQ(WB0). ++JCLIN RELFILE(1). ++MOD(M). ++MOD(N). </pre>	<pre> ++FUNCTION(WC1) REWORK(1990120). ++VER(Z038) REQ(WB0) DELETE(WC0) SUP(WC0). ++JCLIN RELFILE(1). ++MOD(M). ++MOD(N). </pre>	<pre> ++FUNCTION(WD0) REWORK(1990060). ++VER(Z038) REQ(WC0). ++JCLIN RELFILE(1). ++MOD(P). ++MOD(Q). </pre>

Figure 28. New Release of a Base Function with a Functional Prerequisite

Note: The owner of product D must consider that future releases of product C may not be compatible with WD0. For example, they may not provide the required support the same way WC0 or WC1 did. This may not be a problem if they have a part in the development or packaging of your product C. However, if this is not the case, they may have to change or service product D to keep up with the support provided by your new releases of C.

13.5.2 Dependency on an SPE or Service for Another Base Function

Suppose you have provided a small programming enhancement (SPE) or service for function WD0. This service is packaged as PTF(PD1). You have also developed a new product, which will be packaged as base function WE0. When function WE0 interacts with function WD0, it requires PD1. PD1 is, therefore, a **conditional prerequisite** for function WE0. You define this relationship by coding an ++IF statement for PD1 in function WE0, as shown in Figure 29 on page 155.

Product D	Product E
<pre> ++FUNCTION(WD0) REWORK(1990060). ++VER(Z038). ++JCLIN RELFILE(1). ++MOD(P). ++MOD(Q). </pre>	<pre> ++FUNCTION(WE0) REWORK(1990090). ++VER(Z038). ++IF FMID(WD0) REQ(PD1). ++JCLIN RELFILE(1). ++MOD(R). ++MOD(S). </pre>
<pre> ++PTF(PD1). ++VER(Z038) FMID(WD0). ++MOD(P). </pre>	

Figure 29. Dependency on an SPE or Service for Another Base Function

Note: Any replacement for PTF(PD1) must supersede PD1 to ensure that this requisite for function WE0 is still satisfied.

13.5.3 Cross-Product Service for a Base Function with a Prerequisite

Suppose you need to provide service that affects module Q and module R. Module Q is owned by function WD0, and module R is owned by function WE0. To fix the problem, you need two PTFs, one for each module. PD2 fixes module Q, and PE1 fixes module R. Because the functions may be installed with or without each other, the PTFs for those functions are **conditional corequisites**. You define this relationship by coding an ++IF statement in each PTF, as shown in Figure 30.

Product D	Product E
<pre> ++FUNCTION(WD0) REWORK(1990060). ++VER(Z038). ++JCLIN RELFILE(1). ++MOD(P). ++MOD(Q). </pre>	<pre> ++FUNCTION(WE0) REWORK(1990090). ++VER(Z038). ++IF FMID(WD0) REQ(PD1). ++JCLIN RELFILE(1). ++MOD(R). ++MOD(S). </pre>
<pre> ++FUNCTION(XD0ENU) REWORK(1990060). ++VER(Z038) FMID(WD0). ++JCLIN RELFILE(1). ++PNLENU(DP). ++MSGENU(DM). </pre>	<pre> ++FUNCTION(XE0ENU) REWORK(1990090). ++VER(Z038) FMID(WE0). ++JCLIN RELFILE(1). ++PNLENU(EP). ++MSGENU(EM). </pre>
<pre> ++PTF(PD1). ++VER(Z038) FMID(WD0). ++MOD(P). </pre>	
<pre> ++PTF(PD2). ++VER(Z038) FMID(WD0) PRE(PD1). ++IF FMID(WE0) REQ(PE1). ++MOD(Q). </pre>	<pre> ++PTF(PE1). ++VER(Z038) FMID(WE0). ++IF FMID(WD0) REQ(PD2). ++MOD(R). </pre>

Figure 30. Cross-Product Service for a Base Function with a Prerequisite

13.6 Example 5: Mutually Exclusive Dependent Functions

Suppose function XE1 and function XE2 are dependent functions for the same base function, WE0. Function XE1 tailors the base to one specific environment, and function XE2 tailors it to another specific environment. Because these SYSMODs provide mutually exclusive functions, they are **unconditional negative prerequisites** of each other. You define this relationship by coding the NPRE operand on each function's ++VER statement, as shown in Figure 31.

Product E	
Base Function and Language Support	Dependent Functions and Language Support
<pre> ++FUNCTION(WE0) REWORK(1990090). ++VER(Z038). ++IF FMID(WD0) REQ(PD1). ++JCLIN RELFILE(1). ++MOD(R). ++MOD(S). </pre>	<pre> ++FUNCTION(XE1) REWORK(1990130). ++VER(Z038) FMID(WE0) NPRE(XE2). ++MOD(R). ++MOD(S). </pre>
<pre> ++FUNCTION(XE0ENU) REWORK(1990090). ++VER(Z038) FMID(WE0). ++JCLIN RELFILE(1). ++PNLENU(EP). ++MSGENU(EM). </pre>	<pre> and ++FUNCTION(XE1ENU) REWORK(1990130). ++VER(Z038) FMID(WE0) NPRE(XE2ENU) PRE(XE1,XE0ENU) VERSION(XE0ENU). ++JCLIN RELFILE(1). ++PNLENU(EP). ++MSGENU(EM). </pre>
	<pre> (or) ++FUNCTION(XE2) REWORK(1990130). ++VER(Z038) FMID(WE0) NPRE(XE1). ++MOD(R). ++MOD(S). </pre>
	<pre> and ++FUNCTION(XE2ENU) REWORK(1990130). ++VER(Z038) FMID(WE0) NPRE(XE1ENU) PRE(XE2,XE0ENU) VERSION(XE0ENU). ++JCLIN RELFILE(1). ++PNLENU(EP). ++MSGENU(EM). </pre>

Figure 31. Mutually Exclusive Dependent Functions

13.7 Example 6: Functions Supporting More Than One Language

As shown in the previous sections, any language support you provide for a function should be packaged in a language-support dependent function.

In the course of developing a product, you may decide to provide support for additional languages. For each additional language, the language-sensitive elements should also be packaged in a separate language-support dependent function.

These are some of the situations with relationships you might have to define in the course of developing and servicing dependent functions to support more than one language:

- Supporting two languages for a base function
- Providing PTF service for language-sensitive elements
- Supporting two languages for a base function and its related dependent function
- Providing PTF service for common language-sensitive elements
- Providing PTF service for language-sensitive elements unique to the dependent function

13.7.1 A Base Function Supporting Two Languages

Suppose you have a product that must provide information (such as messages and dialog elements) in both U.S. English and French. The language-sensitive elements for each language must be packaged in a separate dependent function for each language. The remaining elements are packaged in the base function (WX0). Because function WX0 is the base function, it is an **unconditional prerequisite** for the language-support dependent functions (XX0ENU and XX0FRA). You define this relationship by coding the FMID operand on the ++VER statements in each dependent function, as shown in Figure 32 on page 158.

Product X	
Base Function	Language Support
<pre> ++FUNCTION(WX0) REWORK(1990140). ++VER(Z038). ++JCLIN RELFILE(1). ++MOD(U). ++MOD(V). </pre>	<pre> ++FUNCTION(XX0ENU) REWORK(1990140). ++VER(Z038) FMID(WX0). ++JCLIN RELFILE(1). ++PNLENU(XP) DISTLIB(AXXXPENU) SYSLIB(SXXXPENU). </pre>
	<pre> ++FUNCTION(XX0FRA) REWORK(1990140). ++VER(Z038) FMID(WX0). ++JCLIN RELFILE(1). ++PNLFRA(XP) DISTLIB(AXXXPFRA) SYSLIB(SXXXPFRA). </pre>

Note: In this example, DISTLIB and SYSLIB values were specified for panel XP to emphasize that language-sensitive elements should be packaged in unique distribution and target libraries. JCLIN was not necessary.

Figure 32. A Base Function Supporting Two Languages

13.7.2 PTF Service for Language-Sensitive Elements

Suppose you need to correct a mistake that exists in panel XP for both dependent functions (XX0ENU and XX0FRA). You need to provide a separate PTF for each dependent function. Because the dependent functions are independent of each other, no relationship needs to be defined between these PTFs. Because the change only affects language-sensitive elements, no PTF is required for the base function. Each dependent function is an **unconditional prerequisite** for its associated PTF. You define this relationship by coding the appropriate FMID on the ++VER statement for each PTF, as shown in Figure 33.

Product X	
English Support	French Support
<pre> ++FUNCTION(XX0ENU) REWORK(1990140). ++VER(Z038) FMID(WX0). ++JCLIN RELFILE(1). ++PNLENU(XP). </pre>	<pre> ++FUNCTION(XX0FRA) REWORK(1990140). ++VER(Z038) FMID(WX0). ++JCLIN RELFILE(1). ++PNLFRA(XP). </pre>
<pre> ++PTF(PX1). ++VER(Z038) FMID(XX0ENU). ++JCLIN RELFILE(1). ++PNLENU(XP). </pre>	<pre> ++PTF(PX2). ++VER(Z038) FMID(XX0FRA). ++JCLIN RELFILE(1). ++PNLFRA(XP). </pre>

Figure 33. PTF Service for Language-Sensitive Elements

13.7.3 Supporting Two Languages for a Base Function and Its Related Dependent Function

Suppose you have a product consisting of a base function plus a dependent function for an optional enhancement. You want to provide support for messages and dialogs in both English and French for the base function and the dependent function.

The language-sensitive elements for each language should be packaged in a separate dependent function for each language. As shown in Figure 34 on page 160, you need two dependent functions to support the language-sensitive elements for the base function, and two more to support the optional dependent function.

- The base function (WY0) is an **unconditional prerequisite** for its language support functions XY0ENU and XY0FRA. It is also an **unconditional prerequisite** for XY1, its dependent function for the optional enhancement.

You define these relationships by coding the FMID operand on the ++VER statements in XY0ENU, XY0FRA, and XY1.

- XY1 is an **unconditional prerequisite** for its language support functions XY1ENU and XY1FRA.

You define this relationship by coding the PRE operand on the ++VER statements in XY1ENU and XY1FRA.

- Because language-support dependent functions XY0ENU and XY0FRA are applicable to base function WY0, they are unconditional prerequisites for language-support dependent functions XY1ENU and XY1FRA, which are applicable to dependent function XY1.

You define this relationship by coding the PRE operand on the ++VER statements in XY1ENU and XY1FRA.

Product Y	
Base Function and Language Support	Dependent Function and Language Support
<pre> ++FUNCTION(WY0) REWORK(1990250). ++VER(Z038). ++JCLIN RELFILE(1). ++MOD(X). ++MOD(Y). </pre>	<pre> ++FUNCTION(XY1) REWORK(1990250). ++VER(Z038) FMID(WY0). ++MOD(X). ++MOD(Y). </pre>
<pre> ++FUNCTION(XY0ENU) REWORK(1990250). ++VER(Z038) FMID(WY0). ++JCLIN RELFILE(1). ++PNLENU(Y0P). </pre>	<pre> ++FUNCTION(XY1ENU) REWORK(1990250). ++VER(Z038) FMID(WY0) PRE(XY1,XY0ENU) VERSION(XY0ENU). ++JCLIN RELFILE(1). ++PNLENU(Y0P). ++PNLENU(Y1P). </pre>
<pre> ++FUNCTION(XY0FRA) REWORK(1990250). ++VER(Z038) FMID(WY0). ++JCLIN RELFILE(1). ++PNLFRA(Y0P). </pre>	<pre> ++FUNCTION(XY1FRA) REWORK(1990250). ++VER(Z038) FMID(WY0) PRE(XY1,XY0FRA) VERSION(XY0FRA). ++JCLIN RELFILE(1). ++PNLFRA(Y0P). ++PNLFRA(Y1P). </pre>

Figure 34. Supporting Two Languages for a Base Function and Its Related Dependent Function

Note: The VERSION operand is required to change ownership of the elements. See 7.2, “++VER Statement” on page 56 and Chapter 6, “Elements and Load Modules” on page 43 for more information.

13.7.4 PTF Service for Common Language-Sensitive Elements

Suppose you need to provide service that affects panel Y0P. There are four versions of this panel: an English and a French version for the base function, and an English and a French version for the dependent function. Each of these versions is owned by a different dependent function for language-sensitive elements. Therefore, to fix the problem, you need four PTFs, one for each of the dependent functions for language support, as shown in Figure 35 on page 161. You must provide ++IF REQ statements to define the relationship between related PTFs for the same language. However, you do not need to define any relationship between PTFs for different languages.

Product Y	
English Support for the Base Function (WY0)	English Support for the Dependent Function (XY1)
<pre> ++FUNCTION(XY0ENU) REWORK(1990250). ++VER(Z038) FMID(WY0). ++PNLENU(Y0P). </pre>	<pre> ++FUNCTION(XY1ENU) REWORK(1990250). ++VER(Z038) FMID(WY0) PRE(XY1,XY0ENU) VERSION(XY0ENU). ++JCLIN RELFILE(1). ++PNLENU(Y0P). ++PNLENU(Y1P). </pre>
<pre> ++PTF(PY2). ++VER(Z038) FMID(XY0ENU). ++IF FMID(XY1ENU) REQ(PY4). ++PNLENU(Y0P). </pre>	<pre> ++PTF(PY4). ++VER(Z038) FMID(XY1ENU). ++PNLENU(Y0P). </pre>

Product Y	
French Support for the Base Function (WY0)	French Support for the Dependent Function (XY1)
<pre> ++FUNCTION(XY0FRA) REWORK(1990250). ++VER(Z038) FMID(WY0). ++PNLFRA(Y0P). </pre>	<pre> ++FUNCTION(XY1FRA) REWORK(1990250). ++VER(Z038) FMID(WY0) PRE(XY1,XY0FRA) VERSION(XY0FRA). ++JCLIN RELFILE(1). ++PNLFRA(Y0P). ++PNLFRA(Y1P). </pre>
<pre> ++PTF(PY3). ++VER(Z038) FMID(XY0FRA). ++IF FMID(XY1FRA) REQ(PY5). ++PNLFRA(Y0P). </pre>	<pre> ++PTF(PY5). ++VER(Z038) FMID(XY1FRA). ++PNLFRA(Y0P). </pre>

Figure 35. PTF Service for Common Language-Sensitive Elements

13.8 Changing the Contents of Products

After the elements that make up a product have been defined, changes in the contents of the product may be required. For example, an element may need to be added, deleted, combined with another element, or moved to another product. The following sections provide information to help you decide how to make these changes.

13.8.1 Adding Elements

You can add elements to a product using a new release of a base or dependent function, or using a PTF. When a new base or dependent function release adds elements, previous releases of the function are not affected. The new elements are serviced as long as the functions that own them are current. In Figure 36, dependent function XK1 adds module C to product K.

Product K	
Base Function	Dependent Function
<pre> ++FUNCTION(WK0) REWORK(1990170). ++VER(Z038). ++JCLIN RELFILE(1). ++MOD(J). ++MOD(K). ++MOD(T). </pre>	<pre> ++FUNCTION(XK1) REWORK(1990200). ++VER(Z038) FMID(WK0). ++JCLIN RELFILE(1). ++MOD(J). ++MOD(K). ++MOD(C). </pre>

Figure 36. Adding Elements

Notes:

1. The ownership of MOD(J) and MOD(K) is transferred to function XK1. (See 7.2.8, “Defining Ownership (VERSION)” on page 64 for versioning rules.)
2. The ++JCLIN statement and JCLIN data are required to define the revised load module structure.

When a PTF adds elements, it specifies the function that is to own the new elements. If the load module structure is changed, the PTF may also include new JCLIN.

13.8.2 Combining Elements

You can combine elements in a product using a new release of a base or dependent function, or using a PTF. For example, instead of using two modules to provide a given user function, you may combine all the function into one module, and delete the other one.

A new base or dependent function may combine and delete elements that existed in a previous release. However, service must continue to be provided for both versions of the elements during the service currency of the previous release of the product.

In Figure 37 on page 163, dependent function XK2 combines modules A and B, deleting module B from prerequisite dependent function XK1.

Product K	
Base Function	Dependent Functions
<pre> ++FUNCTION(WK0) REWORK(1990170). ++VER(Z038). ++JCLIN RELFILE(1). ++MOD(J). ++MOD(K). ++MOD(T). </pre>	<pre> ++FUNCTION(XK1) REWORK(1990200). ++VER(Z038) FMID(WK0). ++JCLIN RELFILE(1). ++MOD(J). ++MOD(K). ++MOD(C). </pre>
	<pre> ++FUNCTION(XK2) REWORK(1990230). ++VER(Z038) FMID(WK0) PRE(XK1). ++JCLIN RELFILE(1). ++MOD(J) VERSION(XK1). ++MOD(K) DELETE. </pre>

Figure 37. Combining Elements

13.8.3 Migrating Elements by Updating Both Functions

This method is straightforward and is the recommended way of migrating elements from one function to another. The element is deleted from one function and added to another. The new releases of the functions are issued simultaneously and must be installed concurrently. In Figure 38, the new base function release WL1 no longer contains module H, which is now included in the new base function release M1.

Product L	Product M
<pre> ++FUNCTION(WL0) REWORK(1990180). ++VER(Z038) REQ(WM0). ++JCLIN RELFILE(1). ++MOD(F). ++MOD(G). ++MOD(H). </pre>	<pre> ++FUNCTION(WM0) REWORK(1990180). ++VER(Z038) REQ(WL0). ++JCLIN RELFILE(1). ++MOD(J). ++MOD(K). </pre>
<pre> ++FUNCTION(WL1) REWORK(1990280). ++VER(Z038) DELETE(WL0) REQ(WM1). ++JCLIN RELFILE(1). ++MOD(F). ++MOD(G). </pre>	<pre> ++FUNCTION(WM1) REWORK(1990280). ++VER(Z038) DELETE(WM0) REQ(WL1). ++JCLIN RELFILE(1). ++MOD(H). ++MOD(J). ++MOD(K). </pre>

Figure 38. Migrating Elements by Updating Both Functions

13.8.4 Migrating Elements by Using a PTF

A PTF can provide new versions of elements for a function, as well as specify which elements are now owned by that function, and which functions had previously owned those elements. All subsequent releases of the functions affected by the migration must reflect the changes made by the PTF.

Appendixes

Appendix A. Summary of Rules, Restrictions, and Recommendations	167
A.1 Rules	167
A.2 Restrictions	191
A.3 Recommendations	194
Appendix B. MVS Service Packaging Rules	209
B.1 Introduction	209
B.1.1 Service Terminology	210
B.2 MVS Service Packaging Rules	212
B.2.1 PTF Size, Format, and Content	212
B.2.2 Standard PTF Structure	213
B.2.2.1 ++PTF	214
B.2.2.2 ++VER	215
B.2.2.3 ++IF	217
B.2.2.4 ++HOLD	218
B.2.2.5 ++MOVE, ++RENAME, ++DELETE	219
B.2.2.6 ++JCLIN	220
B.2.2.7 ++element	220
B.2.2.8 UCLIN	222
B.2.2.9 Other	222
B.2.3 PTF Cover Letter	222
B.2.3.1 PROBLEM DESCRIPTION(S):	225
B.2.3.2 COMPONENT: or PRODUCT ID:	225
B.2.3.3 APARS FIXED:	226
B.2.3.4 SPECIAL CONDITIONS:	226
B.2.3.5 COMMENTS:	233
B.3 IBM Service Delivery	234
B.3.1 Service Process Initialization	234
B.3.2 PTF Submission	235
B.3.2.1 MVS PTF Control	235
B.4 Naming Conventions for Service	236
Appendix C. Mapping of Old Rule Numbers to New Rule Numbers	237

Appendix A. Summary of Rules, Restrictions, and Recommendations

This appendix lists all the rules, restrictions, and recommendations mentioned in this book.

A.1 Rules

Packaging Rules (RELFILE Tapes)

3.1

- 110. A RELFILE tape can contain only files that can be installed by SMP/E and that meet the requirements for the format and contents of a RELFILE tape.
- 120. All files on an MVS-installable product tape must be SMP/E-installable.

Packaging Rules (RELFILE Tape: Format and Contents)

Section 3.1.1

- 300. The SMPMCS file must be a sequential data set consisting of 80-byte, fixed-length records.
- 400. All the other files on the tape or set of tapes must be relative files for the functions defined in the SMPMCS file.
- 500. All the elements for a function SYSMOD must be on the same logical tape as the SMPMCS file that defines the function.
- 600. There can be only one element with the same name in a given relative file. This includes element names and element alias names.
- 700. Each relative file must contain partitioned data sets that were unloaded in IEBCOPY format.
- 800. Sequential data sets must be packaged as members of a partitioned data set so that they can be unloaded by IEBCOPY into a relative file. A postinstallation job can be provided to copy such an element into a sequential data set. OS/390 Release 7 SMP/E or later can also be used to copy such an element into a sequential data set.
- 900. Modules must be in link-edited format. (This is RECFM=U, undefined record format.) The input parameters used for the link-edited format must include NCAL. Providing modules in link-edited format eliminates the need for the LEPARM operand and other data that is required on the ++MOD statement when modules are provided inline. Contrast with the restriction in 9.3, "General Packaging Rules for JCLIN Data" on page 89 regarding what to do for a PTF that introduces a new ++MOD requiring link-edit parameters other than the default.
- 1000. VSAM data set elements must be in AMS REPRO format.

Packaging Rules (RELFILE Tape: Format and Contents)

Section 3.1.1

- 1100. The partitioned data sets to be unloaded must have a member for each element MCS, plus a directory entry for each ALIAS associated with an element MCS. Likewise, each member in a RELFILE must be defined by an element MCS.
- 1300. If a member in a relative file contains JCLIN data for a SYSMOD, the member name must match the function's FMID.
- 1310. Follow the requirements in Table 11 on page 38 when specifying the MCS statements and data set attributes for elements being packaged in RELFILES.

Packaging Rules (RELFILE Tape: Volume Serial Numbers)

Section 3.1.2

- 1320. If two tapes have the same volume serial number (VOLSER), they must contain the same FMIDs. It is permissible for different SUP levels of the same FMIDs to use the same VOLSER.

Packaging Rules (RELFILE Tape: Data Set Names)

Section 3.1.2

- 1350. The data set name of each relative file must be *rfdsnpfx.sysmod_id.Fnnnn*, where:

rfdsnpfx

is the prefix, if any, for the relative file data set names.

If a prefix is used in the data set names, that value must also be defined by the RFDSNPFIX operand on the header MCS for the SYSMOD. RFDSNPFIX tells SMP/E what prefix to use when allocating the data set names for the relative files being loaded.

If no prefix is used in the data set names, no RFDSNPFIX value should be specified on the header MCS for the SYSMOD.

Note: Do not use "IBM" as the prefix.

sysmod_id

is the FMID of the function to which the file is related.

Fnnnn

is the letter F followed by the number specified on the RELFILE operand of the corresponding MCS statement in the SYSMOD. Do not use leading zeroes in the RELFILE number.

