Interactive System Productivity Facility
(ISPF)

Software Configuration and Library Manager (SCLM) Project Manager's and Developer's Guide

z/OS Version 1 Release 10

SC34-4817-00

Interactive System Productivity Facility (ISPF)

Software Configuration and Library Manager (SCLM) Project Manager's and Developer's Guide

z/OS Version 1 Release 10

SC34-4817-00

Note
Before using this document, read the general information under "Notices" on page 313.

First Edition (March 2001)

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

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

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

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

International Business Machines Corporation
Software Reengineering
Department G7IA / Building 503
Research Triangle Park, NC 27709-9990

FAX (United States & Canada): 1+800+227-5088
IBMLink (United States customers only): CIBMORCF@RALVM17
IBM Mail Exchange: USIB2HPD@VNET.IBM.COM
Internet: USIB2HPD@VNET.IBM.COM

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

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

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

The ISPF development team maintains a site on the World-Wide Web. The URL for the site is:

http://www.software.ibm.com/ad/ispf

Contents

Preface

This book provides reference and usage information, along with conceptual and functional descriptions of the Software Configuration and Library Manager (SCLM). This book also contains step-by-step information for setting up and maintaining an SCLM project environment. It describes how to establish and monitor a database and explains the library functions.

Who Should Use This Book

This book is for application developers whose projects are controlled by SCLM. This book is also for project managers who use SCLM to manage the development process.

What Is in This Book

This manual assumes that you are familiar with the operation of ISPF in the MVS environment.

Part One of this book is the **Project Manager's Guide**:

Chapter 1. Defining the Project Environment, describes how to generate a project definition. It explains the steps that enable you to create the database that best meets the needs of your project. The chapter includes step-by-step instructions for setting up the SCLM sample project included with the ISPF product. After completing the steps described in this chapter, you can experiment with basic SCLM operations using the sample project hierarchy.

Chapter 3. Additional Project Manager Tasks, describes additional tasks that project managers perform to maintain SCLM projects. This chapter discusses backing up and recovering a project database, using authorization codes to control SCLM operations, developing and maintaining projects concurrently, and implementing verification and exit routines for SCLM projects.

Chapter 4. Converting Projects to SCLM, describes the steps required to convert existing ISPF software development projects to SCLM.

Chapter 5. Language Definition Considerations describes setup operations you must perform to create a language definition for SCLM to use.

Defining a New Language to SCLM, describes the control structures used to manage SCLM functions and illustrates how to define new languages. It also contains information on converting JCL decks to language definitions.

Chapter 6. Using SCLM and Tivoli Service Desk for OS/390, illustrates the interaction between SCLM and Information Manager through the use of a sample program.

Chapter 7. Understanding and Using the Customizable Parsers, describes the REXX parsers supplied with SCLM and provides examples of how to customize them.

Part Two of this book is the **Developer's Guide**:

Chapter 8. The Software Configuration and Library Manager, provides information on the SCLM project database and the terminology used. The chapter describes the library structure and naming conventions used when you define and maintain SCLM projects.

Chapter 9. Using SCLM Functions, describes how to use the ISPF dialog interface, select SCLM functions to retrieve or process certain information, and generate reports on the information stored in project databases. It also describes information stored in accounting, cross-reference, and intermediate records for members in the project databases.

Chapter 10. Development Scenario, is a programmer scenario that describes the tasks typically performed by SCLM users. This chapter provides step-by-step instructions on how to use the basic SCLM functions to control development projects.

Chapter 11. Architecture Definition, describes architecture configuration and dependency control statements and their uses. It provides examples of each kind of architecture member and describes the special command statements that the architecture members require. It also provides an example of the format of each statement and lists any restrictions.

Chapter 12. Managing Complex Projects, describes advanced topics that aid in managing complex configurations.

The **Appendixes** offer advanced reference material:

Appendix. SCLM Variables and MetaVariables lists SCLM variables and metavariables used in various stages of SCLM processing.

Chapter 13. SCLM Support for DB2, General Information, describes how to configure SCLM and DB2 to work together.

Chapter 14. SCLM Support for Workstation Builds, describes how to set up and use SCLM to do builds on the workstation.

The Glossary of SCLM Terms and the Index sections are available for your reference.

Summary of Changes

z/OS V1R1.0 ISPF contains the following changes and enhancements:
- ISPF Product and Library Changes
- ISPF Dialog Manager Component Changes
- ISPF PDF Component Changes
- ISPF SCLM Component Changes
- ISPF Client/Server Component Changes

ISPF Product Changes

Changes to the ZENVIR variable. Characters 1 through 8 contain the product name and sequence number in the format *ISPF x.y*, where x.y indicates:
- <= 4.2 means the version.release of ISPF
- = 4.3 means ISPF for OS/390 release 2
- = 4.4 means ISPF 4.2.1 and ISPF for OS/390 release 3
- = 4.5 means ISPF for OS/390 Version 2 Release 5.0
- = 4.8 means ISPF for OS/390 Version 2 Release 8.0
- = 5.0 means ISPF for OS/390 Version 2 Release 10.0
- OR
- = 5.0 means ISPF for z/OS Version 1 Release 1.0

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

The ZOS390RL variable contains the ISPF release on your system.

The ZISPFOS system variable contains the level of ISPF code that is running as part of the operating system release on your system. This might or might not match ZOS390RL. For this release, the variable contains **ISPF for OS/390 Version 2 Release 10.0**.

New system variables:

ZBDMAX
BDISPMAX value

ZBDMXCNT
Count of current displays in batch mode session

ZPANELID
Name of currently displayed panel

ZSCREENI
Logical screen data

ZSCREENC
Cursor position within the logical screen data

The ISRDDN utility is now documented in the ISPF User's Guide.