Packaging Rules (Functions)

Section 4.1.1.4

- 2100. All elements included in an MVS-based product that is installed on an MVS system must be SMP/E-installable.

Packaging Rules (Functions)**Section 4.1.1.4**

- 2200. This rule has been changed to a recommendation (see below).
- 2300. This rule has been deleted.
- 2305. All unique elements for a given language must be packaged in a unique SYSMOD for that language.

Packaging Rules (Negative Prerequisite SYSMODs)**Section 4.2.3.3**

- 2330. If two function SYSMODS cannot be installed in the same zone, the MCS of the function SYSMOD with the later availability date must have an NPRES for the other function SYSMOD. If both function SYSMODS have the same availability date, then the MCS for each one must have an NPRES for the other.

Packaging Rules (Installation)**Section 5.1**

- 2600. All files on an MVS-installable product tape must be SMP/E-installable.
- 2700. All products must be packaged so that they can be individually installed using both the RECEIVE-APPLY-ACCEPT method and the RECEIVE-ACCEPT BYPASS(APPLYCHECK)-GENERATE method.
- 2710. The return code from ACCEPT processing for all function SYSMODs must be zero, with these exceptions:
 - Warning message GIM39701W, SYSMOD *sysmod-id* HAS NO ELEMENTS.
 - Warning message GIM50050W, concerning the DESCRIPTION operand.
 - A warning message issued only in certain environments (for example, a product tries to delete an element or load module that is not on the system).

Packaging Rules (++FUNCTION SYSMOD ID)**Section 5.3**

- 2900. A new FMID is required for each new version, release, or modification level of an existing base or dependent function.
- 2950. A new FMID can be added only by a new version, release, or modification level.

**Packaging Rules (Macros, Modules, Source)
Section 5.4**

- 3400. Macros, modules, and source elements must be members of a partitioned data set (DSORG=PO).
- 3410. Distribution libraries must be partitioned; only target libraries may be sequential. Use of sequential distribution libraries would tend to increase the total number of datasets required on the system.
- 3500. The record format (RECFM) for load modules must be U. For more information, see Table 11 on page 38.
- 3510. A product should not change any of the following attributes of an existing dataset:
 - RECFM
 - PDS vs. PDS/E
 - PATH attributes

If such a change is required, a new dataset must be created. It must have a new DDDEF entry as well as a new DDNAME and dataset name.
- 3650. The record format (RECFM) for macros and source must be FB, and the record length (LRECL) must be 80. For more information, see Table 11 on page 38.

**Packaging Rules (Data Elements and hierarchical file system (HFS) Elements)
Section 5.4**

- 3700. Data elements, hierarchical file system (HFS) elements and ++PROGRAM elements must be packaged as members of a partitioned data set (DSORG=PO).
 - 3800. The record format (RECFM) must be F, FA, FM, FB, FBA, FBM, V, VA, VM, VB, VBA, or VBM. The record format (RECFM) of ++PROGRAM elements must be U.
- Notes:**
1. Elements with fixed-length records are not restricted to a logical record length (LRECL) of 80.
 2. A VSAM data set may be a data element if it is in AMS REPRO format. However, after the data is installed by SMP/E, the customer will also have to run an AMS REPRO job to create the original form of the VSAM data. (SMP/E does not support native VSAM data sets as elements.)
- 3810. The maximum LRECL for a data element is 32,654.
 - 3900. Elements with variable-length records may not contain spanned records.
 - 3910. CLISTS must not have sequence numbers.

Packaging Rules (Data Elements and hierarchical file system (HFS) Elements)**Section 5.4**

- 4000. When packaging data elements, use the MCS statements shown in Table 12 on page 45. When packaging hierarchical file system (HFS) elements, use the ++hfs_element MCS statement.

Packaging Rules (Shared Libraries)**Section 5.5**

- 4200. A library cannot contain two or more elements with the same name or alias name, even if they are different types. Therefore, if your product is to be installed in libraries shared with another product, you must ensure that none of your product's elements have the same name or alias name as those for elements of the other product that are installed in the same library.
- 4300. If products share a library, each product must use the same data set attributes for that library. This means that if a product adds elements to an existing product-specific library, the new product must specify the same DCB attributes as the existing library.

Packaging Rules (Elements)**Section 6.1**

- 4910. An element can be owned by only one function. Ownership is defined by the FMID and VERSION operands on the ++VER and element statements.

Packaging Rules (Shared Load Modules)**Section 6.6**

- 5100. This rule has been deleted.
- 5200. One product must not use JCLIN to redefine the content of another product's load module (even for shared load modules). For more information, see 6.1, "General Packaging Rules, Restrictions, and Recommendations for Elements" on page 44.
- 5300. This rule has been deleted.
- 5400. Ensure that the product owner of a module that is shared across products does not use ++MOD DELETE and that it does not change the SYSLIB or DISTLIB of the shared module. For more information about deleting load modules, see 8.3, "Deleting Load Modules (++DELETE)" on page 81.
- 5500. If a module in Product A requires elements from Product B and the products are installed in different zones, the program directory for Product A must define Product B as a prerequisite.

Packaging Rules (Samples)

Section 6.7

- 5750. Sample installation JCL is optional, but if shipped, it must be packaged in a relative file.
- 5810. Products should not ship catalogued or instream procedures to invoke SMP/E during installation. Sample installation jobs should invoke SMP/E directly, and should require the installer to create DDDEF entries for all libraries.
- 5820. If a product installs into the hierarchical file system (HFS), its DDDEF job must include DDDEFs for the /usr/lpp/product_id/vendor_name/ directories it creates or uses. The DDDEF job must create the pathname in the DDDEF, and then provide a separate step to edit the DDDEF and change the path to the user-defined prefix. This is necessary to accommodate long pathnames that are not easily edited by hand.
- 5890. Products must not create DDDEF entries with /etc as a directory in the pathname.

Packaging Rules (++FUNCTION REWORK)

Section 7.1.2

- 5900. REWORK is required on all ++FUNCTION statements, including the initial release.

Specify the REWORK date as *yyyyddd*, which is the year followed by the Julian date (for example, 1995110).

You must change the date every time the function is updated and reissued with the same FMID.

Packaging Rules (Prefix for RELFILE Data Sets)

Section 7.1.3

- 5910. If you specify a value for RFDSNPFX, do not specify RFDSNPFX(IBM). Use a different prefix.

Packaging Rules (++VER)

Section 7.2.1

- 6200. Every SYSMOD referenced on a single ++VER statement must reside in the same zone.
- 6205. When two or more SYSMODs affect the same element, you must specify the relationship among those SYSMODs. Specifically, you must define the order in which they should be processed (indicated by the PRE, SUP, or FMID operand) and the correct version of the element to be installed (indicated by the FMID or VERSION operand).

Packaging Rules (Multiple SYSMODs Affecting an Element)**Section 7.2.1**

- 6210. When two or more SYSMODs affect the same element, you must specify the relationship among those SYSMODs.
 - If both SYSMOD A and SYSMOD B ship the element (or updates to it), the MCS statements in both SYSMODs must define the order in which the SYSMODs should be processed (indicated by the PRE, SUP, or FMID operand) and the correct version of the element to be installed (indicated by the FMID or VERSION operand).
 - If Product A includes an element from Product B via an INCLUDE statement in a JCLIN link-edit step without changing the element, and Product A requires a particular level of Product B, then Product A's MCS statements must specify an unconditional requisite for the appropriate level of Product B.
 - If Product A includes an element from Product B via an INCLUDE statement in a JCLIN link-edit step without changing the element, and multiple levels of Product B would fill the needs of Product A, then Product A's program directory must identify Product B as an installation requirement, specifying the lowest acceptable level of Product B.

Packaging Rules (++VER SREL)**Section 7.2.2**

- 6300. On a single ++VER statement, all SYSMODs specified on the NPRES, PRE, REQ, SUP, and VERSION operands must be applicable to the same SREL as the SYSMOD containing this ++VER statement.
- 6400. You must use one of the following SRELS: Z038 for MVS, C150 for CICS, P004 for NCP, or P115 for IMS and DB2.

Packaging Rules (++VER FMID)**Section 7.2.3**

- 6500. The FMID operand can be used only in a dependent function, not in a base function. The FMID specified in the operand must be the FMID of a base function. Both functions must be applicable to the same SREL. FMID is required for dependent functions.
- 6600. A SYSMOD cannot be both a base function and a dependent function. The FMID operand identifies a SYSMOD as a dependent function; therefore, if you specify the FMID operand, you must include it on all the ++VER statements for the SYSMOD.

Packaging Rules (++VER DELETE) Section 7.2.4

- 6700. If the DELETE operand is used in a base function, it can specify the FMID of a base function or a dependent function. If the DELETE operand is used in a dependent function, it can only specify the FMID of a dependent function.
- 6800. Base functions (other than the initial release) must use ++VER DELETE to delete all previous releases and versions of the product.
Note: Optionally, dependent functions can delete previous releases and versions of the product.
- 6900. A language-support dependent function must not delete an additive dependent function, and vice versa.
- 7000. A function cannot delete itself.

Packaging Rules (Mutually Exclusive Versions) Section 7.2.5

- 7200. If the NPRE operand is used in a base function, it can only specify the FMID of a base function. If the NPRE operand is used in a dependent function, it can specify the FMID of a base function, the FMID of a dependent function, or both. In either case, all functions involved must be applicable to the same SREL.

Packaging Rules (++VER PRE) Section 7.2.6

- 7400. The PRE operand can be used only in a dependent function. It can specify the FMID of a base function (other than its own base) or a dependent function, or it can specify a PTF number. In any case, all functions involved must be applicable to the same SREL.
Note: Do not use the PRE operand in a dependent function to indicate its own base function. You must use the FMID operand for this purpose.
- 7500. The specified prerequisite (or a valid replacement) must be available as long as the specifying SYSMOD is available. When neither the prerequisite function nor the replacement SYSMOD is available, all the functions specifying the prerequisite must be repackaged.
- 7600. If a dependent function specifies a PTF as a prerequisite, the dependent function and the PTF must be applicable to the same base function.

Packaging Rules (++VER SUP)**Section 7.2.7**

- 7700. If the SUP operand is used in a base function, it can specify the FMID of a base function, the FMID of a dependent function, a PTF number, or an APAR number. If the SUP operand is used in a dependent function, it can specify the FMID of a dependent function, a PTF number, or an APAR number. In either case, all functions involved must be applicable to the same SREL.
- 7800. A function must provide all the supported function contained in all the SYSMODs it supersedes.
- 7900. All the superseded SYSMODs must be in the same product as the superseding SYSMOD.
- 8000. For each environment (++VER FMID and SREL), all the elements in the superseded SYSMODs must be contained either in the superseding SYSMOD or in the combination of the superseding SYSMOD and its requisites (other SYSMODs specified on the ++VER REQ operand), unless the element is deleted by the superseding SYSMOD.
- 8100. The environment of a superseded SYSMOD must not be at a higher functional level than the level of the superseding function.
 - If the superseded SYSMOD is a base function, it must apply to the same SREL as the superseding SYSMOD.
 - If the superseded SYSMOD is a dependent function, it must apply to the same SREL as the superseding SYSMOD. In addition, the superseded dependent function must do one of the following:
 - Be applicable to the same base function as the superseding dependent function
 - Be applicable to a lower-level function than the superseding function
- 8200. A new release of a base function can supersede a previous release of that base function only if it also deletes that previous release. Likewise, a new release of a base function can supersede a dependent function applicable to a previous release of that base function only if the new release also deletes that dependent function.
- 8300. A new dependent function can supersede previous releases of that dependent function only if it also deletes those releases.
- 8500. A superseding function (or its requisites) must carry on the SYSMOD relationships defined in the superseded function SYSMODs. Table 14 on page 63 shows the relationships and processing information that the superseding SYSMOD or its requisites may need to include from the superseded SYSMODs.

Note: Table 14 on page 63 also applies to deleting SYSMODs and the information that they or their requisites may need to include from the deleted SYSMODs.

Packaging Rules (Moving and Replacing Elements) **Section 7.2.7**

- 8600. The ++VER statement for each SYSMOD that contains an element that is replaced or moved to a new library must use the PRE or SUP operand to specify the previous SYSMOD, if any, that also replaced or moved that element.

Packaging Rules (++VER VERSION) **Section 7.2.8**

- 8700. You must specify the lower-level function SYSMODs on the VERSION operand of each ++VER statement in the higher-level function SYSMOD.

VERSION is required to establish which elements are functionally higher when SYSMODs for different dependent functions have elements with the same name and type in common. Also, specifying the lower-level function SYSMODs on the VERSION operand on the ++VER statement in the higher-level function SYSMODs ensure that ownership of the elements is given to the highest level SYSMOD.
- 8800. If a dependent function uses the VERSION operand, any subsequent function replacing this dependent function must contain all the elements whose ownership was assumed by the dependent function.
- 9000. A new release of a dependent function can have elements in common with a lower-level dependent function for the same base function. If so, the new release must incorporate those elements and, if the lower-level dependent function is not deleted, must establish the superiority of its version of those elements, as well as its installation relationship with the lower-level function. The superiority of the elements is established by the VERSION operand on either the ++VER or element statement. The installation relationship is established by either the PRE or SUP operand on the ++VER statement. For more information, see the descriptions of these operands elsewhere in this chapter.
- 9100. VERSION must specify all the dependent functions that are functionally lower than the specifying function and that include the elements to be versioned.
- 9200. The VERSION operand must be specified on the ++VER statement if all elements affected by this SYSMOD are to be versioned the same way. The VERSION operand must be specified on the element statement if individual elements can be versioned differently.

Packaging Rules (++IF FMID)**Section 7.3.1**

- 9210. The ++IF statement can be used in a base function or a dependent function. In both cases, the FMID operand can specify either a base function or a dependent function.

- 9300. The function cannot specify its own FMID.

Note: This Rule does not apply to products that require installation using the OS/390 Release 3 (or later) level of SMP/E.

- 9400. This rule has been deleted.

- 9500. If the FMID operand is used in a base function, the specified SYSMOD must be in a previous *version* of the product.

For example, Version 2 Release 2 of a product cannot specify ++IF FMID for Version 2 Release 1; however, it can specify Version 1 Release 3.

Note: A dependent function can specify any function SYSMOD, regardless of whether two functions are part of the same product or product version.

Packaging Rules (++)IF REQ)

Section 7.3.2

- 9600. The REQ operand can be used in a base function or a dependent function. In both cases, the REQ operand can specify either a dependent function or a PTF number.
- 9700. Any dependent function specified on the REQ operand (or a valid replacement) must be announced and must be available as long as the specifying SYSMOD is available.
- 9800. If the specified conditional requisite is a function and it is deleted by a new release of that function, one of the following must be done:
 - The new release can also supersede the specified requisite function. This way, the function specifying the requisite does not need to be repackaged.
 - If the specified requisite function is to be deleted by a new release without also being superseded, the specifying function must be repackaged and redesigned to refer to the new release as the requisite.
- 9900. If the specified conditional requisite is a PTF, any subsequent replacement must supersede the specified PTF. This eliminates the need to repackage the specifying function to redefine the conditional requisite.
- 10000. A SYSMOD cannot specify both a conditional and unconditional relationship for the same SYSMOD ID.

For example, the following statements cannot appear in the same SYSMOD:

```
++VER REQ(ABC1234) .  
++IF FMID(Z) REQ(ABC1234) .
```

Note: This Rule does not apply to products that require installation using the OS/390 Release 3 (or later) level of SMP/E.

- 10010. If the specified SYSMOD is a dependent function, the FMID to which it applies must either:
 - Match the FMID specified on the associated ++IF statement contained in the specifying SYSMOD
 - Unconditionally coexist with the FMID specified on the associated ++IF statement contained in the specifying SYSMOD

Packaging Rules (DISTLIB for Elements)**Section 7.5**

- 10100. Do not use SYSPUNCH as the DISTLIB. It is used by SMP/E and other products to process assembled modules.
- 10110. Do not specify a pathname in a hierarchical file system (HFS) as the DISTLIB.
- 10111. Do not specify SMP/E temporary data sets (SMPLTS, SMPMTS, SMPPTS, SMPSTS, etc.) as DISTLIB or SYSLIB values on MCS.
- 10112. If you must use a new library, it must have a unique ddname and a unique data set name to avoid conflicts with other products. For more information on naming distribution libraries, see 10.4, “Library Names” on page 124.

Packaging Rules (UCLIN)**Chapter 8**

- 10115. Make sure the UCLIN data can be processed using SMP/E. Provide instructions for SMP/E customers to insert the appropriate SET BOUNDARY command before the UCLIN data. SMP/E needs the SET command to update the correct zone with the UCLIN data.
- 10117. Package the UCLIN data as an element that is installed in an appropriate data set for sample code. This allows customers, as well as product installation procedures, to have access to the UCLIN. You can package the data as sample code using the ++SAMP MCS statement. Use standard names for the element, the target library, and the distribution library.
- 10119. Describe any UCLIN data requirements and procedures in the installation instructions. This documentation must provide enough information so that the customer can either invoke SMP/E or use the SMP/E dialogs to process the UCLIN data.

You must use a ++HOLD statement, even if your documentation clearly and fully explains how to handle the UCLIN.

Packaging Rules (++)MOVE)

Section 8.1

- 10200. A dependent function can contain a ++MOVE statement for an element or load module it does not contain only if the element or load module is owned by the base function to which the dependent function applies, or by another dependent function for the same base function. In either case, the moving dependent function must specify the owning function as a prerequisite.

If a previous dependent function has performed a ++MOVE on the element or load module, then the new dependent function must specify that dependent function as a prerequisite.

- 10300. A function can contain only one ++MOVE statement for a given element.
- 10400. A function can contain no more than two ++MOVE statements for a given load module, one for each SYSLIB defined for the load module.
- 10500. All MCS statements following the ++MOVE statements and referring to the elements or load modules that were moved must reflect the new libraries for those elements or load modules. All SYSMODs applied subsequent to the move must reflect the new libraries for those elements or load modules.
- 10600. All changes caused by a ++MOVE MCS must also be specified in any JCLIN and SYSGEN macros that refer to the moved member.
- 10700. If SYSMOD(1) defines or moves an element, subsequent SYSMODs containing that element must specify SYSMOD(1) as a prerequisite.
- 10800. If SYSMOD(1) moves a given load module using a ++MOVE statement, any SYSMOD that supersedes SYSMOD1 must also contain the ++MOVE statement.
- 10900. If an element or load module to be moved to a new SYSLIB is a member of a totally copied library, the moving function must also move the same element or corresponding module to a new distribution library.

Packaging Rules (++)RENAME)**Section 8.2**

- 11000. A dependent function can contain a ++RENAME statement for a load module associated with either the base function to which it applies, or with another dependent function that is applicable to that same base function and that is required by the function containing the ++RENAME statement.
- 11100. All changes caused by a ++RENAME MCS must also be specified in any JCLIN and SYSGEN macros that refer to the old name of the load module.
- 11200. A function can contain only one ++RENAME statement for a given load module.
- 11300. If SYSMOD(1) renames a given load module using a ++RENAME statement and SYSMOD(2) defines that load module under its new name with JCLIN data, SYSMOD(2) must specify its relationship to SYSMOD(1) using the PRE, DELETE, or SUP and DELETE operands on its ++VER statement.
- 11400. If SYSMOD(1) defines a given load module and SYSMOD(2) renames that load module using a ++RENAME statement, SYSMOD(2) must specify its relationship to SYSMOD(1) using the PRE operand on its ++VER statement.
- 11500. If a load module being renamed was totally copied from a distribution library into a target library (defined by JCLIN data as a totally copied load module), this function must also use a ++MOVE statement to move the identically named element (++)MOD) to a new distribution library.
- 11600. If a dependent function is renaming a load module, that function must refer to the last previous lower-level dependent function (if any) that (1) moved the load module being renamed or (2) renamed a load module to the name of the load module being renamed again.
 - If that previous dependent function moved the load module being renamed, this dependent function can either delete or supersede and delete that dependent function, or specify it as a prerequisite.
 - If the previous dependent function renamed a load module to the name of the load module being renamed again, this dependent function must specify that previous dependent function as a prerequisite.

Packaging Rules (++)DELETE)

Section 8.3

- 11700. A dependent function can contain a ++DELETE statement for a load module associated with either the base function to which it applies, or with another dependent function that is applicable to that same base function and that is required by the function containing the ++DELETE statement.
- 11800. A function can contain only one ++DELETE statement for a given load module.
- 11900. A function containing a ++DELETE statement must also include the appropriate changes for its JCLIN or SYSGEN macros (if any) to reflect the change.
- 12000. If SYSMOD(1) deletes a given load module using a ++DELETE statement and SYSMOD(2) defines that load module with JCLIN data, SYSMOD(2) must specify its relationship to SYSMOD(1) using the PRE, DELETE, or SUP and DELETE operands on its ++VER statement.
- 12100. If SYSMOD(1) defines a given load module with JCLIN data and SYSMOD(2) deletes that load module using a ++DELETE statement, SYSMOD(2) must specify its relationship to SYSMOD(1) using the PRE or FMID operand on its ++VER statement.
- 12200. A dependent function that is deleting a load module must refer to the last previous lower-level dependent function (if any) that (1) moved the load module being deleted or (2) renamed a load module to the name of the load module being deleted.
 - If that previous dependent function moved the load module being deleted, this dependent function can either delete or supersede and delete that dependent function or specify it as a prerequisite.
 - If that previous dependent function renamed a load module to the name of the load module being deleted, this dependent function can either delete or supersede and delete that dependent function or specify it as a prerequisite.
- 12210. If a SYSMOD is deleting an alias for a load module but not the load module itself (ALIAS is specified on the ++DELETE statement), you must reflect this change using JCLIN. To do this, include a ++JCLIN statement with JCLIN data that contains 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 CSI.

Packaging Rules (DELETE for Elements)**Section 8.4**

- 12300. A dependent function must not delete a macro or source element from a lower-level function (its parent base function or a lower-level dependent function for the same parent base function), because a PTF that is applicable to the lower-level function may need to update the element (such as by using ++MACUPD or ++SRCUPD). If that element were deleted, there would be nothing to update, and the PTF needed for the lower-level function could not be installed.
- 12310. This rule has been deleted.

Packaging Rules (++MOD CSECT)**Section 8.5**

- 12400. If a SYSMOD changes the CSECTs contained in an existing module, CSECT must be specified and must list all the CSECTs in that module. This is true even if the module now contains only one CSECT whose name matches the module name on the ++MOD statement.
- 12500. This rule has been deleted. It has been replaced by rule 131.3 in 9.6, "Link-Edit Steps" on page 94.