ISPF DM Component Changes

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

- Additional support for panel process:
  - Support added for ″verify data set name with filter, (DSNAMEF)″.
  - Support added for ″verify data set name with filter with member, (DSNAMEFM)″.
  - Support added for ″verify data set name with quotes and parentheses, (DSNAMEPQ)″.
  - Support added for ″verify name with filter, (NAMEF)″.
  - Support added for ″verify specific constants within a variable, (PICTCN, string)″.
  - Support added for ″verify international format date, (IDATE)″.
  - Support added for ″verify standard date, (STDDATE)″.
  - Support added for ″verify Julian date, (JDATE)″.
  - Support added for ″verify Julian standard date, (JSTD)″.
  - Support added for ″verify international time, (ITIME)″.
  - Support added for ″verify standard time, (STDTIME)″.
  - Support added for NOJUMP attribute keyword.
  - Support added to allow INTENS(NON) on LI, LID, VOI and LEF attribute types.
  - Update)HELP section processing to support variables for keyword values and two new keywords MSG(message-name) and PASSTHRU.
- Support added for STKADD keyword on LIBDEF service.
- New QBASELIB service to query base libraries.
- Add Panel Id to CUAATTR utility.
- Add support for starting a new screen or application from the ISPF Task List panel.
- Add support for command CMDE which provides ability to expand command line if more room is required for the command.
- Add support to allow ISPF panel exits to be written in REXX.
- Add support for ZSCREENI and ZSCREENC variables to retrieve data from the logical screen at the cursor position.
- Add a field to the ISPF configuration table for the default language.
- Add fields to the ISPF configuration table to allow customization of the ISPF temporary data sets.
- Add a field to the ISPF configuration table for the default ISPF panel used when invoking ISPF.
- Pass the screen name to the SELECT Service Start and End and DISPLAY Installation exits.
- Update various ISPF messages with additional information. For example, a better message will be displayed when the user's profile is out of space, and the data set name and abend code will be added to the error message displayed as a result of an abend when opening a data set.

ISPDTLC enhancements:

ISPDTLC changes include new invocation options, new tags, and new tag. attributes as ISPF extensions to the Dialog Tag Language.

General improvements:

- A new option has been added to the interactive invocation panel, the DISPLAY(W) option check interval. This option controls the display frequency of a control panel for the DISPLAY and DISPLAYW options. The control panel choices are to continue, cancel the DISPLAY(W) option, or change the interval for the display of the control panel.
- New tags:
  - GENERATE
  - TEXTLINE
  - TEXTSEG
- Remove obsolete OS/2 DM compatibility and ISPF DTL extension messages for OS/390 V3.
- Add support for Tutorial selection panel ZSEL generation via ACTION tags.
- Revise member list processing to behave more like SUPERC by leaving the ″S″ code in the member selection field. Members can be deselected by removing the ″S″ before using PF3 to run the requested members.
- REQ70311 - Provide a user cancel/reset for the DISPLAY and DISPLAYW invoke options. A new panel - ISPCP08 - will display every nn (1 default) panels to allow the user to cancel or continue the display processing.
- Expand the interactive panel to 16 DTL source files.
- Expand the HELP attribute on tags for field level help to support the ISPF enhancement for MSG(message-ID) and PASSTHRU. HELP values can be: NO, YES, help-panel-name, *message-id, %varname, or *%varname. The ″*″ prefix defines a message-id.

New or changed tag attributes:

Tag name	**Attribute update**
ATTR	Add ATTN
CHECKI	Add support for ″VER(&variable, DSNAMEF)″ Add support for ″VER(&variable, DSNAMEFM)″ Add support for ″VER(&variable, DSNAMEPQ)″ Add support for ″VER(&variable, NAMEF)″ Add support for ″VER(&variable, PICTCN, ...)″ Add support for ″VER(&variable, IDATE)″ Add support for ″VER(&variable, STDDATE)″ Add support for ″VER(&variable, JDATE)″ Add support for ″VER(&variable, JSTD)″ Add support for ″VER(&variable, ITIME)″ Add support for ″VER(&variable, STDTIME)″
CHOFLD	Add ATTRCHAR and CAPS Support HELP for: YES, *message-id, *%varname
CHOICE	Add AUTOSEL Support HELP for: YES, *message-id, *%varname
CMDAREA	Add CAPS, NOJUMP, and SCRCAPS Support HELP for: YES, *message-id, *%varname Support SCRVHELP for: YES, *message-id, *%varname
DA	Add HELP and SCRCAPS Support SCRVHELP for: YES, *message-id, *%varname
DTACOL	Add VARCLASS, REQUIRED, and CAPS

Tag name	Attribute update
DTAFLD	Add ATTRCHAR, CAPS, and NOJUMP Support HELP for: YES, *message-id, *%varname Support DISPLAY=NO on CUA output fields
FIG	Add NOSKIP
GRPHDR	Add INDENT
LI	Add NOSKIP
LINES	Add NOSKIP
LP	Add NOSKIP
LSTCOL	Add CAPS and DISPLAY Support HELP for: YES, *message-id, *%varname
LSTFLD	Add SCRCAPS Support HELP for: YES, *message-id, *%varname
MSG	Add FORMAT Support HELP =*
MSGMBR	Add WIDTH
PANEL	Add ERRORCHECK
SELFLD	Support TYPE=TUTOR Support HELP for: YES, *message-id, *%varname
XMP	Add NOSKIP

ISPF PDF Component Changes

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

- An Edit settings dialog is now available via the EDSET and EDITSET primary commands as well as from the Edit_Setting pulldown choice when editing data. This enables the user to change:
  - the line that Edit positions the target of a FIND, CHANGE or EXCLUDE command.
  - whether or not the Editor always scrolls the target of a FIND, CHANGE, or EXCLUDE command to the target line specified.
  - the user session initial macro, a macro to be run whenever an edit session is started.
  - the maximum storage allowed for Edit.
  - Confirm Cancel/Move/Replace.
  - Preserve VB record length.
- The Edit COMPARE command will now compare your current Edit session against another data set without requiring a SAVE.
- The Edit COMPARE parameter SESSION or * will compare your current Edit data against the data saved on disk.
- The Edit COMPARE command can be issued while editing an uncataloged data set to compare members within the same data set.
- The new MEMLIST service provides an interface into ISPF option 3.1, providing all the built-in commands available from option 3.1.

- A new option in the ISPF Configuration Table dialog provides the automatic creation of a ++USERMOD for the configuration data.
- The new DSINFO service will return information about a specified data set in dialog variables.
- The Editor will no longer append a 1 character blank to variable length records that are 8 bytes in length.
- An ISPF Configuration option was added to disallow wildcards in the high level qualifier of option 3.4.
- The SuperC utility now supports an ALLMEMS option to enable compares of all members including alias entries without member selection.
- The primary and secondary quantity for the SuperC LIST and UPDATE data sets can be configured.
- Allow use of the SYSOUT field when doing a local print from option 3.6.
- Add an OPTION(DELETE) to the LMMDISP service to delete a member of the displayed list.
- Update the edit macro command DATASET to also return the data set from which the member being edited was found.
- Add a new dialog service called VIIF (View Interface service) which provides View function for the EDIF environment.
- Add an edit macro command LINE_STATUS which indicates whether a line of data has been changed during the edit session, and if so, how.
- Add additional keywords that can be specified in the expiration date field when creating a data set to indicate permanent retention: 9999, NEVER, NOLIMIT and PERM.
- Add a new option in the ISPF Configuration Table dialog to allow disabling all ENQ displays. This option indicates whether or not users should be able to see who has existing data set ENQs when they press the help key or when they use the ISRDDN utility.
- The LMINIT service specified with the DDNAME parameter will now handle DDNAMEs with up to 16 concatenated data sets. The DATAID generated by the LMINIT can then be passed to services such as EDIT and BROWSE to process members in any of the 16 data sets.

ISPF SCLM Component Changes

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

- Additional/modified SCLM Services:
  - An AUTHCODE service to update authorization codes has been added.
  - A NEXTGRP service to return the promote target for a given group.
  - The MIGRATE service will now allow the DATE/TIME of the member to be set by the caller.
  - The MIGRATE service will now be supported via the FLMLNK interface.
  - The MIGRATE service has a new report output and associated specification on the service call (default is to go to the terminal).
  - The FLMCMPLB macro has been deleted. Any projects using FLMCMPLB currently must be recoded to use: FLMSYSLB dsn,INCLS=COMPOOL.
- Additional exit points have been added:
  - At edit start and when the SPROF command is invoked.
  - When data is saved (Edit SAVE, Migrate, etc.).

  - After the NOTIFY step of a DELETE.
  - After the VERIFY step of a DELETE.
  - After the VERIFY step of a BUILD.
- The Versioning Utility will now allow a SuperC COMPARE of versions to be done.
- The Versioning Utility will capture output members, in addition to editable types.
- Workstation commands can now be used from translators running during a PROMOTE in batch mode.
- SCLM will now display dates in 4-character year format.
- The NRETRIEV command is now supported for SCLM.
- Added the ability to specify separate VERCOUNT values for each group/type combination.
- Additional samples:
  - A sample interface into ServiceDesk for z/OS to show how a change management system can be integrated into SCLM.
  - An Edit autoflagger to automatically flag changed lines.
  - A versioning delete sample.

ISPF Client/Server Component Changes

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

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

ISPF User Interface Considerations

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

- Use the CUAATR command to change the screen colors
- Use the ISPF Settings panel to specify that the TAB or HOME keys position the cursor to the command line rather than to the first action bar item
- Set the command line to the top of the screen by deselecting ***Command line at bottom*** on the ISPF Settings panel
- Set the primary keys to F13–24 by selecting 2 for Primary range on the Tailor Function Key Definition Display panel
- Use the KEYLIST OFF command to turn keylists off
- Use the PSCOLOR command to change point-and-shoot fields to blue.
- Change the DFLTCOLR field in the PDF configuration table ISRCONFG to disable action bars and or edit highlighting

ISPF Migration Considerations

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

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

Note: If you are migrating to z/OS V1R1.0 from OS/390 V2R10.0, there are no migration actions necessary. If you are migrating to z/OS V1R1.0 from a prior release of OS/390, follow the migration actions for OS/390 V2R10.0.

ISPF Profiles

Major changes were made to the ISPF profiles for ISPF Version 4.2 and OS/390 Version 1 Release 1.0 ISPF. The profiles for ISPF Version 3 and the profiles for OS/390 ISPF are not compatible. If you are moving back and forth between an ISPF Version 3 system and OS/390 V1R1.0 or higher system, you must run with separate profiles. Profiles for OS/390 V1R1.0 and higher are compatible with each other.

Year 2000 Support for ISPF

ISPF is fully capable of using dates for the year 2000 and beyond. All of your existing applications should continue to run (some may need minor changes, as explained below) when the year 2000 comes. The base support for the year 2000 was added to OS/390 Version 1 Release 2.0, but the same level of support is available for ISPF Version 3.5, ISPF Version 4, and OS/390 Version 1 Release 1.0 as well. To get support for the earlier versions, be sure that your system has the correct APARs installed. All ISPF APARs that add or correct function relating to the year 2000 contain the YR2000 identifier in the APAR text. You should search for these APARs to ensure you have all the function available.

What function is included?

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

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

- For those services that accept both 4-character year dates and 2-character year dates, you can specify one or the other. If you specify both, the 2-character year date is used to avoid affecting existing dialogs. When the 2-character year date is

used, the configuration table field mentioned above is used to determine whether the date should be interpreted as 19xx or 20xx.