Packaging Rules (VERSION for Elements)

8.6

- 12510. VERSION is required to establish which elements are functionally higher when SYSMODs for different functions have elements with the same type and name in common. You must specify the lower-level function in the VERSION operand of the element statement in the SYSMOD associated with the higher-level function.
- 12600. The specified functions must be able to coexist with the specifying SYSMOD.
- 12700. The specified functions must contain the element described by the element statement.
- 12800. For dependent functions, VERSION must specify all the dependent functions that are functionally lower than the specifying function and include the element being versioned.
- 12900. If VERSION is also specified on a ++VER statement for this SYSMOD, the VERSION operand on the element statement overrides the VERSION values specified on the ++VER statement. However, the VERSION operand on the element statement is not additive; it does not automatically take over ownership from the functions specified on the ++VER VERSION operand. To take over ownership from any of the functions specified on the ++VER VERSION operand, you must repeat those values on the VERSION operand for the element statement.
- 13000. The VERSION operand must be specified on the element statement if individual elements may be versioned differently. The VERSION operand must be specified on the ++VER statement used if all elements affected by this SYSMOD are to be versioned the same way.

Packaging Rules (JCLIN Data)

Section 9.3

- 13100. The combination of JCLIN data and element statements must completely describe all the elements in the function and their target and distribution libraries.
- 13110. A product's installation must not require the editing of the JCLIN.
- 13200. If the low-level qualifier of a data set name is in the format *xccczzzz*, as described in rule 140 in 10.4, "Library Names" on page 124, the low-level qualifier and the ddname must be identical.

NOTE: Since a DDNAME may refer to a subdirectory in the hierarchical file system (HFS), several DDNAMEs may point into one HFS. In these cases, the low-level qualifier and the ddname need not be identical.
- 13300. Input data sets in link-edit steps must not be concatenated.

An exception to this rule is using the support for the automatic library call facility. For more information, see the description of the SYSLIB DD statement in 9.6.2, "Link-Edit Control Statements" on page 97.

Packaging Rules (JCLIN Assembler Steps)**Section 9.4**

- 13400. Assembler steps must be identified by one of the following:
 - EXEC PGM=IFOX00
 - EXEC PGM=IEV90
 - EXEC PGM=ASMA90
 - EXEC PGM=ASMBLR
 - EXEC ASMS

Packaging Rules (JCLIN Copy Steps)**Section 9.5**

- 13500. Copy steps must be identified by the following:
 - EXEC PGM=IEBCOPY
- 13600. The RENAME function must not be used in JCLIN.
- 13700. If the SELECT MEMBER= statement is used to selectively copy elements, the COPY INDD=xxx,OUTDD=xxx control statements for selectively copied elements must include the comment TYPE=xxxx. The format of the TYPE comment on the COPY statement is:

```
COPY INDD=ddname,OUTDD=ddname TYPE=xxxx
```

where xxxx is MOD, MAC, SRC, or DATA.

Notes:

1. If the TYPE=xxxx parameter is not specified, the default used by SMP/E is TYPE=MOD.
2. TYPE=DATA is used for data elements.

Without this additional comment, the GENERATE command cannot determine what type of element is being copied. If the comment is not included, SMP/E assumes the element is a module and may create unnecessary module entries in the target or distribution zone.

For data elements and hierarchical file system (HFS) elements, you must use the SYSLIB and DISTLIB operands on the element statement to specify information used to install the element. During JCLIN processing, SMP/E bypasses any COPY SELECT statements that specify TYPE=DATA.

- 13800. The SELECT statement can specify either the name of the member to be copied or an alias name for the member. The SELECT statement for an alias must specify the comment "ALIAS OF *member*", where *member* is the member name for which *alias* is an alias.
- 13900. A SELECT statement that identifies an alias can specify only one name on the MEMBER operand.

Packaging Rules (JCLIN Copy Steps) **Section 9.5**

- 13910. If a ++MOD on a product tape defines either (1) a complete load module containing single or multiple CSECTs or (2) a partial load module containing multiple CSECTs, any ++MOD by the same name shipped in a subsequent PTF must also be the same type of load module (complete load module or multi-CSECT partial load module). If a CSECT shipped in the original ++MOD is not shipped in the replacement ++MOD, it will no longer exist.

To replace part of a copied ++MOD, the PTF must convert the ++MOD into a link-edited load module by splitting it into smaller serviceable parts, as follows:

1. Delete the original ++MOD with a ++MOD DELETE.
2. Ship a new ++MOD for each of the parts into which the original ++MOD has been split.
3. Provide link-edit JCLIN to define the link edit structure of the resulting load modules.

All future maintenance that affects the load module or any of its parts must explicitly or implicitly specify this PTF as a prerequisite.

Packaging Rules (JCLIN Link-Edit Steps) **Section 9.6**

- 14000. Link-edit steps must be identified by one of the following:
 - EXEC PGM=IEWL
 - EXEC PGM=HEWL
 - EXEC PGM=IEWBLINK
 - EXEC LINKS
- 14100. Link-edit steps must not be sensitive to the order of execution of other link-edit steps, either in the same FMID or in another FMID. Link-edit steps must also not be sensitive to the order of execution of the individual load module builds within the step.
- 14200. No elements to be included in a JCLIN link-edit step can be derived from the output of another JCLIN link-edit step, or from the output of a load module build within the same JCLIN link-edit step.
- 14210. Never specify a JCLIN link-edit step to indicate that a load module resides in the SMPLTS library.

SMP/E automatically link-edits a base version of any load module with a CALLLIBS subentry into the SMPLTS library.
- 14220. Do not specify a pathname in a hierarchical file system (HFS) as the distribution library.
- 14230. All INCLUDE statements in link-edit JCLIN data should specify the included module's distribution library, or SYSPUNCH if it is an assembled module. Do not use data sets such as SYSLIB or SYSLMOD.

Packaging Rules (JCLIN Link-Edit Steps)**Section 9.6**

- 14240. If a load module consists of more than one distribution library module, use an ENTRY statement; otherwise, the entry point of the load module might change each time the load module is relinked by SMP/E.
- 14250. If a specific order of CSECTs within a load module is required, use ORDER statements to define the load module structure.
- 14260. If Product A uses CALLLIBS to indicate libraries created by Product B:
 1. The SYSLIB DD statement in Product A's JCLIN must use the real DDNAME of the library.
 2. If Product A does not require Product B to exist in the same zone, Product A's DDDEF job must create a DDDEF entry for the library with its real DDNAME, using the ADD DDDEF command to avoid possible contamination of an existing DDDEF entry.
 3. If it is possible that the library does not exist on the system, the DDDEF job must instruct the customer to point either to the actual dataset (if it exists) or to an empty dataset. The job must not give the customer a choice of two or more legitimate datasets for one DDDEF. NOTE: The product may not allocate an empty dataset for this purpose.
- 14270. If a product documents in its installation documentation that a return code of 8 is acceptable from APPLY, then RC=8 must be coded on the NAME statement in the JCLIN for the appropriate load modules. This may be the case if the product uses a CALLLIBS library to obtain load modules created by an optional function.

Packaging Rules (Element and Load Module Names)

Section 10.3

- 14300. All products must use the assigned, unique three-character component code as the first three characters of the element names. The first character of the component code follows the conventions shown below to avoid naming conflicts with elements provided by IBM or by other user-written software.

Value	Meaning
--------------	----------------

A–I	When used by IBM, all three characters of the prefix are generally alphabetic (with some exceptions).
-----	---

	Can be used by non-IBM products only if the prefix includes at least one numeric or national character.
--	---

J–Z	Available only for non-IBM products. The prefix can be all alphabetic or can include numeric or national characters.
-----	--

- 14400. Two elements with the same element type cannot have the same name—element names must be unique. This is true regardless of whether the elements are in the same product or in different products. For more information, see 6.1, “General Packaging Rules, Restrictions, and Recommendations for Elements” on page 44.
- 14500. Load modules should have unique names, which should begin with the product’s assigned three-character prefix. However, the same load modules having the same attributes can be defined to two load libraries.
- 14600. Like-named elements, including aliases, must be in separate target and distribution libraries. These libraries must be in separate RELFILES. This prevents unintentional overlaying of elements.
 - See 6.1, “General Packaging Rules, Restrictions, and Recommendations for Elements” on page 44 for information about restrictions on like-named elements.
 - See 3.1.1, “Format and Contents of the RELFILE Tape” on page 12 for additional rules and requirements concerning RELFILES.
- 14700. If more than one version of a product is intended to coexist in the same zone, the element and load module names must be unique for each version.

Packaging Rules (Library Names)**Section 10.4**

- 14900. The low-level qualifier of the name of a new distribution or target library must be unique. You must use your registered component code (*ccc*) in your library name. You should also distinguish between your distribution libraries and your target libraries to make it easier for customers to identify your libraries. One way to do this is to use the format *xccczzzz*, where:
 - *x* is the letter for a distribution library or a target library.
 - *ccc* is the component code (the first three characters assigned to the elements).
 - *zzzz* is whatever the product developer chooses to use, to keep the name unique.

Exception: A data set name need not conform to this format if *all* of the following are true:

1. The data set name is required to have a non-conforming low-level qualifier for unavoidable technical reasons. (For example, C language header file data sets are required by the C compiler to use the low-level qualifier of “H.”)
2. The data set is *not* specified as a target library in any JCLIN data, either on a SYSLMOD DD statement or on an EXEC statement.
3. The data set name is *not* specified in a SYSLIB concatenation in any JCLIN data.

Data sets that qualify under this exception must still use ddnames with the format *xccczzzz* or *xcccczzzz* as defined above, to comply with rule14910.

- 14910. Every target and distribution library must have a unique ddname.
- 15000. See rule 119 in 9.3, “General Packaging Rules for JCLIN Data” on page 89.

Packaging Rules (Library Names)

Section 10.4

- 15010. If a data set whose name does not use the *xccczzzz* format is renamed for any reason, the low-level qualifier of the new data set name must use the *xccczzzz* format, and the data set's new ddname must match the new low-level qualifier.

Exception: A data set name need not conform to this format if *all* of the following are true:

1. The data set name is required to have a non-conforming low-level qualifier for unavoidable technical reasons. (For example, C language header file data sets are required by the C compiler to use the low-level qualifier of "H.")
2. The data set is *not* specified as a target library in any JCLIN data, either on a SYSLMOD DD statement or on an EXEC statement.
3. The data set name is *not* specified in a SYSLIB concatenation in any JCLIN data.

Data sets that qualify under this exception must still use ddnames with the format *xccczzzz* or *xcccczzz* as defined above, to comply with rule 14910.

- 15100. A product's execution must not depend on the high-level qualifier of any data set names. Product code should refer only to ddnames.

Packaging Rules (Language-Sensitive Elements)

Chapter 11

- 15200. This rule has been changed to a recommendation (see below).
- 15350. Language variants of an element may have the same name for programming access. In this case, package each language variant as a different element type using the same name for each variant. However, because the names are the same, you must assign the elements to different libraries. See Figure 6 on page 47 for an example.
- 15410. This rule has been deleted.

Packaging Rules (Language Abbreviations)

Section 11.1

- 15500. When the data element or hierarchical file system (HFS) element MCS indicates the language being supported, use one of the national language identifiers shown in Table 18 on page 129 as the three-character suffix for the element type.
- 15600. Each language variant of an element type constitutes a distinct element type, and rules applying to element types apply to every such variant. For example, ++PNLENU and ++PNLDEU are two different element types.

Packaging Rules (Libraries)**Section 12.1.1**

- 15700. This rule has been deleted.
- 15800. This rule has been deleted.
- 15810. If a ++PROGRAM element is prebound with parts from another product, it must use the lowest supported level of the borrowed parts, and must require that level or higher as a functional (noninstallation) requisite. This will avoid problems in customer environments with varying levels of the product.

Packaging Rules (Hierarchical File System)**Section 12.4**

- 18810. Symbolic links must not exist in the /tmp, /dev, /var or /etc directories.
- 18820. Products must not install anything directly into the /etc directory during APPLY processing; the /etc directory is used only for customization data. Shell scripts invoked by SMP/E must not install or change files in the /etc directory.
- 18830. Permission bits for every file or directory in the hierarchical file system (HFS) must be User >=6, Group >= 4, and Other >= 4. Directories containing SMP/E-installed files must be User=7, Group=5, Other=5.

A.2 Restrictions

IBM Software Delivery Restrictions

Chapter 3

The **IBM Software Delivery Solutions** process supports only functions packaged in relative files that can be processed with SMP/E. Functions packaged in TXLIB or LKLIB data sets are not supported.

End of IBM Software Delivery Restrictions

IBM Software Delivery Restrictions

Section 3.1.1

- Sequential data sets may not be used as target libraries. As long as this restriction exists, post-APPLY jobs may be provided to copy elements into a sequential data set. This is a restriction of the **IBM Software Delivery Solutions** process.
- RELFILEs containing a RECFM=U must specify a BLKSIZE of 6144, so that they can be reblocked upwards at installation.
- ++MOD elements must not contain linkage editor ALIAS statements inline.

End of IBM Software Delivery Restrictions

IBM Software Delivery Restrictions

Section 3.1.2

- RECFM=U RELFILEs must be blocked at 6144, so that they can be reblocked upwards at install time.
- ++MOD elements must not contain linkage editor ALIAS statements inline.

End of IBM Software Delivery Restrictions

IBM Software Delivery Restrictions

Section 6.1

- The TXLIB and LKLIB operands are not supported by the **IBM Software Delivery Solutions** process.
- The **IBM Software Delivery Solutions** process does not support SYSMODs requiring assemblies at ACCEPT time. Therefore, the following restrictions apply to the ++MAC and ++SRC statements:
 - A ++MAC statement must not specify the ASSEM, DISTMOD, DISTOBJ, or PREFIX operands unless the macro is accompanied by the assembled object modules and ++MOD statements.
 - A ++SRC statement must not specify the DISTMOD or DISTOBJ operand. In addition, the source can be accompanied by the assembled object module and ++MOD statement.

End of IBM Software Delivery Restrictions

IBM Software Delivery Restrictions

Section 6.6

The following restrictions apply if you do not require SMP/E Release 7 or later as the minimum level of SMP/E for installing your product:

- SMP/E can be used to manage shared load modules only if all the modules that comprise the load module are in the same zone.
- Postinstallation jobs are required if the shared load module is comprised of modules from more than one zone.

Note: If postinstallation jobs (outside SMP/E) are used, service that is applied to modules used by the postinstallation job is not applied to the shared load module unless the postinstallation job is rerun after the maintenance is applied.

End of IBM Software Delivery Restrictions

IBM Software Delivery Restrictions

Chapter 7

The ++ ASSIGN statement is not described in this section and must not be used.

End of IBM Software Delivery Restrictions

IBM Software Delivery Restrictions

Section 7.4

The CLASS operand must not be used.

++HOLD statements are not permitted in function SYSMODs. This is a restriction of the **IBM Software Delivery Solutions** process.

End of IBM Software Delivery Restrictions

IBM Software Delivery Restrictions

Section 8.1

The ++MOVE statement is not allowed for data elements or hierarchical file system (HFS) elements. This is an SMP/E restriction.

End of IBM Software Delivery Restrictions

IBM Software Delivery Restrictions

Section 9.1

The **IBM Software Delivery Solutions** process does not support the TXLIB operand. Therefore, it is not included in the list of ++JCLIN operands, nor is it described in the sections that follow.

End of IBM Software Delivery Restrictions

IBM Software Delivery Restrictions

Section 9.3

- If a PTF introduces a new ++MOD requiring link-edit parameters other than the default, these parameters must be specified in the LEPARM operand of the ++MOD statement. Parameters specified in JCLIN data are not sufficient. This is a restriction of the **IBM Software Delivery Solutions** process.
- Products that require assemblies during APPLY processing may not require macro libraries provided by products in another SREL. This is a restriction of

the **IBM Software Delivery Solutions** ServerPac process; the macro libraries will not be available during order build processing.

_____ End of IBM Software Delivery Restrictions _____

_____ IBM Software Delivery Restrictions _____

Section 9.6.2

- For the **IBM Software Delivery Solutions** process to operate correctly, a product must not contain a CHANGE statement in a link-edit step.
- A product that uses CALLLIBS must not use the RC= parameter on a NAME statement in its JCLIN unless one of the following is true:
 1. The product using CALLLIBS identifies the product providing the CALLLIBS libraries as a requisite (REQ) in its SMPMCS.
 2. The product using CALLLIBS ships stubs for the elements linked with CALLLIBS.

This is a restriction of the **IBM Software Delivery Solutions** process; the CALLLIBS libraries may not be available during order build processing unless one of the above criteria is met.

_____ End of IBM Software Delivery Restrictions _____

A.3 Recommendations

_____ Packaging Recommendations _____

Section 3.1.1

Modules should be single-CSECT load modules.

_____ End of Packaging Recommendations _____

_____ Packaging Recommendations _____

Section 3.1.2

The data set name of the first file should be *SMPMCS*.

_____ End of Packaging Recommendations _____

_____ Packaging Recommendations _____

Section 4.1.1.4

Common elements should be packaged in a common SYSMOD.

_____ End of Packaging Recommendations _____

Packaging Recommendations

Section 4.2.3.2

When you are building SYSMODs that may be installed as a group, such as prerequisite or corequisite SYSMODs, do not construct the SYSMODs in such a way that their proper installation depends on the internal processing order within SMP/E. From time to time, the processing order may be changed and SYSMODs that depend on that order may not be installed correctly. Follow the packaging rules in this book to define how the SYSMODs should be installed.

End of Packaging Recommendations

Packaging Recommendations

Section 4.2.3.4

- A new release of a function should both delete and supersede the previous release if all of the following are true:
 - The new release contains at least all the function that was in the previous release.
 - If other products specified the deleted function as a requisite, all the internal and external interfaces used by those other products are unchanged in the new release.
 - Other products that specified the previous release as a requisite can run with the new release.
- Evaluate a replacement function using Table 10 on page 28 as a guide. If the replacement function matches that description, then the preferred and recommended way to replace the previous function is to both delete and supersede it.

End of Packaging Recommendations

Packaging Recommendations

Section 4.2.4

Products should not require the customer to install them into their own unique zones; every product should be installable in the same target and distribution zones as any other product in the SREL. This gives the customer the ability to decide which combinations of products will reside together.

The installation information may suggest that the customer use new zones initially to avoid deleting previous releases of the product, but this should not be required.

Products should not require any function or service to be accepted before another function can be applied.

End of Packaging Recommendations

Packaging Recommendations

Section 5.1

A PTF should not increase its product's driving system requirements beyond what is documented in the installation instructions.

End of Packaging Recommendations

Packaging Recommendations

Section 5.4

- A dependent function should contain only those elements needed to provide the additive function, or that are needed to provide the additional language support. This reduces the number of element versions and makes servicing the elements easier.
- Sample job streams and other special data that might be helpful to the customer can be stored as a member of a partitioned data set that is unloaded to a relative file on the RELFILE tape. Examples include:
 - A procedure to allocate and catalog libraries
 - Installation verification procedures (IVPs)

This data should be packaged as sample code using the ++SAMP MCS statement, and it should be defined to be installed in an appropriate data set for sample code.

When SMP/E installs the SYSMOD, it copies this member into the libraries specified by the SYSLIB and DISTLIB operands on the element MCS statement. The sample job stream or other data can then be retrieved from the appropriate library for further processing.

- Jobs allocating target or distribution libraries must specify BLKSIZE=32760 for all RECFM=U datasets, and BLKSIZE=0 (utilizing system-determined block sizes) for all non-RECFM-U datasets, with the following exceptions:
 - SYS1.UADS
 - Font libraries

End of Packaging Recommendations

Packaging Recommendations

Section 5.4

- If a function SYSMOD uses unique target or distribution libraries, you may want to include a procedure to allocate and catalog the libraries. This procedure should be in an appropriate data set for sample code, must be a member in one of the relative files, and must be defined by the appropriate element MCS, as described above.
- A product may have an Installation Verification Procedure (IVP) that may be used by customers to verify that the product has been installed. If an IVP is

included in the product package, it should be in an appropriate data set for sample code, must be a member in one of the relative files, and must be defined by the appropriate element MCS.

_____ End of Packaging Recommendations _____

_____ Packaging Recommendations _____

Section 5.5

To avoid problems with like-named elements or aliases, do not install your product in shared libraries.

_____ End of Packaging Recommendations _____

_____ Packaging Recommendations _____

Section 5.6

Do not use UCLIN. Use MCS statements instead.

Note: UCLIN is acceptable, and recommended, to create or modify DDDEF entries.

_____ End of Packaging Recommendations _____

_____ Packaging Recommendations _____

Section 6.1

Single-CSECT modules are recommended where possible. This makes it easier for the module to be serviced. A single CSECT can be distributed rather than shipping the entire module. SMP/E can perform a CSECT replacement.

_____ End of Packaging Recommendations _____

_____ Packaging Recommendations _____

Section 6.6

- If a shared module is loadable and is used by more than one product, then products that share modules should dynamically load the shared modules during initialization and then link (or branch) to it as needed (there are performance considerations). This way, the latest level of the module is used without having to link-edit the module every time it is serviced.
- If a module is link-edited into a known existing load module and does not require link edit control statements (such as ENTRY, ALIAS, and ORDER), the ++MOD LMOD operand should be used.

_____ End of Packaging Recommendations _____

Packaging Recommendations

Section 6.7

Products should supply sample jobs to perform the SMP/E APPLY and ACCEPT functions.

Products should ship a sample job to allocate any target or distribution libraries that are created by the product, and should require the installer to run it. If any of these libraries are shared libraries that may have been allocated by other products, such libraries should be allocated in a separate job or job step, with instructions to the user explaining when the job or job step should be run.

Products should ship a sample job to create DDDEF entries for new target and distribution libraries, as well as any existing libraries that may not have entries in this product's target or distribution zone. Entries for all distribution libraries should be created in the distribution zone, and entries for all target libraries should be created in the target zone. In addition, entries for all distribution libraries should be created in the target zone to support RESTORE processing.

The installation instructions should identify the names of the sample jobs and in which RELFILE they reside, so that customers can download the jobs directly from the tape, if desired. The installation instructions should also state that the customer can perform an SMP/E RECEIVE to load the jobs into temporary libraries, copy them into private data sets, and then modify and run the jobs from these data sets.

If Function A requires Function B with the FMID, PRE, or REQ operands, and Function A uses Function B's libraries, then Function A is not required to ship allocation or DDDEF jobs for any libraries allocated by Function A.

Sample jobs should include clear and detailed comments. Information necessary to update the job prior to submission should be in the job, not in the installation instructions for the product.

If a sample job is provided on the tape, the text of the job should not appear in the installation instructions. This will reduce the size of the installation instructions, as well as avoid situations where the tape and the installation instructions do not match.

DDDEF jobs should adhere to the following:

1. Use ADD DDDEF, not REP DDDEF.
2. Use the WAITFORDSN operand.
3. Use separate job steps to divide datasets into logical groups. For example, a product could use one step for new datasets, and other steps for datasets introduced in previous releases.

Middle-level qualifiers of VxRxMx should not be specified in sample allocation jobs.

Symbolic links for hierarchical file system (HFS) files should be created in the MKDIR EXEC, and should be relative, not absolute. In order to ensure that the MKDIR EXEC can run multiple times without damage, products creating symbolic

links in the MKDIR EXEC should also provide UNLINK statements for every symbolic link ever created in this or previous levels, including those that have become obsolete.