- ISPF will not necessarily show 4-character dates in all circumstances but it will process them correctly. For example, a member list might only display 2-character year dates but will sort those dates in the proper order.
- SCLM stores dates past the year 1999 in a new internal format. If an accounting file contains dates in this new format, it cannot be processed by a system without year 2000 support. Accounting files without dates past 1999 can be processed with or without the year 2000 support.
- No conversion of the LMF control file is necessary.

Migrating from Previous Versions of SCLM

When migrating from one release of ISPF to another, you must be sure to reassemble and re-link all of your SCLM Project Definitions using the macros provided with the new release. If you have modified any of the SCLM-provided macros then you must re-integrate those changes with the new SCLM-provided macros. Failure to do this results in unpredictable results during SCLM execution.

Versioning Data Sets

In z/OS V1R1.0 ISPF, you can version fixed and variable outputs as well as editable data. If your project contains any record format U data, including load modules, then you will need to review the FLMATVER macros in your project definition. An asterisk (*) value for the TYPE (TYPE=*) on an FLMATVER macro with versioning enabled (VERSION=YES) will cause an error message to be issued when SCLM attempts to version the record format U data found in the project. Under those circumstances, FLMATVER macros should be added to specify each type to be versioned when the project contains record format U data. This change is not necessary when auditing only is enabled (VERSION=NO).

Additional versioning data sets must be allocated for any new types that you might now want to version.

Include Sets

In order to take advantage of the enhanced include search capabilities provided by SCLM 4.2 or later, changes must be made to the project definition. Additional function is available by updating your parsers to return include set information about the includes found by the parsers.

Use of parsers that return include sets other than the default or COMPOOL include set will result in an accounting record with a new format. Releases of SCLM before ISPF Version 4 Release 2 will generate error messages and may not be able to complete processing if they read an accounting record with this new format. To avoid problems with the use of previous releases of SCLM, it is recommended that only the default or COMPOOL include set be used until a project no longer uses releases of SCLM before 4.2.

Year 2000 Support

With the release of OS/390 Version 1 Release 3.0, SCLM began supporting dates beyond the year 2000. This has caused a change to the format of date fields stored in the SCLM VSAM databases. After you have used this release with a system date after December 31, 1999, you cannot go back to an earlier release of SCLM unless it also has support for dates beyond the year 2000.

The internal date format used by SCLM has also changed. The length and format of the **$acct_info** and **$list_info** date fields returned by SCLM services are different. These fields are now 8 characters in length and have the format **YYYYMMDD** (year, month, day). In addition, the 1–character alignment field in the **$acct_info** structure is now three characters long. Any user-written programs that use the SCLM service interface must be modified accordingly.

FLMALLOC Processing for IOTYPE S

After ISPF Version 4 Release 2, a change was made to SCLM FLMALLOC processing for IOTYPE S. When the following criteria are met, SCLM allocates the PDS member directly from the SCLM-controlled library, rather than copying it first to a sequential data set. The criteria are:

1. There is only one input.
2. The input is from a SINC statement.
3. The KEYREF on the FLMALLOC statement is SINC.
4. You are NOT doing input list processing.

Any user defined translators must take into account that the DDNAME allocated can be either a sequential data set or a PDS member.

Load Module Accounting Records and SSI Information

In ISPF Version 4.2 without APAR OW18306, when load modules without an SSI area (load modules that were linked without the SETSSI option) were migrated into SCLM, or when load modules were built using an architecture definition that did not include the LOAD keyword, the dates and times in the accounting records for the load modules were set to zeroes or random characters. Starting with OS/390 V1R3.0, or with ISPF Version 4.2 *with* APAR OW18306, it is not necessary to build a load module with the SETSSI option in order to migrate it into SCLM and still have correct accounting and SSI information.

The SCLM MIGRATE operation generates the data for the SSI area and updates the accounting record with the correct dates and times. Similarly, SCLM BUILD generates the SSI information and sets the correct dates and times in the accounting records for load modules that are generated without an LEC architecture definition. If you are migrating from a system with ISPF Version 4.2 without APAR OW18306 or earlier release, take these actions:

- If you have previously migrated load modules into SCLM that did not have the SSI information set, then you should migrate these modules into SCLM again. Remigrating these members ensures that the SSI information is set and that the accounting dates and times are correct.
- If you have previously generated load modules in SCLM without an LEC architecture definition (meaning that the accounting record date and time fields are zeroes or random characters) then these modules are rebuilt the first time a build is performed after installing z/OS V1R1.0 ISPF. This rebuild is necessary to ensure that the SSI and accounting record information for the load modules are in synch and have been updated with valid data. You might want to schedule the first build of your projects with the affected load modules at a time that minimizes the impact to your system.

What's in the z/OS V1R1.0 ISPF library?

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

z/OS V1R1.0 ISPF

Title	Order Number
z/OS V1R1.0 ISPF Dialog Tag Language Guide and Reference	SC34-4824-00
z/OS V1R1.0 ISPF Planning and Customizing	GC34-4814-00
z/OS V1R1.0 ISPF User's Guide Volume I	SC34-4822-00
z/OS V1R1.0 ISPF User's Guide Volume II	SC34-4823-00
z/OS V1R1.0 ISPF Services Guide	SC34-4819-00
z/OS V1R1.0 ISPF Dialog Developer's Guide and Reference	SC34-4821-00
z/OS V1R1.0 ISPF Reference Summary	SC34-4816-00
z/OS V1R1.0 ISPF Edit and Edit Macros	SC34-4820-00
z/OS V1R1.0 ISPF Library Management Facility	SC34-4825-00
z/OS V1R1.0 ISPF Messages and Codes	SC34-4815-04
z/OS V1R1.0 ISPF Software Configuration and Library Manager Project Manager's and Developer's Guide	SC34-4817–00
z/OS V1R1.0 ISPF Software Configuration and Library Manager Reference	SC34-4818-00
Entire library Bill of Forms	SBOF-8570

Elements and Features in z/OS

You can use the following table to see the relationship of a product you are familiar with and how it is referred to in z/OS Version 1 Release 1.0. z/OS V1R1.0 is made up of elements and features that contain function at or beyond the release level of the products listed in the following table. The table gives the name and level of each product on which a z/OS element or feature is based, identifies the z/OS name of the element or feature, and indicates whether it is part of the base or optional. For more compatibility information about z/OS elements see *z/OS Planning for Installation, GC28-1726*

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

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

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

The ISPF User Interface

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

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

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

Some Terms You Should Know

The following terms are used in this book:

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

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

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

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

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

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

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

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

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

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

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

How to Navigate in ISPF without Using Action Bars

If you use a non-programmable terminal to access z/OS V1R1.0 ISPF and you do not want to take advantage of the command nesting function, you can make selections the same way you always have: by typing in a selection number and pressing Enter.

How to Navigate in ISPF Using the Action Bar Interface

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

Action Bars

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

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

  If mnemonics are defined for action bar choices, you can:

  – In 3270 mode, on the command line, type ACTIONS and the mnemonic letter that corresponds to an underscored letter in the action bar choice text. This results in the display of the pull-down menu for that action bar choice.

  – In 3270 mode, on the command line enter the mnemonic letter that corresponds to an underscored letter in the action bar choice text, and press the function key assigned to the ACTIONS command. This results in the display of the pull-down menu for that action bar choice.

  – In GUI mode, you can use a *hot key* to access a choice on an action bar or on a pull-down menu; that is, you can press the ALT key in combination with the mnemonic letter that is underscored in the choice text to activate the text.

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

Notes:

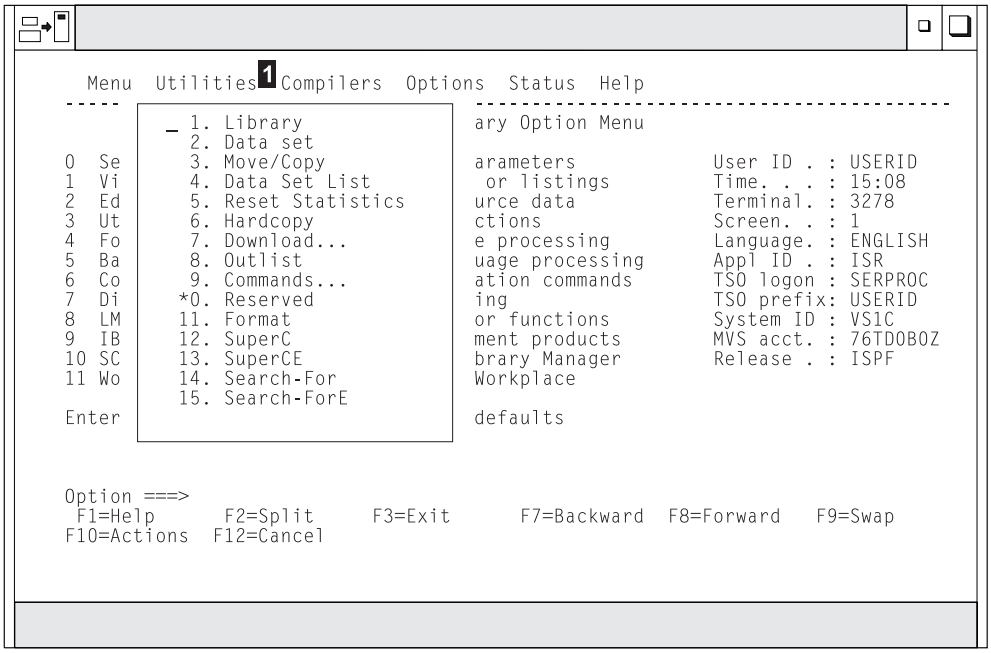
1. ISPF does not provide a mouse emulator program. This book uses *select* in conjunction with point-and-shoot text fields and action bar choices to mean moving the cursor to a field and simulating Enter.

   Note: Some users program their mouse emulators as follows:

   - Mouse button 1 – to position the cursor to the pointer and simulate Enter
   - Mouse button 2 – to simulate F12 (Cancel).

2. If you want the Home key to position the cursor at the first input field on an ISPF panel, type `SETTINGS` on any command line and press Enter to display the ISPF Settings panel. Deselect the **Tab to action bar choices** option.
3. If you are running in GUI mode, the Home key takes you to the beginning of the current field.