The MKDIR EXEC should be called zzzMKDIR, and the jobname of the JCL invoking it should be called zzzISMKD, where zzz is the three-character prefix of the product shipping the elements.

A PTF should not add or delete DDDEF entries, or change dataset or pathnames in a DDDEF entry. If this is unavoidable, the following is required:

1. A ++HOLD ACTION should be placed on the PTF.
2. The changes should be shipped in a separate DDDEF or MKDIR EXEC shipped in the PTF, not by updating and reshipping the existing DDDEF or MKDIR EXEC.
3. The new DDDEF or MKDIR job should appear in the HOLDDATA of the PTF.

Sample allocation jobs should specify UNIT=SYSALLDA for all target and distribution libraries.

Sample DDDEF creation jobs should specify UNIT(SYSALLDA) for all target and distribution libraries.

All products installing into the hierarchical file system (HFS) should statically create their directories in a MKDIR EXEC. The installation documentation should document how to run the exec during the installation of the product, similar to the documentation on running dataset allocation jobs.

The MKDIR EXEC should meet the following requirements:

1. It accepts a parameter for the highest-level directory, rather than hard-coding it.
2. Output is sent to the SYSOUT held queue. It contains a report of what was created, what was not created, and what directories already existed. It also includes the return code received and the date and time it was run.
3. The directory names all appear together.
4. It can be executed multiple times successfully.
5. If a product provides EXECs run during installation, such as MKDIR EXECs, a batch job invoking the EXEC should be provided for the customer's use. This does not apply to EXECs run after installation, such as IVPs or customization.

Products should not use SMP/E's dynamic allocation function to allocate target and distribution libraries as new data sets; the usage of DDDEFs is only recommended after the datasets have been allocated outside of SMP/E.

End of Packaging Recommendations

Packaging Recommendations

Section 7.1.4

The copyright statement should be within the first 15000 bytes of object code on the distribution media.

_____ End of Packaging Recommendations _____

_____ Packaging Recommendations _____

Section 7.2.4

- You should specify additive dependent functions that are applicable to a deleted base function. This allows customers to determine what is deleted by a function by reading the associated MCS. (Specifying these functions is for documentation purposes only. Dependent functions are automatically deleted when the associated base functions are deleted.)
- It is not necessary to specify language-support dependent functions that are applicable to a deleted base function. These functions are automatically deleted when the associated base functions are deleted.
- To improve SMP/E performance during installation, very large products should consider providing users with an example of how to package the ++VER DELETE information separately in a dummy function SYSMOD.

This dummy function SYSMOD is received, applied, and accepted to delete the previous releases of your product from the existing target and distribution libraries, and UCLIN is run to delete the SYSMOD entries for the deleted function and for the dummy function. The new release of the product is then installed.

For example, assume the previous release of your product is MYFUNC1, and you want to explain to users how to delete it with 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, along with the instructions you should provide to your users:

```
++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 */.
          DELETEFMID     /* for the deleted functions.*/.
          (DELFUNC MYFUNC1) /* Delete the FMIDs from the */.
                          /* GLOBALZONE entry.        */.

```

_____ End of Packaging Recommendations _____

_____ Packaging Recommendations _____

Section 7.2.8

If use of the VERSION operand between two products is unavoidable, it is the responsibility of the development owner of Product B to ensure that the development owner of Product A understands and agrees to what has been done.

VERSION can also be specified on an element statement to establish the functional level of elements and override the VERSION values specified on the ++VER statement. However, the VERSION operand on the element statement is not additive; it does not automatically take over ownership from the functions specified on the ++VER VERSION operand. To take over ownership from any of the functions specified on the ++VER VERSION operand, you must repeat those values on the VERSION operand for the element statement.

_____ End of Packaging Recommendations _____

_____ Packaging Recommendations _____

Section 7.3.1

- The ++IF MCS should include a comment to identify the product required by the FMID operand.

_____ End of Packaging Recommendations _____

_____ Packaging Recommendations _____

Section 7.3.2

Provide ++IF REQs for all functionally required service, with comments explaining the reason for the REQ.

_____ End of Packaging Recommendations _____

_____ Packaging Recommendations _____

Section 7.5

If you package an element with a ++SRC statement, you should also include the associated ++MOD statement.

_____ End of Packaging Recommendations _____

_____ Packaging Recommendations _____

Section 7.5

Do not use ++MAC, ++MOD, or ++SRC statements to package elements that are not macros, modules, or source, respectively. Use data element statements or hierarchical file system (HFS) element statements (as appropriate) to package such elements.

_____ End of Packaging Recommendations _____

_____ Packaging Recommendations _____

Section 8.1

- If an element needs to be moved, a ++MOVE statement must be used instead of UCLIN.
- A base function should not contain a ++MOVE statement, unless a PTF containing the statement was integrated into a service update of that function.
- New releases of a base function do not own elements that would need to be moved from one library to another. However, there may be shared load

modules that should be moved. In this instance, a base function may contain a ++MOVE for the shared load module.

_____ End of Packaging Recommendations _____

_____ Packaging Recommendations _____

Section 8.2

- A base function should not contain a ++RENAME statement, unless a PTF containing the statement was integrated into a service update of that function.

New releases of a base function do not own elements that would need to be renamed. However, there may be shared load modules that should be renamed. In this instance, a base function may contain a ++RENAME for the shared load module.

- If you want to rename a load module and use inline JCLIN to create a new load module with the original name of the renamed load module, you must package your changes in two SYSMODs: one to rename the existing load module, and one to create the new load module.

The two SYSMODs must not state any relationship to each other and must be applied separately: first the SYSMOD that renames the existing load module, then the one that creates the new load module.

If the SYSMODs need to be restored, they must also be restored separately, in the reverse order of the installation: first the SYSMOD that created the new load module, then the one that renamed the existing load module.

_____ End of Packaging Recommendations _____

_____ Packaging Recommendations _____

Section 8.3

A base function should not contain a ++DELETE statement.

Although a program object 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 without deleting the program object. Instead, you need to resupply JCLIN to define the program object 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.

_____ End of Packaging Recommendations _____

_____ Packaging Recommendations _____

Section 8.5

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. If this is done, SMP/E can

change the affected load module at the CSECT level when a function or module is being deleted.

End of Packaging Recommendations

Packaging Recommendations

Section 8.6

++element VERSION should be used only by different functions of the same product. If the VERSION operand is used by a function that is not part of the same product as the element it wants to assume ownership of, unpredictable results may occur. For example, if Product A owns an element and Product B uses VERSION to assume ownership of that element, it may not be clear which product should ship a given PTF for that element.

If use of the VERSION operand between two products is unavoidable, it is the responsibility of the development owner of Product B to ensure that the development owner of Product A understands and agrees to what has been done.

End of Packaging Recommendations

Packaging Recommendations

Section 9.3

- Use the simplest possible JCL statements.
- Specify all information in uppercase (verbs as well as values).

This is necessary to avoid syntax errors or incorrect results during SMP/E processing.

Note: This convention does not apply to values on the ALIAS statement. These values must be specified in the desired case (uppercase or mixed-case), because they are used as is.

- Do not use update steps in JCLIN data; SMP/E ignores them.
- In the JCLIN of the dependent function, describe only new or changed structure.

The JCLIN for a dependent function should not repeat data already provided in the JCLIN of the base function.

- Do not use abbreviations.
SMP/E may not recognize all abbreviations.
- For copied members (except for ++MOD), use the SYSLIB and DISTLIB operands on the element statements instead of JCLIN to define copies.
- If possible, do not use continued utility control statements. Although SMP/E tries to support all existing formats of the utility control statements, it cannot completely duplicate the syntax checking of the utility. The safe method is to use the simplest format of the utility control statement without continuations.
- Test the JCLIN data as follows:

- Perform RECEIVE, APPLY, and ACCEPT of the product on one system.
- Perform RECEIVE, ACCEPT BYPASS(APPLYCHECK), GENERATE of the product on a second system.
- Compare the SMP/E reports from the two products, checking for discrepancies.
- Compare every library, member by member, between the two products, checking for discrepancies.
- Run the JCLIN data outside of SMP/E and compare the resulting load modules with those built during the SMP/E installs. There should be no differences.

_____ End of Packaging Recommendations _____

_____ Packaging Recommendations _____

Section 9.5

Although JCLIN can be used to identify copied elements, the preferred way of copying elements other than ++MODs is to specify the DISTLIB and SYSLIB operands on the element MCS.

_____ End of Packaging Recommendations _____

_____ Packaging Recommendations _____

Section 9.5.1.3

If you develop a new release of a function that uses totally copied libraries, and the new release copies the distribution library into a different target library from the previous release, you should instruct the users to delete the DLIB entry from the CSI before they apply or accept the new release. This ensures that when SMP/E installs the new release, it builds new DLIB entries pointing to only the new target library.

_____ End of Packaging Recommendations _____

_____ Packaging Recommendations _____

Section 9.6

- Product A should not INCLUDE modules created by Product B unless all of the following are true:
 1. Product B is guaranteed to always be present in the same target zone as Product A
 2. The module always exists in the same library, no matter which release of any product is present
 3. The library containing the module is always guaranteed to exist

If any of these conditions are not true, the product should use CALLIBS instead of explicit INCLUDEs.

- The LMOD RC parameter should be specified on every JCLIN NAME statement. The value for each load module should match the expected return code from link-editing that load module, and the highest value within the JCLIN for an FMID should match the expected APPLY return code documented in the installation instructions for the product.
- If a product's JCLIN specifies INCLUDE AABCMODS(element), the product should REQ the product that installed the element into the library.

_____ End of Packaging Recommendations _____

_____ Packaging Recommendations _____

Section 9.6.2

EXPAND statements should not be used in JCLIN data, because they would be saved in the LMOD entry and would cause the load module to be expanded each time it is link-edited. This is not always desirable.

_____ End of Packaging Recommendations _____

_____ Packaging Recommendations _____

Section 9.6.2

IDENTIFY statements should not be used in JCLIN data. They are produced as part of servicing a module. If found in the JCLIN, they are stored in the LMOD entry and can result in incorrect data being stored during the application of service.

_____ End of Packaging Recommendations _____

_____ Packaging Recommendations _____

Section 10.3

The recommended approach is to use a unique, 3-character component code as the first three characters of the element names, as mentioned above.

_____ End of Packaging Recommendations _____

_____ Packaging Recommendations _____

Section 10.4

The variable portion of the library name should be used to describe the library. For example, the type of elements found in the library could be indicated by MOD, MAC, or PNL, or the national language of the library could be indicated by identi-

fiers such as ENU, FRA, or ESP. Table 18 on page 129 lists the national language identifiers.

_____ End of Packaging Recommendations _____

_____ Packaging Recommendations _____

Chapter 11

Each language-support dependent function should have its own unique FMID.

Each supported language should be individually orderable. Each package should ship everything needed to install the function and the language, including all required functions and installation publications.

Languages can be packaged in a number of ways, including:

- Each language has a separate FMID
- One language is included in the base function and the rest have separate FMIDs
- All languages are packaged in the base FMID

The decision should be based on such factors as:

- If the language functions are large, separate FMIDs permit customers to save space by only installing the languages they wish to use
- If most customers will want most or all of the languages, using a single FMID makes installation easier without wasting space
- How many tapes will be required to ship the various combinations of functions?

_____ End of Packaging Recommendations _____

_____ Packaging Recommendations _____

Section 12.4

- Products should not provide jobs, execs, scripts, or instructions to create files or directories under /var, /tmp, or /dev. If a product needs one of these directories for execution, it should be created dynamically by the product during execution.
- The permission bits for HFS files should be User=7, Group=5, Other=5 for executables, and User=6, Group=4, Other=4 for all other files. (NOTE: there may be some exceptions for daemons, started tasks, and other setuid 0 programs.)
- Products should not require a product-specific HFS. Instead, document the amount of space needed for the product, and allow the installer decide whether or not to install in the root HFS.

_____ End of Packaging Recommendations _____

Appendix B. MVS Service Packaging Rules

This appendix specifies the rules and requirements for building all software fixes that will be installed by SMP/E on MVS systems. It contains the following major sections:

- B.1, "Introduction" introduces service packaging and the importance of standard service packaging rules. This section also defines common terms that relate to service and MVS service packaging.
- B.2, "MVS Service Packaging Rules" on page 212 defines the specifics of the MVS service packaging rules. This section provides the reader with the detail necessary to construct and package a PTF.
- B.3, "IBM Service Delivery" on page 234 describes the existing IBM processes for MVS Service Delivery. These processes are those used by IBM product owners and are available to all Independent Software Vendors (ISVs).
- B.4, "Naming Conventions for Service" on page 236 describes the naming conventions for APAR or APAR fix and PTF SYSMOD IDs.

B.1 Introduction

Most product owners provide problem resolutions when errors are discovered in their software. These problem resolutions generally fall into three categories:

- Immediate, informal fixes (ZAPs)
- Formal, temporary fixes (APAR fixes)
- Formally tested and packaged permanent fixes (PTFs)

Creating the fixes requires the necessary software updates or replacements to be built and tested. The way these elements are combined with installation instructions is called *service packaging*. For MVS environments, the creation and packaging of a permanent fix produces a PTF SYSMOD. Service packaging for MVS is critical to successful PTF SYSMOD installation using SMP/E.

Adhering to the SMP/E syntax rules alone meets the basic requirement for service packaging; however, each PTF SYSMOD can look different and require unique installation activities. Because of these differences, product users continually demand that service packaging be taken one step farther. Product users demand that all MVS service be constructed according to one common format so that unique installation methods are not required. When all MVS PTFs can be installed similarly using SMP/E, software product maintenance becomes easier for product users.

Use of standard MVS service packaging rules satisfies the product users' demand and also benefits product owners. With these rules, product owners save time and effort every time they create an MVS PTF.

The purpose of this appendix is to define and identify standard service packaging rules for MVS. All MVS product owners should use it to package their PTFs regardless of the service delivery methods that will be used.

B.1.1 Service Terminology

The purpose of this section is to define common terms that are used throughout this appendix.

Product owner

The originator of the software product and the supplier of service. Product owners should adhere to MVS service and product packaging rules.

Product user

The customer licensed to use the software product. Product users install and maintain the software product via FMIDs packaged in relative-file format and PTFs packaged in either relative-file or single-file format.

ZAP

An informal correction to a software error. Product owners provide ZAPs to users immediately to correct or bypass a severe problem. ZAPs usually consist of a series of replacement program instructions that the product user must install. ZAPs are not always SMP/E-packaged.

APAR

Authorized Program Analysis Report (APAR). The documented record of a software error. APARs contain problem description information including problem symptoms and elements affected.

APAR identifiers must have seven alphanumeric characters. Consult B.4, "Naming Conventions for Service" on page 236 for an explanation of APAR naming conventions and to avoid conflicts with values used by IBM.

APAR fix

An SMP/E-packaged, temporary correction for a software error. APAR fixes are often designed for use by a specific product user. APAR fixes are eventually replaced and superseded by permanent fixes that are made available to all product users. APAR fixes are identified to SMP/E by the ++APAR statement.

APAR fix SYSMOD IDs use the APAR identifier; however, an APAR fix prefix is substituted for the actual APAR prefix (the APAR fix prefix defaults to "A"). This substitution permits the product owner to deliver different versions of APAR fixes with unique APAR-related names. Product owners must supersede all APAR fixes when packaging the permanent fix.

PTF

Program Temporary Fix (PTF). Considered the official, permanent correction to a software error. PTFs are built and packaged according to service packaging rules. PTFs are made available to all product users who consider them permanent fixes for existing product releases. These fixes are eventually incorporated into the next product release.

PTFs are identified to SMP/E by the ++PTF statement. PTF SYSMOD IDs must have seven alphanumeric characters. Consult B.4, "Naming Conventions for Service" on page 236 for an explanation of PTF naming conventions and to avoid conflicts with values used by IBM.

COMPID

Component ID (COMPID). A nine-character alphanumeric product identifier. It is used within APAR problem descriptions and PTFs to identify the product affected by the problem and changes.

	<p>COMPID is not the same as the unique three-character component code or prefix for element and load module names described in 10.1, “Component Codes” on page 121.</p>
Element	<p>An SMP/E-packaged module, macro, panel, book, message, and so on, shipped as part of a product. Elements are maintained by updates or replacements shipped via PTFs.</p> <p>See 10.3, “Element, Alias, and Load Module Names” on page 122 for an explanation of element naming conventions and to avoid conflicts with values used by IBM.</p>
FMID	<p>Function Modification Identifier (FMID). Identifies the product that owns the elements. You must specify one FMID on each SMP/E ++VER statement.</p> <p>FMIIDs are represented by seven alphanumeric characters. See 10.2, “SYSMOD IDs for Functions” on page 121 for an explanation of FMID naming conventions and to avoid conflicts with values used by IBM.</p>
Cover letter	<p>A comment within a PTF containing its associated informal documentation. One cover letter is required within each MVS PTF. Product owners are encouraged to specify more than the minimum required information to assist the product users with planning and PTF installation.</p> <p>PTF cover letters can be identified by their placement before the first ++HOLD, ++MOVE/RENAME/DELETE, or ++element statement.</p>
Single Fix Strategy	<p>The basis of the MVS service packaging rules. It calls for product owners to fix one or as few software errors as possible in a single PTF packaged in single-file, inline format.</p>
Corrective Service	<p>Small packages of PTFs that are distributed to product users at their request. These packages contain a requested PTF plus any requisites and are intended to provide immediate resolution to users experiencing a specific software problem.</p>
Preventive Service or Expanded Service Options (ESO)	<p>An accumulation of PTFs that have been incorporated into a single package for ease of installation by product users to avoid software problems. These packages are customized to different degrees according to the product user's requirements.</p>
Standard Service Delivery	<p>The content of the delivered package regardless of corrective or preventive delivery method. MVS service packages delivered must contain the following information organized into separate files:</p> <ul style="list-style-type: none">• PTFs• Package handling instructions, helpful hints, and so on• Softcopy packaging list• ERROR ++HOLD statements for PTFs-in-error• UCLIN, if applicable

Standardization of file numbering and placement is not required.

B.2 MVS Service Packaging Rules

It is assumed that when constructing and packaging their MVS software products, product owners understand and adhere to the rules, standards, and naming conventions set forth in this manual and in the following books:

- *SMP/E User's Guide*
- *SMP/E Reference*
- *OS/390 SMP/E Commands* (for OS/390)

This ensures that the appropriate relationships between products and functions have been defined. These relationships specified via the ++FUNCTION, ++VER, ++IF, and ++element statements must be consistently represented and carried forward within the PTFs for each product and function.

This section provides the rules to construct and package standard MVS PTFs that will be installable by SMP/E. Specific IBM service restrictions have also been identified in this section. Planned process changes will eliminate these restrictions over time, but for now, product owners who intend to have their service analyzed and validated by IBM Software Delivery MVS PTF Control before distribution must pay attention to these restrictions.

B.2.1 PTF Size, Format, and Content

Each product owner must balance the number of problems fixed and the number of elements packaged in a single PTF with the effectiveness and impact of that PTF for the product user. Although no rule has been established to limit the number of problems (APARs) that can be fixed by a single PTF, the more elements packaged in one PTF, the more testing required by the product user to apply the service.

As a guideline, a single PTF SYSMOD should not contain more than 70% of the elements for the function to which it applies. When required to update or replace more than 70% of the elements, the product owner should refresh the function (integrate all available, existing service with the product) for redistribution to all users. Product owners should also attempt to make it impossible to obtain an entire product via PTF collections or combinations.

Packaging Rules

S151. Single PTF SYSMODs should not exceed 30 megabytes (Mb) in size.

IBM Service Restriction

Electronic delivery of Corrective Service packages is currently limited to 2 Mb (compacted).

S152. PTFs must use the SMP/E inline packaging technique whenever possible. This is a single file containing all elements of the PTF. Indirect library and relative file packaging techniques should be avoided.

IBM Service Restriction

Electronic delivery of MVS Corrective Service packages is currently limited to single PTF files; therefore, the FILES operand is not supported on any SMP/E modification control statement (MCS) within a PTF.

S153. File attributes for inline PTF SYSMODs must be fixed-block, 80-byte records (RECFM=FB, LRECL=80). Single PTF files should not exceed a block size (BLKSIZE) of 12960.

Note: SMP/E provides the GIMDTS service routine to assist inline packaging of elements with other than fixed-block/80-byte data formats.

S154. Standard MVS PTFs must contain three distinct types of data:

- Instructions to SMP/E that identify what elements are in the PTF SYSMOD and how to install them. These are modification control statements.
- Element update or replacement data.
- Documentation to the PTF user detailing its contents and special handling instructions. This documentation must be located within the PTF (inline) and is to be called the cover letter.

Note: Additionally, standard MVS PTFs can contain JCLIN.

S155. The SMP/E modification control statements within a PTF must use correct syntax. SMP/E syntax rules are defined in the *SMP/E Reference* manual.

Appropriate MCS operands and parameters along with standard cover letter construction are the basis of the following packaging rules.

B.2.2 Standard PTF Structure

The order and format of the standard MVS PTF is presented in the following example. Pay attention to the inclusion of the modification control statements, cover letter, and update and replacement elements. Required format and placement of the modification control statements, cover letter, and update and replacement elements are presented where applicable.

Note: Along with the cover letter, required modification control statements and operands are **highlighted**.

Example

```
++ PTF(tsnnnn5) /* 5752-SC123-tvvvrrr */.  
++ VER(srel)  
   FMID(tvvvrrr)  
   PRE(tsnnnn4)  
   REQ(tsnnnn2)  
  
   SUP(asnnnnn2tsnnnn3)  
   VERSION(tvvvrrr)  
++ IF FMID(tvvvrrr) REQ(tsnnnnn)  
/*  
  (See B.2.3, "PTF Cover Letter" on page 222 for required information.)  
*/.  
++ HOLD...  
++ MOVE...  
++ RENAME...  
++ DELETE...  
++ JCLIN...  
++ element...  
++ element...  
++ element...  
++ element...
```

B.2.2.1 ++PTF Packaging Rules

S156. An MVS PTF SYSMOD must begin with the SMP/E ++PTF statement. The ++PTF statement must include the PTF number, or PTF SYSMOD ID, and a comment containing the applicable nine-character alphanumeric component ID or COMPID (product identifier).

IBM Service Restriction

The ++PTF statement must also include the function modification identifier (FMID). The COMPID/FMID specified within the comment on the ++PTF statement must be identical to the first, if more than one, COMPID/FMID specified in the COMPONENT section of the PTF cover letter, and the FMID must match the FMID coded on the first ++VER statement of the PTF. The hyphens within and between the component and function modification identifiers, as illustrated in the example, are required.

S157. The PTF SYSMOD ID, *tsnnnn5* in the example, must have seven alphanumeric characters (A–Z and 0–9).

- Consult B.4, "Naming Conventions for Service" on page 236 for values used by IBM to avoid conflicts with the PTF prefix values (that is, *ts*).