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



1 The selected action bar choice is highlighted.

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

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

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

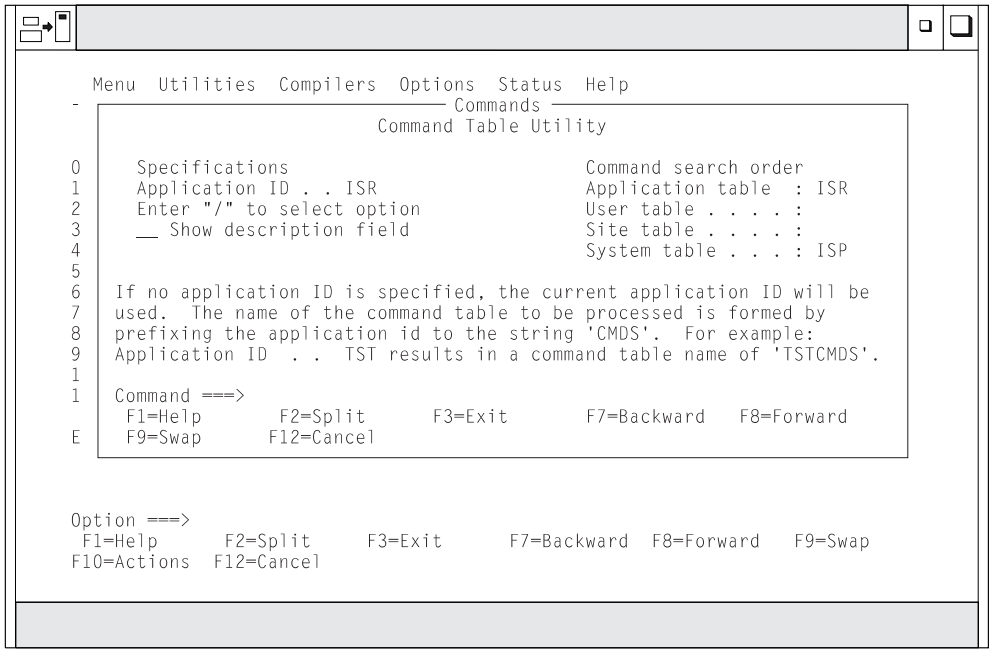
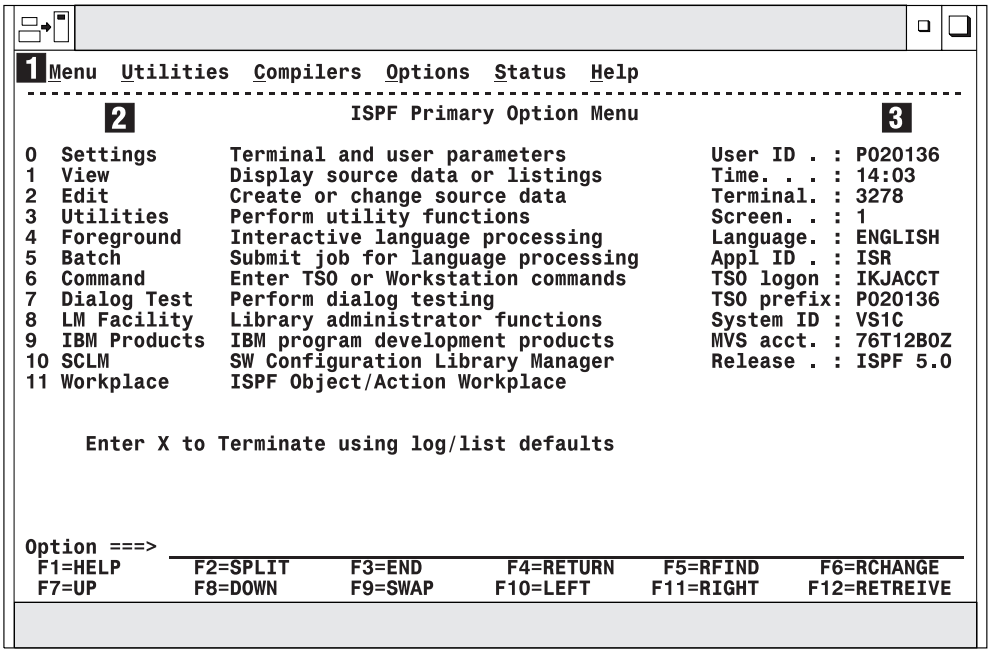


Figure 2. Pop-Up Selected from an Action Bar Pull-Down



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

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

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

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

Action Bar Choices

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

Menu Action Bar Choice

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

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

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

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

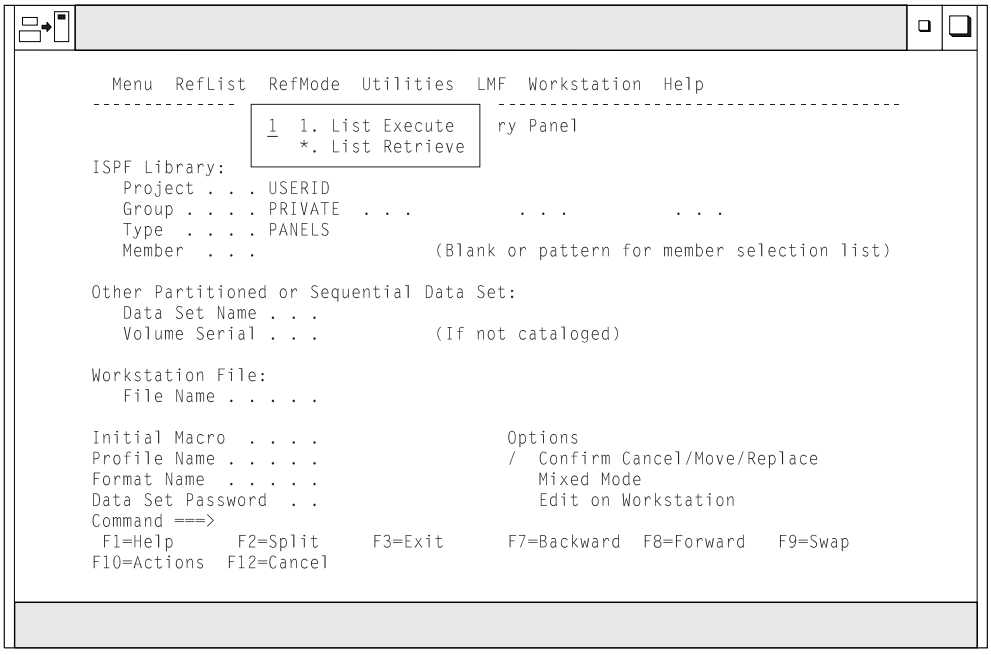


Figure 4. An Unavailable Choice on a Pull-Down

Utilities Action Bar Choice

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

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

Point-and-Shoot Text Fields

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

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

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

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

Function Keys

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

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

The following function keys help you navigate in ISPF:

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

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

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

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

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

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

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

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

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

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

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

Selection Fields

z/OS V1R1.0 ISPF uses the following CUA-compliant conventions for selection fields:

A single period (.)

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

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

You can select only one member to edit.

A single underscore (_)

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

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

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

An underscored field (____)

Member lists or text fields that use underscores in the selection field

recognize multiple selections. For example, from the Display Data Set List Option panel, you may select multiple members for print, rename, delete, edit, browse, or view processing.

Command Nesting

Command nesting allows you to *suspend* an activity while you perform a new one rather than having to end a function to perform another function. For example, in previous versions of ISPF, if you are editing a data set and want to allocate another data set, you type `=3.2` on the command line and press Enter. ISPF *ends* your edit session before taking you to the Data Set Utility panel. When you have allocated the data set and want to return to your edit session, you type `=2` and press Enter; ISPF returns you to the Edit Entry Panel. With z/OS V1R1.0 ISPF, from your edit session, select the Data set choice from the Utilities pull-down on the Edit panel action bar. ISPF suspends your edit session and displays the Data Set Utility panel. When you have allocated the new data set and end the function, z/OS V1R1.0 ISPF returns you directly to your edit session rather than to the Edit Entry Panel.

Part 1. Project Manager's Guide

Chapter 1. Defining the Project Environment

This chapter describes the tasks performed by project managers to set up and maintain an SCLM project environment. The required steps are described in complete detail, with examples and recommended procedures where applicable. After you understand the steps discussed in the first part of this chapter, you can experiment with installing an actual project by completing the steps outlined in "Project Manager Scenario" on page 41. The data sets used in the scenario are available as part of the ISPF product. You can use ISPF Option 10.7 to create a small sample project.

If SCLM does not appear on any of your menu panels or on your *Menu* pull-down, you can still access it by typing **TSO SCLM** on any ISPF command line, then pressing Enter. If SCLM is available to your terminal session, the SCLM Main Menu is displayed. If SCLM is not available on your system, a panel (ISRNOSLM) is displayed to inform you that SCLM is not available to your terminal session.

Overview of Project Manager Tasks

The primary function of the project manager is to create and manage the project environment. The SCLM project environment consists of three types of information associated with an individual project:

- User Application Data (see "User Application Data" on page 141)
- Project Definition Data (see "Project Definition Data")
- SCLM Control Data (see "Step 6: Allocate and Create the Control Data Sets" on page 18).

Project Definition Data

The project manager uses the SCLM project definition to generate and maintain the project environment. A project definition defines the desired development environment to SCLM for an individual project. Using the project definition, the product manager can define:

- The structure of the project hierarchy using groups and types
- The languages to use, such as COBOL and Pascal
- The rules to move data within the hierarchy (authorization codes)
- The SCLM options, such as audit and versioning

More than one project definition can be generated for a single project. The main project definition for an SCLM project is the *primary* project definition. All other project definitions for the same project are *alternate* project definitions. Alternate project definitions are usually used for performing specific tasks that cannot or should not be done with the primary project definitions. The use of alternate project definitions should be kept to a minimum, if any are required.

Generating a Project Environment

To create the project environment, the project manager should be familiar with VSAM data sets and MVS high-level qualifiers. It is also helpful if the project manager understands Job Control Language (JCL).

The project manager should determine which compatible (such as DATABASE 2) programs, if any, are to be used with SCLM, then use the following steps to generate a project environment:

STEP	See page	
	Standard SCLM	**With DB2**
1.Determine the project's hierarchy.	4	276
2.Identify the types of data to be supported.	8	276
3.Establish authorization codes.	8	276
4.Allocate the PROJDEFS data sets.	12	276
5.Allocate the project partitioned data sets (PDS).	13	277
6.Allocate and create the control data sets.	18	277
7.Protect the project environment.	24	277
8.Create the project definition.	25	277
9.Assemble and link the project definition.	40	277

Step 1: Determine the Project's Hierarchy

As a project manager, you are responsible for generating and updating the hierarchy of the project to accommodate project requirements. This step helps you plan the project hierarchy. When you have completed this step, you should have a diagram of the hierarchy with all the groups labeled, as well as an understanding of how each group is used.

It is usually easier at first to draw a diagram of your hierarchy. This lets you visualize what the hierarchy looks like. The following rules govern the creation of hierarchies:

- Each group can have no more than one parent.
- Each group can have multiple groups promoting into it.
- There is no restriction on the total number of groups a hierarchy can have.
- A hierarchical view can contain no more than 123 groups. This is because MVS has a limit of 123 extents for a concatenated partitioned data set.
- Each hierarchy has one root group, the topmost group.
- It is possible to have more than one hierarchy defined for one project.
- Defining no more than four layers makes it easier to use ISPF tools on the SCLM-controlled members.

The following two figures show two examples of hierarchies. These hierarchies are set up based on the development phases potential projects might use. You can create hierarchies other than those presented here. As a project evolves, the requirements that the project has on the hierarchy will change. With SCLM, you can change the hierarchy to meet the needs of the project.

The reasoning behind the hierarchy shown in Figure 5 on page 5 follows:

- The development groups (USER1, USER2, and USER3) are where all modifications to SCLM-controlled members are made.
- The INT group is for integrating (combining) all the SCLM-controlled members from the development groups.
- The TEST group is the group where system or function testing of the application will take place.

- The RELEASE group will contain the final version of the application being developed. It is from this group that the application could be put into production.