Note: "PX" has been set aside as the Business Partner PTF prefix value for vendor service that will be recorded in RETAIN.

- It is recommended that the five-digit PTF numbers 90000–90999 be used only for those PTFs that are considered to ship Small Programming Enhancements (SPEs).

S158. Do not use the REWORK operand on the ++PTF statement. When a PTF is discovered to be in error, build a new PTF that supersedes the erroneous PTF.

IBM Service Restriction

The REWORK operand is not supported on the ++PTF statement. It is reserved for use by IBM Software Delivery processes.

S159. You must code the ++PTF statement and required comment on the same line.

S160. The COMPID, 5752-SC123 in the example, must be nine alphanumeric characters. The COMPID used should identify the orderable product to which the PTF applies.

Note: PTFs created by product owners using RETAIN must list the COMPID value that is registered in RETAIN.

B.2.2.2 ++VER

The ++VER statement and its operands describe the environment required for SMP/E to install the PTF SYSMOD.

Packaging Rules

S161. You must code at least one ++VER statement in each MVS PTF.

S162. The ++VER statement must immediately follow the ++PTF statement.

S163. Each PTF ++VER statement must specify only one system release (SREL). See 7.2.2, “Identifying the SREL” on page 57 for valid SREL values.

S164. The DELETE operand must not be specified on a PTF ++VER statement.

S165. The NPRES operand must not be specified on a PTF ++VER statement.

S166. Each PTF ++VER statement must specify a single FMID.

S167. FMIDs must have seven alphanumeric characters. See 10.2, “SYSMOD IDs for Functions” on page 121 for naming conventions and values used by IBM.

S168. ++VER statements within the same PTF must not specify functions that can coexist.

S169. A single PTF can be used to fix common code between multiple FMIDs of a product only if the FMIDs cannot coexist within the same SREL and neither ++JCLIN nor UCLIN is required.

S170. The VERSION operand list of a PTF ++VER statement must specify only FMIDs that are within the same product domain and can coexist with the FMID coded on the PTF ++VER statement.

S171. When updating a current function by adding elements that exist in functionally lower functions, you must use the VERSION operand on either the ++VER or the individual ++element statements associated with the elements being added.

The VERSION operand list must specify all functionally lower FMIDs that had this element.

S172. Only SMP/E product PTFs can reference an SMP/E PTF. Any dependency on SMP/E service levels must not be specified via SMP/E ++VER statements. You must specify these dependencies via SYSTEM ++HOLD statements with REASON(DEPENDENCY).

S173. Within a single ++VER statement, all SYSMODs specified in the PRE, REQ, or VERSION operand lists must be applicable to the same SREL specified on that ++VER statement.

S174. PTF ++VER statements can specify only PTF SYSMODs as PREs and REQs.

S175. All ++VER SREL/FMID combinations and all SYSMODs referenced by PRE, REQ, or VERSION operands must be applicable to a previously announced function, product, and SREL.

S176. When a PTF PREs another PTF that has requisites (PRE, REQ, or IF), those requisites should not be specified for the current, PTF that specifies them as prerequisites.

S177. Element intersections must not exist between a PTF and its corequisite (REQ) PTFs when both specify the same SREL/FMID.

S178. Defining corequisites (++VER REQ) between PTFs must occur in both directions either explicitly (by REQ or ++IF) or by superseding another PTF that satisfies the reference.

S179. Any PTF that is part of a set fixing the same APAR should PRE, REQ, or ++IF any PTFs for the same SREL that contain an element that is not common to the current PTF. (This rule does not apply to PTF sets that contain only language-sensitive elements.)

S180. The SUP operand is always required on the PTF ++VER statements. The SUP operand must list at least one (and all, when multiple exist) APAR SYSMOD being fixed for the first time by that PTF and all APARs fixed by PTFs being superseded.

Note: When superseding the APAR SYSMOD, you must specify the APAR fix prefix rather than the true APAR prefix (that is, use AL, AR, AW, AX or AY; not, PL, IR, OW, BX, or OY). When multiple versions of a corrective APAR SYSMOD have been distributed, you should supersede each version of the APAR fix prefix.

Consult B.4, "Naming Conventions for Service" on page 236 for valid APAR fix prefix values.

S181. Superseding PTFs must reference all of the PREs, REQs, ++IFs, and SUPs of the superseded PTF. These references can be explicit or via PTFs that are specified as a prerequisite (PRE) or corequisite (REQ) by the current, superseding, PTF.

S182. PTF ++VER statements can only SUP PTF SYSMODs that are applicable to the same SREL/FMID combination.

S183. When one PTF SUPs another, the following items from the PTF being superseded must be wholly contained (with the same data, format, and same operand lists) in the current PTF or in the combination of the current PTF and its corequisite (REQ) PTFs (of same SREL/FMID):

- VERSION operands
- ++HOLD statements
- ++MOVE statements
- ++RENAME statements
- ++DELETE statements
- ++JCLIN statements
- Update elements
- Replacement elements
- UCLIN data

B.2.2.3 ++IF Packaging Rules

S184. ++IF statements should follow the ++VER statement to which they apply.

S185. The FMID specified in any PTF ++IF statement must be different from the FMID specified on the ++VER statement to which it is associated.

S186. The REQ operand on the ++IF statement must only specify PTF SYSMODs.

S187. PTF ++IF statements can describe dependencies between two products, even if those products have different SRELS. If the products have different SRELS, the PTF must contain a SYSTEM ++HOLD MCS specifying REASON(DEP).

S188. PTFs must not specify ++IFs to functions that unconditionally coexist with the FMID specified on the ++VER statement to which it applies. Specify these references via PRE or REQ.

S189. Any PTFs that are part of a set fixing the same APAR and that apply to the same SREL should contain a ++IF statement to the PTFs at the functionally higher level.

S190. Any PTFs that are part of a set fixing the same APAR and that apply to the same SREL should contain a ++IF statement to all of the PTFs at a functionally higher level when those PTFs contain at least one common element as the current PTF. (This rule does not apply to PTF sets that contain only language-sensitive elements.)

S191. Any PTF that is part of a set fixing the same APAR should PRE, REQ, or ++IF any PTFs for the same SREL that contain an element that is not common to the current PTF. (This rule does not apply to PTF sets that contain only language-sensitive elements.)

B.2.2.4 ++HOLD

Example

```
++HOLD (ptf_sysmodid) SYSTEM FMID (fmid) REASON (reason_id)
      DATE (yyddd) COMMENT (describe special processing or required
      installation activities here).
```

Packaging Rules

S192. SYSTEM is the only HOLD type allowed within a PTF.

S193. The ptf_sysmodid must match the PTF SYSMOD value specified on the ++PTF statement.

Note: If the PTF is to be installed by an OS/390 system at Release 3 or later, then the ptf_sysmodid must match one of the following:

- the PTF SYSMOD value specified on the ++PTF statement.
- or –
- a PTF SYSMOD value superseded on all ++VER statements.

S194. Consult the *SMP/E Reference* manual for currently supported, valid SYSTEM REASONS (reason_ids).

S195. When used, there must be a ++HOLD statement for each different FMID specified on the ++VER statements that is affected by the reason for the HOLD.

S196. Only one ++HOLD statement for a reason_id per FMID is allowed within a PTF.

S197. The DATE operand is required, and the value specified for yyddd should reflect the date on which the ++HOLD statement was generated.

S198. The COMMENT field is required and must contain the description of the reason for the HOLD status and any special activities that must be performed.

S199. Whenever a ++HOLD statement is used within a PTF, it is recommended that the SPECIAL CONDITIONS section of the PTF cover letter also contain the COMMENT information. Refer to B.2.3, “PTF Cover Letter” on page 222 for more details.

IBM Service Restriction

Whenever a ++HOLD statement is used within a PTF, the SPECIAL CONDITIONS section of the PTF cover letter must contain the COMMENT information. Refer to B.2.3, “PTF Cover Letter” on page 222 for more details.

S200. You must place ++HOLD statements after all ++VER and ++IF statements but immediately before any ++MOVE, ++RENAME, ++DELETE, or ++element statements.

S201. You must not use the CLASS operand on PTF ++HOLD statements. It is reserved for use by IBM.

B.2.2.5 ++MOVE, ++RENAME, ++DELETE

Because SMP/E will not restore a PTF containing a ++DELETE statement, it is recommended that ++DELETE be used in smaller PTFs to reduce the risk of users needing to restore these PTFs.

Packaging Rules

S202. A PTF can only ++MOVE, ++RENAME, or ++DELETE an LMOD that is within the domain of the functions specified in the PTF ++VER statements.

S203. A PTF can contain only one ++MOVE statement for a given LMOD “TO” the same data set.

S204. A maximum of two ++MOVE statements are allowed for an LMOD within PTFs, one ++MOVE for each library.

S205. A PTF that supersedes another PTF containing a ++MOVE statement must identically carry that statement forward, unless the current PTF is deleting the LMOD via ++DELETE.

S206. If a PTF must ++MOVE the element or LMOD to a different data set, that PTF must PRE any previous PTFs that renamed the LMOD or contained a ++MOVE for the same element.

S207. A PTF can contain no more than one ++RENAME or ++DELETE statement for a given LMOD.

S208. For each PTF containing a ++MOVE statement for a MOD, MAC, or SRC, its ++VER statement must:

- PRE or SUP the last previous PTF for the same SREL/FMID combination (if one exists) that moved that element
- PRE or SUP the last previous PTF for the same SREL/FMID combination that replaced that element
- PRE or SUP all PTFs for the same SREL/FMID combination that updated the element since it was last replaced or moved
- PRE the last previous PTF that renamed the element

S209. PTFs containing a ++RENAME statement must PRE, not SUP, the last previous PTF (if one exists) that contained a ++RENAME or ++MOVE statement for the same element.

S210. When superseding a PTF that used UCLIN to perform a MOVE, RENAME, or DELETE, the current PTF must include a ++MOVE, ++RENAME, or ++DELETE statement to perform the same action.

S211. ++MOVE, ++RENAME, and ++DELETE must follow all ++VER, ++IF, and ++HOLD statements.

S212. All modification control statements following ++MOVE, ++RENAME, or ++DELETE statements must properly reference the new element information designated by these statements.

S213. When superseding a PTF that contained a ++DELETE, the current PTF must identically carry the ++DELETE forward.

S214. When a PTF contains a ++DELETE statement, it must PRE or SUP any previous PTFs that contained ++MOVE or ++RENAME for the same LMOD.

S215. PTFs that contain ++DELETE statements must contain a SYSTEM ++HOLD with REASON(DELETE).

B.2.2.6 ++JCLIN Packaging Rules

S216. You can include only one ++JCLIN statement in a PTF.

S217. You must carry forward ++JCLIN and associated data from superseded PTFs.

S218. You must place the ++JCLIN statement after any and all ++VER, ++IF, ++MOVE, ++RENAME, or ++DELETE statements but before the first ++element statement.

B.2.2.7 ++element Packaging Rules

S219. See 10.3, “Element, Alias, and Load Module Names” on page 122 for information and restrictions regarding element naming conventions.

S220. Each element packaged within an MVS PTF must be in either update or replacement format. You must provide data immediately following the ++element statement unless you are using the DELETE operand.

S221. A single PTF can contain both update and replacement forms of elements; however, a single PTF cannot contain the same type update and replacement (that is, MACUPD and MAC) for the same element.

S222. When adding an element to an FMID, the PTF must contain the replacement form of the element.

S223. When specifying an update PTF as a prerequisite, the current PTF must not include the previous updates.

S224. When superseding an update PTF, the current PTF must include all previous updates.

S225. For each update element within a PTF, each ++VER statement must:

- PRE the last previous PTF for the same SREL/FMID combination (if one exists) that replaced the element
- PRE or SUP all previous PTFs for the same SREL/FMID combination that updated the element since it was last moved or replaced
- PRE or SUP the last previous PTF for the same SREL/FMID combination that moved the element

Notes:

1. If the last previous PTF moved the element and contained a replacement for the element, then you must specify that PTF as a prerequisite and not supersede it.
2. An update PTF that has been superseded by another update PTF is mentioned as a PRE for the current PTF can be omitted from the PRE list.

S226. For each replacement element within a PTF, each ++VER statement must:

- PRE or SUP the last previous PTF for the same SREL/FMID combination (if one exists) that replaced the element
- PRE or SUP all PTFs for the same SREL/FMID combination that updated the element since it was last moved or replaced
- PRE or SUP the last previous PTF for the same SREL/FMID combination that moved the element

S227. When updating a function by adding elements that exist in functionally lower functions, you must use the VERSION operand on either the ++VER or the individual ++element statements associated with the elements being added. The VERSION operand list should specify all functionally lower FMIDs that had this element.

S228. The PTF must identify any element to be deleted by using the replacement format of the ++element statement.

S229. When deleting an element that has multiple forms, you must individually delete each form.

S230. Do not provide data for an element being deleted. Only the ++element statement with the DELETE operand is required.

S231. When one PTF supersedes another PTF that contained a ++element DELETE, the current PTF must carry the ++element DELETE forward.

S232. ++element statements for the same part with different element types must specify different libraries.

S233. The RMID operand must not be used on any ++element statement.

S234. Data types for existing elements can be changed only by deleting the existing element and adding a new element.

IBM Service Restriction

Currently, to delete an existing element and add a new element with the same name, you must use two PTFs whenever the new element is being added to the same DISTLIB as the original element.

S235. PTFs must not require a newer version or release of SMP/E than the product itself did.

B.2.2.8 UCLIN Packaging Rules

S236. Avoid using UCLIN data within a PTF. (For assistance with UCLIN avoidance, contact the owner of this manual.)

S237. If UCLIN is required, you must deliver it as PTF documentation within the SPECIAL CONDITIONS section of the cover letter.

S238. Each PTF containing UCLIN must include a SYSTEM ++HOLD with REASON(ACTION) to alert the product user of the UCLIN steps.

S239. "SET BDY" must be the first statement of the UCLIN data.

S240. "ENDUCL" must be the last statement of the UCLIN data.

B.2.2.9 Other

S241. The ++FUNCTION statement is not allowed within PTFs.

IBM Service Restriction

The ++ASSIGN statement must not be used within PTFs. It is reserved for use by IBM Software Delivery Solutions processes.

IBM Service Restriction

A PTF is not allowed to change the attributes of a distribution library. The build process for CBIPO orders uses the library information from the original product, and does not recognize any subsequent changes.

B.2.3 PTF Cover Letter

The cover letter is that portion of the PTF containing informal documentation. The cover letter is constructed as a large comment. PTF summary information along with special handling instructions are provided within this comment.

Many IBM service restrictions exist today regarding the MVS PTF cover letter content, format, and placement. Remember that these restrictions apply only to those PTFs that will be analyzed and validated by IBM Software Delivery Solutions MVS PTF Control before distribution.

All product owners must adhere to the minimum data requirements for the standard MVS PTF Cover Letter identified by the **Packaging Rules**. It is suggested that product owners consult the expanded descriptions contained within the IBM Software Delivery Solutions Restrictions to obtain information and guidelines about additional cover letter content.

Packaging Rules

S242. Product owners must include one, and only one, cover letter in each MVS PTF.

Note: There are no restrictions limiting the size of PTF cover letters. When constructing cover letters, product owners should consider the guideline docu-

mented earlier concerning single PTF SYSMOD size. Product owners using RETAIN should also remember that RETAIN allows a maximum of 590 64-byte records of problem summary information to be entered on the Responder Page of the APAR record (use of the PIN command can reduce this maximum).

The standard MVS PTF Cover Letter is presented in the example that follows.

Note: All required headings are **highlighted**. Keywords within the SPECIAL CONDITIONS section are entered dependent on the PTF content.

Example

```

/*
PROBLEM DESCRIPTION(S):

COMPONENT: or PRODUCT ID:

APARS FIXED:

SPECIAL CONDITIONS:

COPYRIGHT:
INTEGRITY:
ACTION:
AO:
DELETE:
DEPENDENCY:
DOCUMENTATION:
EC:
ENHANCEMENT:
EXRF:
FULLGEN:
IOGEN:
MSGSKEL:
MVSCP:
UCLIN:
NONE:

COMMENTS:
*/.

```

Packaging Rules

The cover letter is handled as one single comment by SMP/E. This is accomplished by use of the SMP/E start-of-comment delimiter (/*) and the end-of-comment delimiter (*).

Specific columns will be outlined for the placement of all headings and keywords within the cover letter. Except for the cover letter delimiters (/*) and (*), the specific columns are intended for appearance purposes only. No PTF is considered to have violated the MVS service packaging rules for violation of these cosmetic column requirements unless the violation causes the PTF to be either unacceptable to or incorrectly installed by SMP/E.

S243. The cover letter must follow the SMP/E ++PTF statement and precede the first ++HOLD, ++MOVE/RENAME/DELETE, or ++element statement. The cover letter becomes part of an SMP/E statement.

IBM Service Restriction

The cover letter must immediately follow the last SMP/E ++VER or ++IF statement and immediately precede the first ++HOLD, ++MOVE/RENAME/DELETE, or ++element statement. Therefore, the cover letter becomes part of the last SMP/E ++VER statement (or the last ++IF statement, if one is present) of the PTF.

S244. To start the cover letter, remove the period ending the SMP/E statement preceding the cover letter and place the /* after column one on the next line. The cover letter text must start on the following line.

IBM Service Restriction

To start the cover letter, remove the period ending the last ++VER statement (or the last ++IF, if one exists) and place the /* after column one on the next line. The cover letter text must start on the following line.

S245. To end the cover letter, use the */ followed by a period after column one on a separate line.

S246. /* must not be used within the PTF cover letter text except to end the cover letter. /* and /** can be used within the PTF cover letter text.

S247. You must enter cover letter text into four sections, which are identified by headings. These are **highlighted** in the example. All four cover letter sections are mandatory.

IBM Service Restriction

You must enter cover letter text into five sections, which are identified by headings. APARS FIXED: is the heading identifying the additional section required. All five cover letter sections are mandatory.

S248. Section headings must not be used more than once within the cover letter.

IBM Service Restriction

Cover letter sections and headings must appear in the same order as presented in the example.

S249. Each section's heading should start in column 3, must be capitalized, and must include a colon as illustrated in the example. Enter data for each heading two columns past the colon and must not exceed column 72. You can continue data on the following line after column 6.

S250. SPECIAL CONDITIONS keywords should begin after column 4, must be capitalized, and must include a colon. Data for each SPECIAL CONDITIONS keyword should begin two columns past the colon and must not exceed column 72. Data can be continued on the following line after column 6.

At this point, each heading and SPECIAL CONDITIONS keyword will be identified and discussed individually.

B.2.3.1 PROBLEM DESCRIPTION(S):

Example

```
PROBLEM DESCRIPTION(S):
  BX11131 - PROGRAM CHECK IN xxxx WHEN EOF RECORD FOUND ON THE LAST
           TRACK OF A CYLINDER.
  BX12347 - INVALID ERROR MESSAGE WHEN I/O ERROR OCCURS DURING EXPORT.
  BX13579 - INTEGRITY PROBLEM
```

Packaging Rules

S251. Specify all new APARs (or software error identifiers) fixed by the current PTF. Additionally, the PTF owner can opt to include APAR numbers and problem descriptions for APARs fixed previously by PTFs that are being referenced (specified as a prerequisite or corequisite, or as superseded) by the current PTF.

IBM Service Restriction

Specify the original APAR prefix; not, the APAR fix prefix (that is, use PL, IR, OW, BX, or OY; not, AL, AR, AW, AX, or AY).

S252. Enter a brief summary of the problem documented by each APAR listed.

S253. If the APAR describes an integrity or security problem, then only the words "INTEGRITY PROBLEM" are to be listed as the problem description text.

B.2.3.2 COMPONENT: or PRODUCT ID:

Example

```
COMPONENT: 5752-SC123-tvvvrr1
           5752-SC124-tvvvrr2
```

Packaging Rules

S254. You must specify the full nine-character alphanumeric component ID or product identifier applicable to the PTF in this section.

S255. For a multi-component or multi-product PTF, you must list all the components or products affected on a separate line.

S256. The first COMPID or product identifier specified in this section must be identical to that specified within the comment on the ++PTF statement.

IBM Service Restrictions

- You must use the COMPONENT: heading, not the PRODUCT ID: heading.
- You must specify the full nine-character alphanumeric component ID and function identifier (separated by hyphens) applicable to the PTF in this section.
- For a multi-component PTF, you must list all the components affected with each COMPID/FMID on a separate line.
- The first COMPID/FMID specified in the COMPONENT: section must be identical to that specified within the comment on the ++PTF statement.

B.2.3.3 APARS FIXED:

This heading and section are optional for product owners who choose not to have their PTFs analyzed and validated by IBM Software Delivery MVS PTF Control before distribution.

Example

```
APARS FIXED: BX04442,BX04497,BX14150,BX17654,BX22000,BX32050,  
            BX35150
```

IBM Service Restrictions

- Specify those APAR numbers that are being fixed for the first time in all of the environments stated in the ++VER statements. Additionally, the PTF owner can opt to enter any APARs fixed by PTFs that are being superseded by the current PTF.
- Use the complete APAR number including the original APAR prefix, not the APAR fix prefix (that is, use PL, IR, OW, BX, or OY; not, AL, AR, AW, AX, or AY).
- Do not list an APAR number more than once, even if the APAR is fixed in multiple environments by the PTF.
- Include a comma or space (or both) between each APAR number specified.

B.2.3.4 SPECIAL CONDITIONS:

Use the SPECIAL CONDITIONS keywords based on the PTF content (individual keyword descriptions follow).

Packaging Rules

S257. When a PTF has special information that cannot be represented in the SMP/E modification control statements or if the PTF requires the use of the ++HOLD facility, you must include descriptive information in this section of the cover letter.

S258. List the keywords that are applicable to the PTF.

S259. Do not use the SPECIAL CONDITIONS: heading or any of its keywords more than once, unless otherwise indicated.

S260. If none of the keywords are applicable, you must list the NONE: keyword.

COPYRIGHT: Check with your company regarding legal requirements and guidelines for copyright statements.

Note: All IBM licensed code must contain a copyright statement and property legend.

Example

This example shows an original copyright date of 1976 for an IBM domestic licensed product. The XXXX could show the date of a later modification.

```
SPECIAL CONDITIONS:
  COPYRIGHT: 5740-xx2 (C) COPYRIGHT IBM CORP. 1976 XXXX
            LICENSED MATERIAL - PROGRAM PROPERTY OF IBM
```

Packaging Rules

S261. If applicable, enter one, and only one, copyright statement.

Note: Typically, a copyright statement contains the program number, the copyright symbol or "(C)," the word "COPYRIGHT," the owner's name, and the date of copyright. In case of modifications derived from the original code, multiple dates can be shown. However, if only one date is shown, it must be the original date rather than the modification date.

INTEGRITY: Example

```
SPECIAL CONDITIONS:
  INTEGRITY: PX30181,PX30182,PX30192,PX30194,
            PX30196,PX30197
```

(or)

```
SPECIAL CONDITIONS:
  INTEGRITY:
  INTEGRITY SET: PX30181,PX30182,PX30192,PX30194,
                PX30196,PX30197
```

Packaging Rules

S262. Specify the INTEGRITY: keyword when the PTF corrects a system exposure that allowed unauthorized access to protected resources.

S263. All PTFs within a set fixing the same integrity problem must be listed following the INTEGRITY: keyword. Separate the PTF numbers with commas or blanks.

S264. For a set containing only one PTF, list only that PTF after the INTEGRITY: keyword.

IBM Service Restrictions

- Specify the INTEGRITY: and INTEGRITY SET: keywords when the PTF corrects a system exposure that allowed unauthorized access to protected resources.
- You must use the INTEGRITY: and INTEGRITY SET: keywords together.
- All PTFs within a set that fix the same integrity problem must be listed following the INTEGRITY SET: keyword. Separate the PTF numbers with commas or blanks.
- For a set containing only one PTF, list only that PTF after the INTEGRITY SET: keyword.

ACTION: Example

SPECIAL CONDITIONS:

ACTION: A "CLPA" MUST BE PERFORMED AT IPL TIME FOR THIS PTF TO BECOME ACTIVE.

ACTION: RUN AN ACCEPT, NOAPPLY OF THIS PTF. AFTER IT IS ACCEPTED, RUN A LINK EDIT JOB TO ALLOW THE "RENT" ATTRIBUTE TO BE TURNED ON.

Packaging Rules

S265. Use the ACTION: keyword whenever the PTF requires an action by the product user that is not covered by any of the other SPECIAL CONDITIONS keywords.

S266. Use ACTION: as many times as there are unique actions required. If multiple ACTIONS are specified, you must enter them as new paragraphs.

S267. With this keyword, the PTF must be held from installation until the condition has been taken care of by the user. The PTF must contain an inline SYSTEM ++HOLD statement with REASON(ACTION).

AO: Example

SPECIAL CONDITIONS:

AO: NEW OPERATOR MESSAGE IEA278I

Packaging Rules

S268. Use the AO: keyword whenever the PTF requires action to change automated operations procedures, associated data sets, and user exits in products or in customer applications.

S269. Use AO: as many times as there are unique actions required. If multiple AO: actions are specified, you must enter them as new paragraphs.

S270. With this keyword, the PTF must be held from installation until the condition has been taken care of by the user. The PTF must contain an inline SYSTEM ++HOLD statement with REASON(AO).

DELETE: Example

SPECIAL CONDITIONS:

DELETE: THIS PTF CONTAINS A ++DELETE MCS FOR THE FOLLOWING LMODS:

LMOD1	SYSLIB1
LMOD2	SYSLIB5
LMOD3	SYSLIB1, SYSLIB5

SMP/E WILL NOT RESTORE THIS PTF.

APPLY AND ACCEPT THIS PTF BY INCLUDING A
 BYPASS(HOLDSYS(DELETE)) ON THE APPLY AND ACCEPT CMDS.

Packaging Rules

S271. Specify the DELETE: keyword any time the PTF contains a ++DELETE statement.

S272. With this keyword, the PTF must be held from installation until the condition has been taken care of by the user. The PTF must contain an inline SYSTEM ++HOLD statement with REASON(DELETE).

DEPENDENCY: Example

SPECIAL CONDITIONS:

DEPENDENCY: XXXXX LEVEL 6.27 REQUIRED PRIOR TO INSTALLING
 THIS SYSMOD.

Packaging Rules

S273. Specify the DEPENDENCY: keyword any time the PTF has a software dependency that cannot be specified on a ++VER or ++IF statement.

Note: Only SMP/E product PTFs can reference an SMP/E PTF. You must specify any dependency on SMP/E service levels via the DEPENDENCY: keyword.

S274. Insert a brief description that provides specific instructions to be followed before, during, or after the installation of the PTF to satisfy the dependency.

S275. Use DEPENDENCY: as many times as there are unique dependencies related to this PTF.

S276. With this keyword, the PTF must be held from installation until the condition has been taken care of by the user. The PTF must contain an inline SYSTEM ++HOLD statement with REASON(DEP).

DOCUMENTATION: Example

SPECIAL CONDITIONS:

DOCUMENTATION: NEW MESSAGE IDC116I ADDED.
 MESSAGE (text)
 EXPLANATION (text)
 SYSTEM ACTION (text)
 PROGRAMMER OR OPERATOR RESPONSE (text)

Packaging Rules

S277. Specify the **DOCUMENTATION:** keyword any time the PTF adds a message, code, or command; expands on them; or changes their meaning, text or definition.

S278. Enter important information that should be read or recorded before PTF installation.

S279. Use **DOCUMENTATION:** as many times as there are unique messages or codes added or changed by this PTF.

S280. With this keyword, the PTF must be held from installation until the condition has been taken care of by the user. The PTF must contain an inline **SYSTEM ++HOLD** statement with **REASON(DOC)**.

EC: Example

```
SPECIAL CONDITIONS:  
EC: EC123456 REQUIRED ON xxxx PRIOR TO INSTALLING THIS PTF.
```

Packaging Rules

S281. Specify the **EC:** keyword any time the PTF has a hardware dependency.

S282. Following the keyword, enter all information necessary for the product user to satisfy the dependency.

S283. Use **EC:** as many times as there are unique hardware dependencies related to this PTF.

S284. With this keyword, the PTF must be held from installation until the condition has been taken care of by the user. The PTF must contain an inline **SYSTEM ++HOLD** statement PTF with **REASON(EC)**.

ENHANCEMENT: Example

```
SPECIAL CONDITIONS:  
ENHANCEMENT: PTF IMPLEMENTS DEVICE SUPPORT FOR D/T XXXX
```

Packaging Rules

S285. You must use the **ENHANCEMENT:** keyword for PTFs that add program options or new functions, or those that contain Small Programming Enhancements (SPEs). SPE PTFs are denoted by the five-digit PTF number of 90000-90999.

S286. You must enter text to describe the new or enhanced function.

S287. Use **ENHANCEMENT:** as many times as there are unique enhancements to identify.

EXRF: Example

SPECIAL CONDITIONS:

EXRF: IF THIS IS AN XRF ENVIRONMENT, THIS MAINTENANCE MUST BE APPLIED TO ALL SYSTEMS IN THIS ENVIRONMENT AT THE SAME TIME. USERS NOT USING XRF SHOULD USE THE BYPASS FOR THIS REASON CODE.

Packaging Rules

S288. Specify the EXRF: keyword when the PTF applies to an Extended Recovery Facility Environment (MVS/XA, CICS/MVS, IMS, DFP, VTAM or NCP), and the PTF must be applied to all XRF systems at the same time to maintain system compatibility.

S289. Use EXRF: as many times as there are unique installation instructions required. You must specify multiple entries of EXRF: as new paragraphs.

S290. With this keyword, the PTF must be held from installation until the condition has been taken care of by the user. The PTF must include an inline SYSTEM ++HOLD statement with REASON(EXRF).

FULLGEN: Example

SPECIAL CONDITIONS:

FULLGEN: SYSGEN REQUIRED FOR COMPLETE IMPLEMENTATION OF THE FIX SUPPLIED BY THIS PTF - GENTYPE = ALL

Packaging Rules

S291. Specify the FULLGEN: keyword any time the PTF requires a SYSGEN to be effective.

S292. Enter specific instructions to be followed before, during, or after the application of the PTF.

S293. With this keyword, the PTF must be held from installation until the condition has been taken care of by the user. The PTF must include an inline SYSTEM ++HOLD statement with REASON(FULLGEN).

I0GEN: Example

SPECIAL CONDITIONS:

I0GEN: YOU MUST PERFORM AN I/O GENERATION FOR COMPLETE IMPLEMENTATION OF THE FIX SUPPLIED BY THIS PTF.
GENTYPE = IO

Packaging Rules

S294. Specify the I0GEN: keyword any time the PTF requires an I/O generation or other partial generation to become effective.

S295. Enter specific instructions to be followed before, during, or after the application of the PTF.

S296. With this keyword, the PTF must be held from installation until the condition has been taken care of by the user. The PTF must include an inline SYSTEM ++HOLD statement with REASON(IOGEN).

MSGSKEL: Example

```
SPECIAL CONDITIONS:
MSGSKEL: IF USING THE MVS MESSAGE SERVICE (MMS), USE THE
MVS MESSAGE COMPILER AS A POST INSTALL STEP TO UPDATE THE
RUN TIME LIBRARY(S). FAILURE TO UPDATE THE RUN TIME
LIBRARY(S), MAY RESULT IN DISPLAY OF ENGLISH MESSAGE TEXT ONLY.
```

Packaging Rules

S297. Enter the MSGSKEL: keyword any time the PTF contains message changes that must be compiled in order for translated versions of the message changes to become operational on extended TSO consoles.

S298. Use MSGSKEL: as many times as there are unique actions that are required to make the PTF changes operational.

S299. With this keyword, the PTF must be held from installation until the condition has been taken care of by the user. The PTF must contain an inline SYSTEM ++HOLD statement with REASON(MSGSKEL).

MVSCP: Example

```
SPECIAL CONDITIONS:
MVSCP: YOU MUST EXECUTE THE MVS CONFIGURATION PROGRAM FOR
COMPLETE IMPLEMENTATION OF THE FIX SUPPLIED BY THIS PTF.
```

Packaging Rules

S300. Specify the MVSCP: keyword any time that the designated PTF requires the MVSCP (MVS Configuration Program) to be run to incorporate the configuration changes.

S301. With this keyword, the PTF must be held from installation until the condition has been taken care of by the user. The PTF must contain an inline SYSTEM ++HOLD statement with REASON(MVSCP).

UCLIN: Example

```
SPECIAL CONDITIONS:
UCLIN: THIS UCLIN WILL ADD A NEW MODULE TO THE
TARGETZONE AND HAS TO BE RUN BEFORE APPLYING THE SYSMOD.
SET BDY(TGT1).
UCLIN.
ADD MOD (IGG0CLA0) LMOD (IGG0CLA1)
FMID (JBB1123).
ENDUCL.
```

Packaging Rules

S302. Specify the UCLIN: keyword any time the PTF requires a UCLIN job be run to become effective.

S303. When specified, the product owner must state whether the UCLIN is pertinent to APPLY processing, ACCEPT processing, or both.

S304. Use UCLIN: as many times as there are unique UCLIN steps required by this PTF.

S305. The cover letter must contain the full UCLIN statement package, following the UCLIN: keyword and explanation.

S306. With this keyword, the PTF must be held from installation until the condition has been taken care of by the user. The PTF must contain an inline SYSTEM ++HOLD statement with REASON(ACTION). Use of this ++HOLD also requires that the keyword ACTION: be specified to alert the product user of the UCLIN steps.

NONE: Example

```
SPECIAL CONDITIONS:
  NONE:
```

Packaging Rules

S307. If none of the other SPECIAL CONDITIONS keywords are applicable, you must list the NONE: keyword. You must never use this keyword if any other keyword is used.

B.2.3.5 COMMENTS:

Example

```
COMMENTS: THIS PTF IS BEING DISTRIBUTED AS A LEVEL-SET BECAUSE
           THE PREREQUISITE CHAINS OF xxxx HAVE BECOME EXCEEDINGLY COMPLEX.
           THIS PTF CONTAINS ALL IMPACTED MOD/MAC(S) AT THE CURRENT SERVICE
           LEVEL, LEVEL-SETTING xxxx TO THE CURRENT LEVEL AND SUPERSEDING
           THE FORMER PRE CHAINS.
```

```
THIS PTF FIXED PE APAR OZ12345, WHICH REPORTED AN ERROR WITHIN
PTF UZ00123.
```

(or)

```
COMMENTS: NONE
```

Packaging Rules

S308. Comments are to be entered to pass on additional notes to the product user. However, if no comments are to be passed on, you must specify the word "NONE."

S309. If multiple comments are given, you must enter them as new paragraphs. You should not use the COMMENTS: heading more than once.

B.3 IBM Service Delivery

This section outlines the MVS Service Delivery processes used by IBM product owners. Adherence to the rules and IBM Service Restrictions¹ set forth in this appendix positions product owners to take advantage of all IBM MVS Service Delivery mechanisms.

Contact Information: To obtain additional information regarding IBM Software Delivery Solutions MVS Service Delivery mechanisms and services, Independent Software Vendors should call 1-800-926-0364. IBM product owners should send inquiries to MVSAUTH at MAHVM1.

Through the different IBM Software Delivery Solutions processes, product owners can provide the following service deliverables worldwide:

- Corrective service
- Preventive service or Expanded Service Options (ESO)
- Online program listings
- PTF-in-error (PE) information

IBM product owners primarily interface with PTF Control within Software Delivery for MVS Service Delivery.

MVS PTF Control collects and error-checks all PTFs. This single point of control enforces the standardization of PTF format and construction. PTF Control distributes the PTFs to manufacturing centers worldwide for corrective and preventive service delivery, administers the View Program Listings (VPL) facility, and performs daily PE management for corrective and preventive packages.

B.3.1 Service Process Initialization

The PTF Control processes and database must be prepared for the receipt and handling of PTFs for new products, releases, and modifications.

Product owners must notify PTF Control of new products, releases, and modifications by completing an MVS Service Planning Information Form. With the information provided on this form PTF Control initializes their database to recognize new functions (FMIDs).

For each new FMID, the Service Planning Information Form must contain:

- Nine-character component ID, assigned during Program Information Form (PIF) processing
- Component names
- Three-character release number, assigned during PIF processing
- Applicable system releases (SRELs)
- PTF prefix
- APAR fix prefix

¹ Product owners can choose IBM Software Delivery Solutions for service distribution *without* PTF validation in order to avoid these process restrictions.

- Program listing authorization and control information, if applicable

PTF Control also requires the SMP/E modification control statements (SMPMCS file), or install logic, for each new FMID. PTF Control uses the product definition information during PTF validation.

B.3.2 PTF Submission

Product owners submit PTFs, associated PTF information files, and program listings, if applicable, either electronically or via courier to MVS PTF Control.

IBM Service Restriction

PTF information and files submitted to IBM Software Delivery Solutions MVS PTF Control must be submitted according to the specifications outlined in the Process Documentation References (PDRs).

PTF files that do not satisfy the submission requirements and the rules in this appendix will be returned to the product owner for rework.

B.3.2.1 MVS PTF Control

MVS PTF Control is responsible to:

- Receive PTFs and program listings from product owners worldwide
- Examine and acknowledge receipt of PTF and program listings
- Syntax check, via SMP/E RECEIVE and this appendix, modification control statements and PTF construction
- Validate PTF documentation (cover letter)
- Analyze and validate PTF relationships and installability (++VER, ++IF, ++element information)
- Distribute PTF to worldwide Common Corrective Service System (CCSS) sites for Corrective delivery
- Distribute PTF to worldwide manufacturing centers for preventive delivery
- Activate and administer online PTF program listing availability
- Perform daily PE management for corrective and preventive service packages:
 - Interpret daily RETAIN Automatic Software Alert Process (ASAP) Report
 - Transform PE, PE Delete transactions into ERROR ++HOLD/++RELEASE statements
 - Update RETAIN Preventive Service Planning (PSP) PE buckets and upgrades

All PTFs processed by MVS PTF Control are automatically eligible for distribution via CCSS, PUTs, CUMs, and Service-only CBPDOs.

B.4 Naming Conventions for Service

This section describes the naming conventions to follow when using SMP/E. Specifically, it discusses the naming conventions for APAR (or APAR fix) and PTF SYSMOD IDs.

Consult Chapter 10, “Naming Conventions” on page 121 for naming convention information about element names and function SYSMOD IDs (FMIDs).

The general naming convention rule is that all SYSMOD IDs must have seven alphanumeric or national characters (A–Z and 0–9, or @, #, \$) and must start with a letter. Beyond this, the specific naming conventions depend on the type of SYSMOD: APAR or APAR fix, PTF, or USERMOD.

To avoid conflicts between the names of Independent Software Vendor (ISV) and IBM-written SYSMODs, the ID for an ISV SYSMOD should not start with a letter used by IBM. IBM uses the following letters to start the names of its SYSMODs:

- A–K and V–Z are used for APARs or APAR fixes.
- U is used for PTFs.

The easiest way to avoid a conflict is to start the ID of an ISV SYSMOD with a letter from L to T.

Below is a description of the conventions used by IBM for the remaining six characters of SYSMOD IDs. This information is provided to help you develop a naming scheme for your own SYSMODs and to avoid conflicts with IBM-written SYSMODs.

The IBM convention for the SYSMOD ID of a service SYSMOD (APAR, APAR fix, PTF) or USERMOD is *tsnnnn*, where:

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 ISVs
U	Used by IBM for PTFs
V–Z	Used by IBM for various levels of an APAR fix

s is the system to which the SYSMOD and its associated product apply. These are the values used by IBM for PTFs and APAR fixes:

L, N, P	Licensed products for MVS
R	Unlicensed products for MVS
W, Y, Z	MVS products only

Any valid character can be used for user SYSMODs.

nnnn is an additional identifier for the SYSMOD. For PTFs and APAR fixes supplied by IBM, it is a number from 00001 to 99999.

Appendix C. Mapping of Old Rule Numbers to New Rule Numbers

This appendix lists all the old rule numbers and the equivalent new rule numbers.

Table 19 (Page 1 of 5). Table for Mapping old rule number to new rule numbers. This table maps the rule numbers used in previous versions of the document to the new rule number used in this version.

Old Rule Number	New Rule Number
1	110
2	120
3	300
4	400
5	500
6	600
7	700
8	800
9	900
10	1000
11	1100
12	1200
13	1310
14	1320
15	1350
16	2100
17	2200
18	2300
19	2305
19.1	2330
20	2600
21	2700
21.1	2710
22	2900
23	2950
24	3400
24.1	3410
25	3500
25.1	3510
26	3650
27	3700

Table 19 (Page 2 of 5). Table for Mapping old rule number to new rule numbers. This table maps the rule numbers used in previous versions of the document to the new rule number used in this version.

Old Rule Number	New Rule Number
28	3800
28.1	3810
29	3900
29.1	3910
30	4000
31	4200
32	4300
33	4910
34	5100
35	5200
36	5300
37	5400
38	5500
39	5750
39.1	5810
39.2	5820
39.3	5830
40	5900
41	5910
42	6200
43	6205
44	6210
45	6300
46	6400
47	6500
48	6600
49	6700
50	6800
51	6900
52	7000
53	7200
54	7400
55	7500
56	7600
57	7700
58	7800

Table 19 (Page 3 of 5). Table for Mapping old rule number to new rule numbers. This table maps the rule numbers used in previous versions of the document to the new rule number used in this version.

Old Rule Number	New Rule Number
59	7900
60	8000
61	8100
62	8200
63	8300
64	8500
65	8600
66	8700
67	8800
68	9000
69	9100
70	9200
71	9210
72	9300
73	9400
74	9500
75	9600
76	9700
77	9800
78	9900
79	10000
80	10010
81	10100
82	10110
82.1	10111
83	10112
84	10115
85	10117
86	10119
87	10200
88	10300
89	10400
90	10500
91	10600
92	10700
93	10800

Table 19 (Page 4 of 5). Table for Mapping old rule number to new rule numbers. This table maps the rule numbers used in previous versions of the document to the new rule number used in this version.

Old Rule Number	New Rule Number
94	10900
95	11000
96	11100
97	11200
98	11300
99	11400
100	11500
101	11600
102	11700
103	11800
104	11900
105	12000
106	12100
107	12200
107.1	12210
108	12300
108.1	12310
109	12400
110	12500
111	12510
112	12600
113	12700
114	12800
115	12900
116	13000
117	13100
118	13110
119	13200
120	13300
121	13400
122	13500
123	13600
124	13700
125	13800
126	13900
126.1	13910

Table 19 (Page 5 of 5). Table for Mapping old rule number to new rule numbers. This table maps the rule numbers used in previous versions of the document to the new rule number used in this version.

Old Rule Number	New Rule Number
127	14000
128	14100
129	14200
130	14210
131	14220
131.1	14230
131.2	14240
131.3	14250
131.4	14260
131.5	14270
132	14300
133	14400
134	14500
135	14600
136	14700
137	14900
138	14910
139	15000
140	15010
141	15100
142	15200
145	15350
146	15410
147	15500
148	15600
149	15700
150	15800
150.1	15810
150.2	18810
150.3	18820
150.4	18830

Glossary

This glossary defines terms and abbreviations used in this publication. If you do not find the term you are looking for, refer to the index portion of this book or to the *IBM Dictionary of Computing*, New York: McGraw-Hill, 1994.

Sequence of Entries: For clarity and consistency of style, this glossary arranges the entries alphabetically on a letter-by-letter basis. In other words, only the letters of the alphabet are used to determine sequence; special characters and spaces between words are ignored.

Organization of Entries: Each entry consists of a single-word or multiple-word term or the abbreviation or acronym for a term, followed by a commentary. A commentary includes one or more items (definitions or references) and is organized as follows:

1. An item number, if the commentary contains two or more items.
2. A usage label, indicating the area of application of the term, for example, "In programming," or "In SMP/E." Absence of a usage label implies that the term is generally applicable to SMP/E, to IBM, or to data processing.
3. A descriptive phrase, stating the basic meaning of the term. The descriptive phrase is assumed to be preceded by "the term is defined as ..." The part of speech being defined is indicated by the opening words of the descriptive phrase: "To ..." indicates a verb, and "Pertaining to ..." indicates a modifier. Any other wording indicates a noun or noun phrase.
4. Annotative sentences, providing additional or explanatory information.
5. References, directing the reader to other entries or items in the dictionary.

References: The following cross-references are used in this glossary:

Contrast with. This refers to a term that has an opposed or substantively different meaning.

Synonym for. This indicates that the term has the same meaning as a preferred term, which is defined in its proper place in the glossary.

Synonymous with. This is a backward reference from a defined term to all other terms that have the same meaning.

See. This refers you to multiple-word terms that have the same last word.

See also. This refers the reader to related terms that have a related, but not synonymous, meaning.

Deprecated term for or Deprecated abbreviation for. This indicates that the term or abbreviation should not be used. It refers to a preferred term, which is defined in its proper place in the glossary.

Selection of Terms: A term is a word or group of words to be defined. In this glossary, the singular form of the noun and the infinitive form of the verb are the terms most often selected to be defined. If the term may be abbreviated, the abbreviation is given in parentheses immediately following the term. The abbreviation is also defined in its proper place in the glossary.

A

ACCEPT. In SMP/E, the process initiated by the ACCEPT command that places SYSMODs into the distribution libraries or permanent user libraries.

APPLY. In SMP/E, the process, initiated by the APPLY command, that places SYSMODs into the target libraries.

B

base function. A SYSMOD that defines elements of the base system or other products that were not previously present in the target libraries. Base functions are identified to SMP/E by the ++FUNCTION statement.

binder. A program that processes the output of language translators and compilers into an executable program (load module or program object). Part of DFSMS/MVS, it replaces the linkage editor and batch loader.

BYPASS. In SMP/E, an ACCEPT command operand that bypasses certain conditions to allow SMP/E processing of a SYSMOD to continue, regardless of the existence of that condition. For example, BYPASS (APPLYCHECK) indicates that all SYSMODs found in the PTS and not yet accepted are eligible for ACCEPT processing even if they have not been applied.

C

coexisting functions. Functions that can reside on the same system and be described by the same target zone.

common element. An element that is part of two different functions. It has the same name and type in each function. See also *element intersection*.

conditionally coexisting functions. Functions that coexist but do not have to be in the same zone.

consolidated software inventory (CSI). The primary SMP/E data set, which is divided into multiple partitions called zones. The three types of zones are the global zone, target zone, and distribution zone. A zone may be associated with one or more SRELS. Each zone contains information necessary for defining a system or subsystem and processing data for installing product function and service SYSMODs onto that system or subsystem.

CSI. Consolidated software inventory.

CSI distribution zone. See *distribution zone*.

CSI target zone. See *target zone*.

customization. Jobs or procedures required after “installation” before a product’s function can be used, or before a product’s service is effective.

D

data element. An element that is not a macro, module, or source code; for example, a dialog panel or sample code.

dependent function. A function that introduces new elements or redefines elements of the base-level system or other products. A dependent function cannot exist without a base function. Dependent functions are identified to SMP/E by the ++FUNCTION statement.

DFSMS environment. An environment that helps automate and centralize the management of storage. This is achieved through a combination of hardware, software, and policies. In the DFSMS environment for MVS, this function is provided by MVS/ESA SP and DFSMS/MVS, DFSORT, and RACF.

distribution library. Data sets supplied by the product packager containing one or more products that the user restores to disk for subsequent inclusion in a new system.

Distribution libraries are used as input to the SMP/E GENERATE command or the system generation

process to build target libraries for a new system. They are also used by SMP/E for backup when a running system has to be replaced or updated. In SMP/E, these data sets are updated by ACCEPT processing, and are identified with the DISTLIB operand.

distribution zone. In SMP/E, a group of VSAM records that describe the SYSMODs and elements in the distribution libraries.

DLIB. Distribution library.

E

element. In SMP/E, a macro, module, source module, data element, or element installed in a hierarchical file system (HFS).

element intersection. The existence of more than one element version in a given system or subsystem. See also *common element*.

element MCS. An MCS used to define a new or replacement element, or to update an existing element.

element names. An element naming structure for MVS that ensures there will be unique names within the system. This structure further ensures that no two elements have the same name unless they are equivalent or are different element types.

element selection. The process of choosing the appropriate modifications to an element from the SYSMODs selected by SMP/E for APPLY or ACCEPT processing from those that have elements in common.

element version. A specific module, macro, source module, or data element that represents one stage in the evolution of that element. The element version is identified by the FMID of the SYSMOD that contains the particular element version. Also see *versioned element*.

environment. The functions (FMIDs) that are installed on a particular system or subsystem (SREL).

F

FMID. Function modification identifier.

function. In SMP/E, a product (such as a system component or licensed program) that can be optionally installed in a user’s system. Functions are identified to SMP/E by the ++FUNCTION statement. Each function must have a unique FMID.

functionally higher SYSMOD. A SYSMOD that uses the function contained in an earlier SYSMOD (called the

functionally lower SYSMOD) and contains additional functions as well.

functionally lower SYSMOD. A SYSMOD whose function is also contained in a later SYSMOD (called the functionally higher SYSMOD).

function modification identifier (FMID). The FMID is the SYSMOD ID of a function SYSMOD and identifies the function that currently owns an element.

function SYSMOD. Any SYSMOD identified by the ++FUNCTION statement. The function SYSMOD is the SMP/E SYSMOD used for product base and dependent functions.

G

GENERATE. An SMP/E command used to create a job stream to build a set of target libraries from a set of distribution libraries.

global zone. A collection of records within the SMP/E CSI that contains information defining a common area that SMP/E uses to represent information not specific to a target zone or distribution zone. For example, the global zone is used to describe SYSMODs residing on the PTS. A CSI can contain only one global zone.

H

higher functional level. An element version that contains all of the functions of all other relevant versions of that element. See *functionally higher SYSMOD*.

I

indirect library. A partitioned data set used to package elements or JCLIN data instead of packaging them inline or in RELFILEs. Indirect libraries can be used if both of these conditions are met:

- The data set contains element replacements or JCLIN data (not element updates).
- Users who will be installing the SYSMOD have access to the data set.

See also *link library* and *text library*.

J

JCLIN. May be defined as any of the following:

- The SMP/E process of creating or updating the target zone using JCLIN input data.
- The data set that contains the Stage 1 output from a system, subsystem or product generation, used by SMP/E to update or create the target zone.
- The SMP/E JCLIN command used to read in the JCLIN data.
- The ++JCLIN Statement in a SYSMOD that enables SMP/E to perform the target zone updates during APPLY processing.

JCLIN data. The JCL statements associated with the ++JCLIN statement or saved in the SMPJCLIN data set. They are used by SMP/E to update the target zone when the SYSMOD is applied. Optionally, JCLIN data can be used by SMP/E to update the distribution zone when the SYSMOD is accepted.

L

licensed program (LP). Generally, a software package that can be ordered from an IBM Software Distribution Center. A licensed program may contain one or more function SYSMODs (FMIDs).

link library (LKLIB). A data set that contains link-edited object modules. It is used as an “indirect library” when the object modules are provided in partitioned data sets rather than inline or in relative files.

LKLIB. Link library.

load module. A computer program in a form suitable for loading into main storage for execution. It is usually the output of a linkage editor.

lower functional level. An element version that is contained in a later element version. See *functionally lower SYSMOD*.

LP. Licensed program.

M

macro. An instruction in a source language that is to be replaced by a defined sequence of instructions in the same source language.

MCS. Modification control statement.

modification control statement (MCS). An SMP/E control statement used to package a SYSMOD. These

statements describe the elements of a program and the relationships that program has with other programs that may be installed on the same system.

module. An element that is discrete and identifiable with respect to compiling, combining with other units, and loading (for example, the output from a compiler or assembler). Synonym for *object module* or *single-module load module*.

N

national language support (NLS). Product support may be required for multiple languages. This support will affect the design, packaging, and service of the product.

negative prerequisite. In SMP/E, a SYSMOD that must not be present in the system in order for another SYSMOD to be successfully installed.

NLS. National language support.

O

object deck. Object module input to the linkage editor that is placed in the input stream, in card format.

object module. A module that is the output from a language translator (such as a compiler or assembler). An object module is in relocatable format with machine code that is not executable. Before an object module can be executed, it must be processed by the linkage editor.

When an object module is link-edited, a load module is created. Several modules can be link-edited together to create one load module (for example, as part of SMP/E APPLY processing), or an object module can be link-edited by itself to create a single-module load module (for example, to prepare the module for shipment in RELFILE format or in an LKLIB data set or as part of SMP/E ACCEPT processing). This is also known as an *object deck*.

P

packaging. Adding the appropriate SMP/E modification control statements to elements to create a SYSMOD, then putting the SYSMOD in the proper format on the distribution medium.

partitioned data set extended (PDSE). A system-managed data set that contains an indexed directory and members that are similar to the directory and members of partitioned data sets. A PDSE can be used instead of a partitioned data set.

PDR. Process Documentation Reference.

PDSE. Partitioned data set extended.

PE-PTF. Program error PTF.

prerequisite. In SMP/E, a SYSMOD that must either be already installed or be installed along with another SYSMOD for that other SYSMOD to be successfully installed.

preventive service. The mass installation of PTFs to avoid rediscoveries of the APAR fixes included in those PTFs.

product. Generally, a software package, such as a licensed program or user application. A product may contain one or more functions and may consist of one or more versions and releases.

product function. See *function* and *function SYSMOD*.

program error PTF (PE-PTF). A PE-PTF is a PTF that was found to contain an error and is identified on a ++HOLD ERROR statement, along with the APAR that first reported the error.

program object. An executable program stored in a PDSE program library. A program object is similar to a load module, but has fewer restrictions.

program packaging. See *packaging*.

program product. Deprecated term for *licensed program*.

product service. See *service SYSMOD*.

program temporary fix (PTF). A temporary solution or bypass of a problem that may affect all users and that was diagnosed as the result of a defect in a current unaltered release of the program. PTFs are identified to SMP/E by the ++PTF statement.

PTF. Program temporary fix.

PTF in error. See *program error PTF*.

R

RECEIVE. In SMP/E, the process initiated by the RECEIVE command that reads SYSMODs and stores them on the PTS and CSI global zone for subsequent SMP/E processing.

regression. The condition that occurs when a modification is made to an element by a SYSMOD that is not

related to SYSMODs that previously modified the element.

REJECT. In SMP/E, the process initiated by the REJECT command that removes SYSMODs from the PTS and CSI global zone.

relative file (RELFILE) format. A SYSMOD packaging method in which elements and JCLIN data are in separate relative files from the MCSs. When SYSMODs are packaged in relative file format, there is a file of MCSs for one or more SYSMODs, and one or more relative files containing unloaded source-code data sets and unloaded link-edited data sets containing executable modules. Relative file format is the typical method used for packaging function SYSMODs.

relative files (RELFILES). Unloaded files containing modification text and JCL input data associated with a SYSMOD. These files are used to package a SYSMOD in relative file format.

release. A distribution of a new product or new function and APAR fixes for an existing product. Contrast with *version*.

RELFILE tape. The RELFILE tape contains one or more product functions in a format that can be installed on an MVS system or subsystem by SMP/E. It is a multiframe, standard label tape containing the SMP/E control statements for the functions and the data libraries for the functions.

replacement modification identifier (RMID). The modification identifier of the last SYSMOD to completely replace a given element.

requisite. A SYSMOD that must be installed before or at the same time as the SYSMOD being processed.

RESTORE. In SMP/E, the process initiated by the RESTORE command that removes applied SYSMODs from the target libraries, target zone and, optionally, the global zone.

RMID. Replacement modification identifier.

S

service level. The owner of the element (FMID), the last SYSMOD to replace the element (RMID), and all the SYSMODs that have updated the element since it was last replaced (UMIDs).

service order relationship. A relationship among service SYSMODs that is determined by the PRE and SUP operands, and the type of SYSMOD.

service SYSMOD. Any SYSMOD identified by an ++APAR or ++PTF statement.

shared load module. A load module containing multiple modules, some of which are owned by multiple FMIDs.

shared module. A module that is link-edited into more than one load module or dynamically accessed by more than one load module.

single-CSECT load module. See *single-module load module*.

single-module load module. A load module created by link-editing a single object module by itself (for example, to prepare the module for shipment in RELFILE format or in an LKLIB data set or as part of SMP/E ACCEPT processing).

SMP/E. System Modification Program Extended.

SMS. Storage Management Subsystem.

source module. An element containing the source statements that constitute the input to a language translator (such as a compiler or assembler) for a particular translation.

SREL. System release.

Storage Management Subsystem (SMS). A DFSMS/MVS or MVS/DFP facility used to automate and centralize the management of storage. Using SMS, a storage administrator describes data allocation characteristics, performance and availability goals, backup and retention requirements, and storage requirements to the system through data class, storage class, management class, storage group, and ACS routine definitions.

SYSMOD. System modification.

SYSMOD ID. System modification identifier.

SYSMOD packaging. See *packaging*.

SYSMOD relationships. Although individual SYSMODs can be installed independently, certain inter-SYSMOD relations must be observed if the results are to be meaningful. The following SYSMOD relationships are addressed in this publication:

- Unconditional
- Conditional
- Hierarchical
- Prerequisite
- Corequisite
- Negative prerequisite
- Delete
- Supersede

- Delete and supersede
- Coexistence

SYSMOD selection. The process of determining which SYSMODs are eligible to be processed by SMP/E.

system modification identifier (SYSMOD ID). The name that SMP/E associates with a system modification. It is specified on the ++APAR, ++FUNCTION, ++PTF, or ++USERMOD statement.

system modification (SYSMOD). A collection of software elements that can be individually distributed and installed. The SYSMOD is the input data to SMP/E that defines the introduction, replacement, or update of product function elements for SMP/E processing into target libraries and associated distribution libraries.

System Modification Program Extended (SMP/E). SMP/E is the IBM product designed to install new function and subsequent service into target libraries and distribution libraries.

system release (SREL). A 4-byte value representing a system or subsystem and its release level (for example, Z038 specifies MVS and C150 specifies CICS).

T

target library. A collection of data sets in which the various parts of an operating system are stored. This is sometimes called a system library. Target libraries contain the executable code that constitutes the running system. In SMP/E, these data sets are updated by APPLY processing, and are identified with the SYSLIB operand.

target zone. In SMP/E, a collection of VSAM records in the SMP/E CSI describing the SYSMODs, elements, and load modules in a target library.

text library (TXLIB). A data set containing JCLIN input or replacements for macros, source, or object modules that have not been link-edited. It is used as an “indirect library” when the JCLIN or elements are provided in partitioned data sets rather than inline or in relative files.

transformed data. Data that has been processed by the GIMDTS service routine so that it can be packaged inline in fixed-block 80 records.

TXLIB. Text library.

U

UCLIN. In SMP/E, the command used to initiate changes, through subsequent UCL statements, to the SMP/E database.

UMID. Update modification identifier.

unconditionally coexisting functions. Functions that coexist and must be in the same zone.

update modification identifier (UMID). The modification identifier of the SYSMOD that updated the last replacement of a given module, macro, or source module.

USERMOD. User modification.

user modification (USERMOD). A change constructed by a user to either modify an existing function, add to an existing function, or add a user-defined function. USERMODs are identified to SMP/E by the ++USERMOD statement.

V

version. A separate licensed program that is based on an existing licensed program and that usually has significant new code or new functions. Contrast with *release*.

versioned element. An element that is part of more than one function (for example, one that is part of a base function and a dependent function). See also *element version*.

Z

zone. A partition within an SMPCSI data set.

Bibliography and Classes

This section tells you more about the SMP/E publications and education on SMP/E that you might find helpful.

SMP/E Books in the OS/390 Library

Table 20 summarizes the books in the OS/390 library that apply to SMP/E.

<i>Table 20. Publications for OS/390 Version 2 Release 8 SMP/E</i>	
Title	Description
<i>OS/390 Release 7 Planning for Installation, GC28-1726</i>	Provides a plan for installing OS/390, including SMP/E
<i>OS/390 SMP/E Diagnosis Guide, SC28-1737</i>	Explains how to handle suspected SMP/E problems
<i>OS/390 SMP/E Messages and Codes, SC28-1738</i>	Explains SMP/E messages and return codes and the actions to take for each
<i>OS/390 SMP/E User's Guide, SC28-1740</i>	Describes how to use SMP/E to install programs and service. Also contains information formerly found in <i>SMP/E Primer</i> , which introduces the basic principles needed for using SMP/E, without the expert-level details found in other SMP/E publications.
<i>OS/390 SMP/E Commands, SC28-1805</i>	Explains SMP/E commands and processing in detail
<i>OS/390 SMP/E Reference, SC28-1806</i>	Provides additional SMP/E reference material
<i>OS/390 SMP/E Reference Summary, SX22-0037</i>	Reviews the SMP/E commands in a convenient form
<i>Standard Packaging Rules for MVS-Based Products, SC23-3695</i>	Explains how to package programs for installation by SMP/E

The SMP/E Release 8.1 Library

Table 21 summarizes the books in the SMP/E Release 8.1 library.

<i>Table 21 (Page 1 of 2). Publications for SMP/E Release 8.1</i>	
Title	Description
<i>SMP/E R8.1 Primer, GC23-3771</i>	Introduces the basic principles needed for using SMP/E, without the expert-level details found in other SMP/E publications
<i>SMP/E R8.1 Master Index, GC23-3812</i>	Helps users quickly determine which book in the SMP/E library contains the information they are looking for
<i>SMP/E R8.1 Program Directory (English Feature), GC23-0130</i> <i>SMP/E R8.1 Program Directory (Japanese Feature), GC23-0469</i>	Explains how to plan for installing SMP/E Release 8.1
<i>SMP/E R8.1 User's Guide, SC28-1302</i>	Describes how to use SMP/E to install programs and service
<i>SMP/E R8.1 Messages and Codes, SC28-1108</i>	Explains SMP/E messages and return codes and the actions to take for each

<i>Table 21 (Page 2 of 2). Publications for SMP/E Release 8.1</i>	
Title	Description
<i>SMP/E R8.1 Reference, SC28-1107</i>	Explains SMP/E commands and processing in detail
<i>SMP/E R8.1 Reference Summary, SX22-0006</i>	Reviews the SMP/E commands in a convenient form
<i>SMP/E R8.1 CBIPO Dialogs User's Guide, SC23-0538</i>	Explains how to use the CBIPO dialogs to install, reinstall, and redistribute CBIPO orders
<i>Migrating to the CBIPO Dialogs, GC23-3810</i>	Compares functions provided in the batch CBIPO installation method with functions provided in the CBIPO dialogs contained in SMP/E
<i>SMP/E R8.1 Diagnosis Guide, SC23-3130</i>	Explains how to handle suspected SMP/E problems
<i>Standard Packaging Rules for MVS-Based Products, SC23-3695</i>	Explains how to package programs for installation by SMP/E

Classes and Self-Study Courses for SMP/E

Table 22 shows the recommended education on CBIPOs, CBPDOs, and SMP/E that is offered through the various IBM locations.

<i>Table 22 (Page 1 of 2). Classes and Self-Study Courses</i>			
Location	Recommended Education	Catalog of Courses	Phone Number for More Information
Australia	<ul style="list-style-type: none"> • "SMP/E: A Guide for the New SMP/E User" (Self-Study Course 32186) • "SMP/E Fundamentals" (Course H3765) • "Integrated System Maintenance Using SMP/E" (Course H3763) • "MVS Installation and Tailoring" (Course H3903) • All prerequisites for the above classes, or equivalent experience 	Contact your local branch office.	Contact your local branch office.
Canada	<ul style="list-style-type: none"> • "SMP/E: A Guide for the New SMP/E User" (Self-Study Course 32186) • "New SMP Users" (Course S4716) • "MVS Installation and Tailoring" (Course S6375) • All prerequisites for the above classes, or equivalent experience 	<i>Education Course Catalogue, G209-0073 (bilingual version) or G209-0062 (English version)</i>	IBM Direct-Education at 1-800-465-1234
EMEA	<ul style="list-style-type: none"> • "SMP/E: A Guide for the New SMP/E User" (Self-Study Course 32186) • "System Installation and Maintenance with SMP/E" • "MVS/XA Installation Practice and Procedure" or "MVS/ESA Installation and Implementation" • "MVS/ESA Customization" • All prerequisites for the above classes, or equivalent experience 	See your country's education course catalog.	See your country's education course catalog for enrollment procedures.

Table 22 (Page 2 of 2). Classes and Self-Study Courses

Location	Recommended Education	Catalog of Courses	Phone Number for More Information
Japan	<ul style="list-style-type: none"> • "How to Use SMP/E" (Self-Study Course 25024) • "MVS Installation and Tailoring" (Course H3903) • "MVS/ESA Installation" (Course 24226) • "MVS/ESA Customization" (Course 24228) • All prerequisites for the above classes, or equivalent experience 	<i>Catalog of IBM Education, GR18-5200</i>	IBM DIRECT at 03-865-5748
United States	<ul style="list-style-type: none"> • "SMP/E: A Guide for the New SMP/E User" (Self-Study Course 32186) • "SMP/E Fundamentals" (Course H3765) • "Integrated System Maintenance Using SMP/E" (Course H3763) • "MVS Installation and Tailoring" (Course H3903) • All prerequisites for the above classes, or equivalent experience 	<i>Catalog of IBM Education, G320-1244</i>	IBM DIRECT at 1-800-IBM-TEACH

Index

Special Characters

- ++DELETE MCS
 - overview 81
 - packaging rules 81
 - superseding SYSMODs 64
- ++element MCS
 - See also* ++hfs_element MCS
 - See also* ++MAC MCS
 - See also* ++MOD MCS
 - See also* ++SRC MCS
 - See also* data element MCS
 - operands
 - DELETE 83
 - VERSION 84
 - overview 43, 73
 - restrictions
 - software delivery process 44
 - superseding SYSMODs
 - VERSION operand rules 63
 - use in this book xiii
- ++FUNCTION MCS
 - copyright comment rules 55
 - operands
 - REWORK 54
 - RFDSNPF 54
 - SYSMOD ID 54
 - overview 54
- ++hfs_element MCS
 - description 71
- ++HOLD MCS
 - superseding SYSMODs 63
- ++IF MCS
 - examples
 - avoiding loss of PTF for previous release 140
 - corequisite dependent functions 153
 - cross-product prerequisite for service 154
 - cross-product service 155
 - operands
 - FMID 67
 - REQ 67
 - overview 67
 - saving data from 68
 - superseding SYSMODs 63
- ++JCLIN MCS
 - operands
 - RELFILE 85
 - packaging rules 89
 - superseding SYSMODs 64
- ++MAC MCS
 - description 71
- ++MAC MCS (*continued*)
 - overview 43, 73
 - packaging rules 44
 - restrictions
 - software delivery process 44
- ++MOD MCS
 - description 71
 - operands
 - CSECT 84
 - overview 43, 73
 - packaging rules 44
 - superseding SYSMODs
 - CSECT operand rules 64
 - LMOD operand 64
- ++MOVE MCS
 - overview 75
 - packaging rules 75
 - superseding SYSMODs 63
- ++RENAME MCS
 - overview 78
 - packaging rules 78
 - superseding SYSMODs 64
- ++SRC MCS
 - description 71
 - overview 43, 73
 - packaging rules 44
 - restrictions
 - software delivery process 44
- ++VER MCS
 - examples
 - cross-product service 143
 - defining base and dependent functions 146
 - defining mutually exclusive functions 156
 - deleting a function 144, 152
 - establishing the order of dependent functions 152
 - fixing an erroneous post-cutoff PTF 141
 - replacing a function 141
 - superseding an APAR 138
 - operands
 - DELETE 57
 - FMID 57
 - NPRES 60
 - PRE 60
 - SUP 61
 - VERSION 64
 - overview 56
 - superseding SYSMODs
 - PRE operand rules 63
 - REQ operand rules 63
 - SUP operand rules 63
 - VERSION operand rules 63

A

- adding elements 162
- alias names
 - elements 48, 123
 - load modules 48, 123
 - naming conventions 122
 - restrictions 123
- ALIAS statement
 - link-edit step
 - JCLIN processing 97
- APAR fixes
 - avoiding regression of 138
 - corrective service 24
 - overview 24
 - superseding with a PTF 138
- APPLY processing
 - processing non-zero return code 95, 187
 - products requiring assemblies during 90
 - using the etc/ directory during 136, 191
- ASMA90 utility
 - See assembler utility
- assembler utility
 - rules for JCLIN data 90
- autocall
 - See automatic library call function
- automatic library call function
 - JCLIN for 117
 - LIBRARY statement to exclude modules from automatic library search 102
 - packaging support for high-level languages 131
 - SYSLIB DD statement in link-edit steps 104

B

- base functions
 - compared with dependent functions 22
 - defining 146
 - deleting 141
 - deleting and superseding 144, 154
 - naming conventions 121
 - overview 21
- BLKSIZE
 - See block size
- block size
 - data elements 36
 - distribution libraries 37
 - hierarchical file system (HFS) elements 36
 - macros 36, 37
 - modules 36
 - RELFILEs 14
 - source 36
 - target libraries 37
- book titles xi

build process 3

C

- callable services
 - including modules from another product 104
- CALLLIBS
 - cross-product referencing
 - JCLIN 95, 187
 - resolving external references 117
 - using with optional functions 95, 187
- CBIPOs
 - introduction 2
- CBPDOs
 - introduction 2
- ccc (product version code) 121
- CHANGE statement
 - JCLIN processing 101
- coexisting functions 29
- coexisting SYSMODs 29, 30
- combining elements 162
- common elements among SYSMODs
 - defining SYSMOD relationships 152
- component code 121
 - element names, first 3 characters of 122, 188
- conditional relationships 25, 67
- copy utility
 - rules for JCLIN data 91, 92
- copyright requirements
 - ++FUNCTION MCS 55
 - comment 55
- corequisite SYSMODs
 - defining with the REQ operand 67
 - description 27
 - examples
 - corequisite dependent functions 153
 - cross-product service for a base function with a prerequisite 155
 - cross-product service for corequisite base functions 143
 - defining a chain of requisite PTFs 149
 - erroneous post-cutoff PTF 141
 - saving fixes for previous releases 140
- corrective service
 - APAR fixes 24
 - PTFs 23
- cross-product relationships
 - corequisite dependent functions 153
 - prerequisites for functions 153
 - prerequisites for service 154
 - service for a base product with a prerequisite 155
 - service for corequisite base functions 143
- CSECT
 - ++MOD MCS operand 64, 84
 - module that contains 44

CSECT (*continued*)

- specifying for a load module 84
- specifying order through ORDER statement 97
- superseding SYSMODs 64

customizing functions via USERMODs 24

D

data element MCS

- description 71

data elements

- formatting for inline packaging 16
- LRECL values allowed 38, 170
- RECFM values allowed 38, 170
- translated 128
- types 45

data set names

- distribution libraries 124
- RELFILE tape 15
- target libraries 124

DDDEF entries

- jobs creating 51
- PTFs modifying 51
- sample jobs, creating 51

DDDEFs

ddnames

- distribution libraries 102, 124
- SYSPUNCH usage 102
- target libraries 124

DELETE

- ++element MCS operand 83
- ++VER MCS operand 57
- example 144

deleted modules, reintroducing 58

deleting elements 83

deleting functions

- See also* deleting SYSMODs
- defining with the DELETE operand 57
- dummy function SYSMOD 59
- explicit deletion 23

deleting load modules 81

deleting SYSMODs

- See also* deleting functions
- description 27
- examples

- base function 141
- comparison of deleting and superseding 27
- dependent function 152
- dummy function SYSMOD to delete another function 59
- function with a corequisite 144

dependent functions

- common elements
- defining the lower level 65, 84, 85, 176, 184
- lower-level dependent function for same parent base function 65, 176

dependent functions (*continued*)

- compared with base functions 22
- defining 146
- deleting 152
- establishing the order of 152
- naming conventions 121
- overview 22
- packaging example 35
- restrictions

- elements 37

distribution libraries

- considerations 72, 179
- creating 37, 170
- data set name for new library 124
- ddname
- JCLIN requirements 89, 184
- new library 124
- naming conventions 124

dummy function SYSMOD to delete another function 59

Dynamic Allocation

E

editing MCS statements 89, 184

element statements

- packaging rules 44

element updates

- superseding SYSMODs 64

elements

- adding 162
- aliases for 48, 123
- combining 162
- from different base functions 143, 155
- from different dependent functions 153
- common
- corequisite PTFs for 147
- defining higher levels of using SYSMOD types 25
- defining lower levels of (++element VERSION) 84, 85, 184
- defining lower levels of (++VER VERSION) 65, 176
- dependent functions for the same base function 65, 176
- definition of 1
- deleting 83
- fully-defined 93
- MCS statements 43, 71, 73
- migrating
- updating both functions 163
- using a PTF 164
- moving 75
- naming conventions 122, 188
- ownership 45

elements (*continued*)

- restrictions
 - dependent functions 37
- same names 123
 - restrictions for RELFILES 13, 167
 - supporting different languages 127, 190
- translated 128
- types 45, 47
- versions 84
 - creating with dependent functions, overview 85
 - creating with PTFs, overview 85

enhancing a function, packaging example 35

ENTRY statement

- defining a load module entry 97
- for PL/I load modules 97
- JCLIN processing 101

excluding modules from automatic library search 102

EXPAND statement

- JCLIN processing 101

explicitly deleting functions

- packaging options 23, 58

external references

- resolving through SYSLIB allocation and CALLLIBS 117

F

files, relative (RELFILES)

- See RELFILE tape

FMID

- ++IF MCS operand 67
- ++VER MCS operand 57
- adding new 36
- description of 21
- naming conventions 121
- product version code 121
- release value 121

fully-defined elements 93

function

- See function SYSMODs

function SYSMODs

- base functions 21
- choosing between base and dependent functions 22
- copyright statement for 55
- dependent functions 22
- installation hierarchy 25
- installation overview 2
- JCLIN data 87
- naming conventions 121
- RELFILE requirements 11
- repackaging
 - indicated by REWORK operand 54
 - new release of a conditional requisite 68, 178
- UCLIN
 - avoiding 41

functions

- packaged as function SYSMODs 21

G

GIMDTS, used to format elements for inline packaging 16

H

HFS elements

- LRECL values allowed 38, 170
- RECFM values allowed 38, 170

hierarchical file system (HFS)

- creating symbolic links in MKDIR jobs 51
 - ddname 89, 184
 - elements residing in 71
 - load modules residing in
 - JCLIN for 119
 - LIBRARYDD comment 105, 106
 - SYSLIB DD statement in link-edit steps 105
 - SYSLMOD DD statement in link-edit steps 106
 - not allowed as distribution library 72, 95, 179, 186
 - packaging rules 136, 191
 - permission bits 136, 191
 - products installing into 50, 51, 172
- hierarchical file system (HFS) elements
- formatting for inline packaging 16
 - MCS statement 71
 - translated 128
 - types 47

hierarchical file system (HFS) elements

hierarchy of SYSMOD types 25

high-level languages

- including modules from another product 104

I

IDENTIFY statement

- JCLIN processing 101

IEBCOPY utility

- See copy utility

IEWBLINK utility

- See link-edit utility

IEWL utility

- See link-edit utility

implicitly deleting SYSMODs

- packaging options 58

implicitly including modules from another product 104

implicitly-included modules

- including through SYSLIB allocation and CALLLIBS 117

INCLUDE statement

- JCLIN processing 101
- utility input 101

indirect libraries 17
 initial release of a function, packaging example 35
 inline packaging 15
 INSERT statement
 JCLIN processing 102
 installation
 customizing a function 24
 hierarchy 25
 JCL, RELFILE member for 50, 172
 methods 33
 overview
 functions 2
 service 2
 integration process 3
 IVP (installation verification procedure)
 RELFILE member for 37, 38

J

JCL for installation
 RELFILE member for 50, 172
 JCLIN command
 processing link-edit steps
 creating LMOD entry 103
 creating MOD entry 101
 JCLIN data
 assembler rules 90
 copy rules 91, 92
 examples
 assembling source to create a module 116
 load modules residing in a hierarchical file system
 (HFS) 119
 load modules using the link-edit automatic library
 call function 117
 macros 116
 modules 112
 source 116
 fully-defined elements 93
 functions, providing for 87
 link-edit rules 95, 96
 not required 88
 packaging recommendations 96
 packaging rules 89
 RELFILE requirements 13, 168
 superseding SYSMODs 64

L

language abbreviations 128
 See also NLS (national language support)
 language-sensitive elements
 See also NLS (national language support)
 packaging examples 157
 libraries
 See also distribution libraries

libraries (*continued*)
 See also target libraries
 naming conventions 124
 sharing 39
 totally copied 93
 LIBRARY statement
 JCLIN processing 102
 LIBRARYDD comment for pathname in link-edit
 steps 105, 106
 link-edit autocall
 See automatic library call function
 link-edit automatic library call function, JCLIN for
 See automatic library call function
 link-edit utility
 automatic library call function 131
 parameters recognized by SMP/E 106
 rules for JCLIN data 95, 96
 linkage editor
 See link-edit utility
 LMOD
 superseding SYSMODs
 ++MOD MCS operand rules 64
 LMOD entry
 See also load modules
 created by JCLIN 103
 load modules
 aliases for 48, 123
 defining the ENTRY point 97
 deleting 81
 description 48
 external references
 JCLIN for 117
 hierarchical file system (HFS)
 JCLIN for 119
 LIBRARYDD comment in link-edit steps 105,
 106
 SYSLIB DD statement in link-edit steps 105
 SYSLMOD DD statement in link-edit steps 106
 including modules from other distribution
 libraries 97
 moving 75
 moving shared load modules 76
 naming conventions 122
 packaging
 automatic library call function, JCLIN for 117
 RECFM=U required 37
 renaming 78
 renaming shared load modules 79
 sharing 48
 single-CSECT 93
 logical record length
 data elements 36, 38, 170
 HFS elements 38, 170
 hierarchical file system (HFS) elements 36
 macros 36, 37

logical record length (*continued*)

modules 36
source 36, 37

LRECL value

See logical record length

M

macros

deleting 83
LRECL=80 required 37
MCS statement 43, 71, 73
moving 75

MCS statements

++DELETE 81
++FUNCTION MCS 54
++hfs_element 71
++IF 67
++MAC 43, 71, 73
++MOD 43, 71, 73
++MOVE 75
++RENAME 78
++SRC 43, 71, 73
++VER 56

data element 71
order of 53

packaging rules 53

middle-level qualifiers

specifying on sample allocation jobs 51

migrating elements

updating both functions 163
using a PTF 164

MKDIR

EXECs

naming 51, 52

jobs, creating symbolic links in 51

MOD entry

created by JCLIN 101

modules

See *also* load modules

adding to a new load module 73

adding to an existing load module 73

block size 36

CSECTs in 44, 84

deleting 83

including from another product's load module 107

including into another product's load module 108

logical record length 36

MCS statement 43, 71, 73

moving 75

record format 36

sharing 48

moving elements and load modules 75

moving shared load modules 76

multiple languages

See NLS (national language support)

multiple physical tapes 14

mutually exclusive functions 156

N

NAME statement

JCLIN processing 103

naming conventions 121

alias names 122

elements 122

libraries 124

load modules 122

SYSMOD IDs 121

functions (FMIDs) 121

national language identifiers 128

See *also* NLS (national language support)

national language support

See NLS (national language support)

negative prerequisite SYSMODs

defining with the NPRES operand 60

description 27

examples 156

using to specify mutually exclusive SYSMODs 27,
169

NLS (national language support)

language abbreviations 128

naming of element types 123

packaging options 127

single base function 128

restrictions

DELETE operand on ++VER MCS 58, 174

element considerations 52

NPRES

++VER MCS operand 60

using to specify mutually exclusive SYSMODs 27,
60, 169

O

object modules

See modules

order of SYSMOD types 25

ORDER statement

JCLIN processing 103

using to specify CSECT order 97

OVERLAY statement

JCLIN processing 102

ownership

++VER MCS 64

for function SYSMODs 64

of elements 45

packaging considerations 45

P

packaging

- evolution of 1
- examples 35
- language-sensitive elements 127

packaging rules 53

- ++DELETE MCS 81
- ++FUNCTION REWORK 54
- ++FUNCTION sysmod_id 36
- ++IF MCS
 - FMID 67
 - REQ 68
- ++MAC MCS 44
- ++MOD MCS 44
 - CSECT 84
- ++MOVE MCS 75
- ++RENAME MCS 78
- ++SRC MCS 44
- ++VER MCS 56, 57, 172
 - DELETE 58
 - FMID 57
 - NPRES 60
 - PRE 61
 - REQ 64
 - SREL 57
 - SUP 62
 - VERSION 65

data element statements 45

editing MCS statements 89, 184

element names 122

element statements 44

- DELETE 83
- DISTLIB 72
- VERSION 85

functions 53

JCLIN data

- assembler steps 90
- copy steps 91
- introduction 89
- link-edit steps 95

language-sensitive elements 52

library names 124

UCLIN changes 74

versions, mutually exclusive 60

packaging SYSMODs

See also SYSMOD packaging

inline JCLIN 15

PATH

- changing for an existing dataset 37
- operand for HFS pathname 105, 106

pathname

- not allowed as distribution library 72, 95, 179, 186

PDS vs. PDS/E

- changing for an existing dataset 37

physical tapes, multiple 14

PL/I, using ENTRY statements 97

planning considerations

- component code 121
- prefix for names of product elements and load modules 121
- product identifiers 121
- SYSMOD IDs 121

PRE

- ++VER MCS operand 60
- superseding SYSMODs 63

prefix for names of product elements and load modules 121

prefix for relative file data sets

See RFDSNPF

prerequisite SYSMODs

- defining with the PRE operand 60
- defining with the REQ operand 67
- description 26
- examples
 - defining base and dependent functions 146
 - defining cross-product prerequisites for functions 153
 - defining cross-product prerequisites for service 154
 - defining service for a function 138
 - defining service that depends on previous service 139
 - establishing the order of dependent functions 152
- overview 26

preventive service (PTFs) 23

product identifiers 121

product processes

introduction to 1

product version code (ccc) 121

PTF

- common elements 147
- corrective service 23
- cross-product requisites 143, 155
- cutoff dates
 - fixing an erroneous post-cutoff PTF 141
- missing ++IF MCS 141
- overview 23
- preventive service 23
- requisite chain 147, 149
- saving fixes for previous releases 140
- superseding an APAR 138

PTFs

- load modules, adding or changing 111

R

RECFM value

See record format

- record format
 - changing for an existing dataset 37
 - data elements 38, 170
 - distribution libraries 37
 - HFS elements 38, 170
 - hierarchical file system (HFS) elements 36
 - load modules 37
 - source 37
 - target libraries 37
 - regression, avoiding
 - for mispackaged requisites 141
 - superseding the lower-level SYSMOD 138
 - relationships between SYSMODs 24
 - relative files (RELFILES)
 - See also* RELFILE tape
 - defining
 - prefix for data set names 54
 - prefix for data sets 54
 - release number
 - FMID 121
 - RELFILE
 - ++JCLIN MCS operand 85
 - RELFILE tape
 - construction rules 11, 13
 - contents 12
 - creating 15
 - data set names 15
 - defining JCLIN data 85
 - defining relative file number 85
 - examples 12
 - format 12
 - packaging requirements 11
 - restrictions
 - software delivery process 11, 191
 - volume serial numbers 15
 - renaming load modules 78
 - renaming shared load modules 79
 - REPLACE statement
 - JCLIN processing 103
 - replacing a function, packaging example 35
 - replacing elements by superseding them 63
 - replacing SYSMODs via ++VER DELETE 57
 - REQ
 - ++IF MCS operand 67
 - ++VER MCS operand
 - superseding SYSMODs 63
 - requisite SYSMODs
 - See also* ++IF MCS
 - See also* corequisite SYSMODs
 - See also* negative prerequisite SYSMODs
 - See also* prerequisite SYSMODs
 - conditional 25
 - unconditional 25
 - REWORK
 - ++FUNCTION MCS operand 54
 - RFDSNPFX
 - ++FUNCTION MCS operand 54
 - IBM not allowed 55, 172
 - relationship to RELFILE data set name 15, 168
 - rrr (release value)
 - FMID 121
 - rule numbers
 - changes to xv
 - mapping of old to new 237
- ## S
- sample code, packaging
 - installation JCL 50, 172
 - recommendations 37
 - sample jobs
 - considerations 51
 - SELECT statement
 - when to use for copy operations 92
 - servicing a function, packaging example 36
 - shared
 - libraries 39
 - load modules 48
 - moving 76
 - renaming 79
 - single-CSECT modules 44, 93
 - sizes
 - block sizes 36
 - logical record length 36
 - record formats 36
 - SMPLTS
 - not allowed as DISTLIB or SYSLIB on MCS 72, 179
 - SMPMCS
 - specifying as the dataset name of the first file 15
 - SMPMTS
 - not allowed as DISTLIB or SYSLIB on MCS 72, 179
 - SMPOBJ
 - JCLIN processing 102
 - SMPPTS
 - not allowed as DISTLIB or SYSLIB on MCS 72, 179
 - SMPSTS
 - not allowed as DISTLIB or SYSLIB on MCS 72, 179
 - software delivery process
 - restrictions
 - ++element MCS 44
 - ++MAC MCS 44
 - ++SRC MCS 44
 - source
 - LRECL=80 required 37
 - MCS statement 43, 71, 73
 - moving 75

-
- source (*continued*)
 - RECFM=FB required 37
 - SREL 57
 - packaging rules 57
 - SUP
 - ++VER MCS operand 61
 - superseding SYSMODs 63
 - superseding SYSMODs 27
 - ++DELETE MCS 64
 - ++element MCS
 - VERSION operand rules 63
 - ++HOLD MCS 63
 - ++IF MCS 63
 - ++JCLIN MCS 64
 - ++MOD MCS
 - CSECT operand rules 64
 - LMOD operand 64
 - ++MOVE MCS 63
 - ++RENAME MCS 64
 - ++VER MCS
 - PRE operand rules 63
 - REQ operand rules 63
 - SUP operand rules 63
 - VERSION operand rules 63
 - APARs, examples 138
 - description 27
 - element updates 64
 - functions
 - defining with the SUP operand 61
 - JCLIN data 64
 - SUP operand on the ++VER MCS 61
 - UCLIN data 64
 - SYSDEFSD DD statement
 - JCLIN processing 104
 - SYSLIB DD statement
 - JCLIN processing 104
 - link-edit steps 105
 - PATH operand for HFS pathname 105
 - resolving external references 117
 - SYSLMOD DD statement
 - JCLIN processing 106
 - link-edit steps 106
 - PATH operand for HFS pathname 106
 - SYSMOD IDs 121
 - SYSMOD packaging
 - See also* packaging rules
 - examples 137
 - indirect libraries
 - LKLIB 17
 - TXLIB 17
 - LKLIB 17
 - methods of 11
 - indirect libraries 17
 - inline data 15
 - relative files (RELFILES) 11
 - SYSMOD packaging (*continued*)
 - relative files (RELFILES) 11
 - TXLIB 17
 - SYSMOD relationships
 - coexisting SYSMODs 29
 - corequisite SYSMODs 27
 - deleting 27
 - negative prerequisite SYSMODs 27, 169
 - prerequisite SYSMODs 26
 - superseding 27
 - sysmod_id
 - ++FUNCTION MCS operand 54
 - SYSMODs
 - adding new FMIDs 36
 - APAR fixes 24
 - definition of 1
 - deleting 27
 - evaluating relationships 35
 - functions 21
 - hierarchy 25
 - naming conventions 121
 - overview 21
 - packaging
 - definition of 1
 - PTFs 23
 - relationships 24
 - conditional 25
 - deleting SYSMODs 27
 - requisite 25
 - unconditional 25
 - rules for packaging 53
 - superseding 27
 - USERMODs 24
 - SYSYPUNCH
 - considerations 72, 179
 - JCLIN processing 102
- ## T
- target libraries
 - data set name for new library 124
 - ddname
 - JCLIN requirements 89, 184
 - new library 124
 - totally copied library 93
 - translation
 - See* NLS (national language support)
- ## U
- UCLIN changes
 - avoiding use of 40
 - packaging rules 74
 - superseding SYSMODs 64
-

Index

unconditional relationships 25

USERMOD

customizing a function 24

overview 24

V

VERSION

++element MCS operand 84

superseding SYSMODs 63

++VER MCS operand 64

superseding SYSMODs 63

data element MCS operand 84

versions of elements 84

versions of functions

mutually exclusive, packaging rules for 60

volume serial numbers

RELFILE tape 15

W

WAITFORDSN operand

specifying on DDDEF jobs 51

Z

zones

++VER rules 56, 172

coexisting SYSMODs 29

mutually exclusive functions in a zone 60

negative prerequisite 60

updated by JCLIN data 87

Communicating Your Comments to IBM

Software Delivery
Standard Packaging Rules for
MVS-Based Products
Publication No. SC23-3695-09

If you especially like or dislike anything about this book, please use one of the methods listed below to send your comments to IBM. Whichever method you choose, make sure you send your name, address, and telephone number if you would like a reply.

Feel free to comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. However, the comments you send should pertain to only the information in this manual and the way in which the information is presented. To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer.

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.

If you are mailing a reader's comment form (RCF) from a country other than the United States, you can give the RCF to the local IBM branch office or IBM representative for postage-paid mailing.

- If you prefer to send comments by mail, use the RCF at the back of this book.
- If you prefer to send comments by FAX, use this number:
 - FAX: (International Access Code)+1+914+432-9405
- If you prefer to send comments electronically, use one of these network IDs:
 - IBM Mail Exchange: USIB6TC9 at IBMMAIL
 - Internet e-mail: mhvrcfs@us.ibm.com
 - World Wide Web: <http://www.ibm.com/s390/os390/>

Make sure to include the following in your note:

- Title and publication number of this book
- Page number or topic to which your comment applies

Optionally, if you include your telephone number, we will be able to respond to your comments by phone.

Reader's Comments — We'd Like to Hear from You

Software Delivery Standard Packaging Rules for MVS-Based Products

Publication No. SC23-3695-09

You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you. Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate.

Note: Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.

Today's date: _____

What is your occupation?

Newsletter number of latest Technical Newsletter (if any) concerning this publication:

How did you use this publication?

- | | | | |
|--------------------------|-------------------------------|--------------------------|------------------------|
| <input type="checkbox"/> | As an introduction | <input type="checkbox"/> | As a text (student) |
| <input type="checkbox"/> | As a reference manual | <input type="checkbox"/> | As a text (instructor) |
| <input type="checkbox"/> | For another purpose (explain) | | |

Is there anything you especially like or dislike about the organization, presentation, or writing in this manual? Helpful comments include general usefulness of the book; possible additions, deletions, and clarifications; specific errors and omissions.

Page Number: Comment:

Name

Address

Company or Organization

Phone No.



Cut or Fold
Along Line

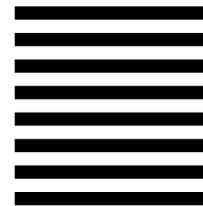
Fold and Tape

Please do not staple

Fold and Tape



NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES



BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
Department 55JA, Mail Station P384
522 South Road
Poughkeepsie NY 12601-5400



Fold and Tape

Please do not staple

Fold and Tape

Cut or Fold
Along Line



Program Number: 5645-001



Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SC23-3695-09