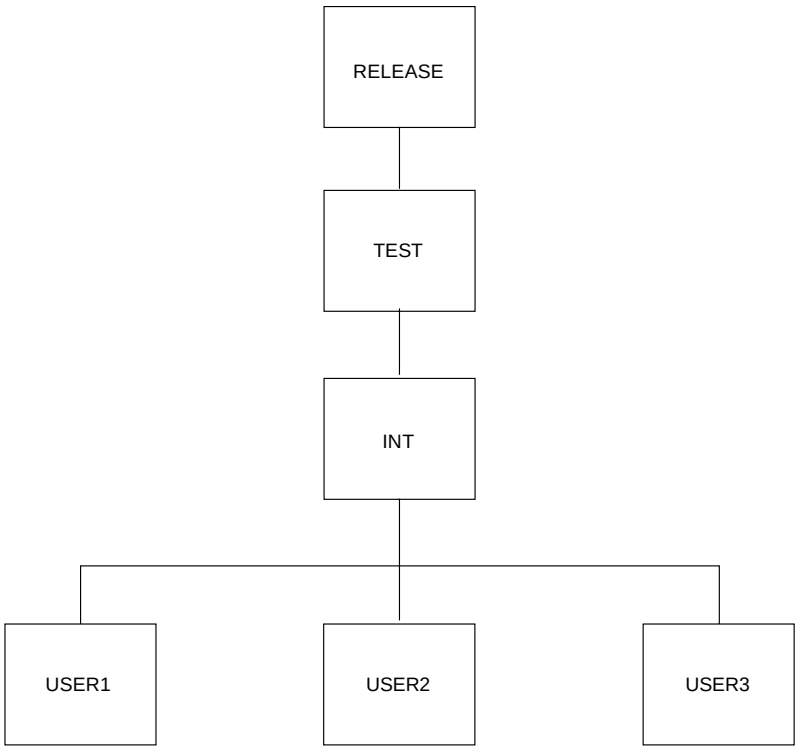


Figure 5. Example of SCLM Hierarchies

The second hierarchy, shown in Figure 6, is different. This hierarchy has two separate legs. Each leg of the hierarchy contains a separate subsystem of the application being developed. The stage groups (STAGE1 and STAGE2) in each hierarchy leg are used for integrating and unit testing the subsystems within each hierarchy leg. The SYSTEST group is used to combine the subsystems from both legs of the hierarchy for delivery to a system test organization.

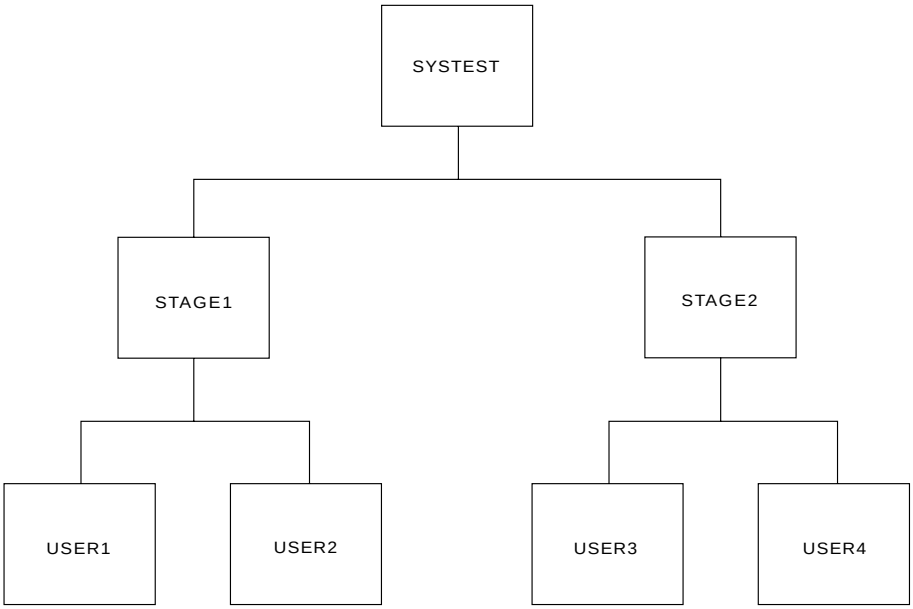


Figure 6. Example of SCLM Hierarchies

Use the preceding rules and the requirements of your project to draw your hierarchy and label each group.

Primary Non-Key Group Testing Techniques

You can use primary non-key groups as a technique to allow integration and testing of a software application. The technique is useful where integration work can have far-reaching and undesirable effects, for example, when a global change to an application affects the majority of developers. The technique is also useful when schedule or other pressures are such that you must perform high-risk integration of software. SCLM does not allow you to promote from a primary non-key group.

In a normal SCLM scenario, you promote code from individual development libraries to a common integration group before performing integration testing. However, you can generate an alternate project definition that deviates from the default project definition. The alternate project definition defines an intermediate non-key group for integrating subsets of development groups. Define the non-key group so that only key groups promote into the non-key group. Developers authorized to this intermediate group can then promote code to it for unit and function testing. Testing takes place in this group before promotion to the normal integration group. Because being at a non-key group does not cause members to be purged from a key group during a promote, no members are removed from the default project definition. In this way, you avoid potential integrity problems.

Using this technique, the activities of small groups of integrators do not affect the normal hierarchy until their testing is complete. By switching to the alternate project definition, developers can easily test their integration by promoting to the primary non-key group. When promoting to a non-key group, code still exists in the normal hierarchy in the development libraries. SCLM promotion from the development libraries, using the default project definition, would then incorporate the code into the normal integration group. New code can go through an accurate configuration test before being applied to the normal hierarchy. Code developed using this scenario is potentially more complete and accurate than code developed in a normal scenario.

Use Figure 7 and Figure 8 to compare a default hierarchy structure with an alternate hierarchy structure. Figure 7 shows a default hierarchy structure for a project. You can perform all normal development activities within the default hierarchy structure.

Figure 8 shows an alternate hierarchy structure with a primary non-key integration group for the project shown in Figure 7.

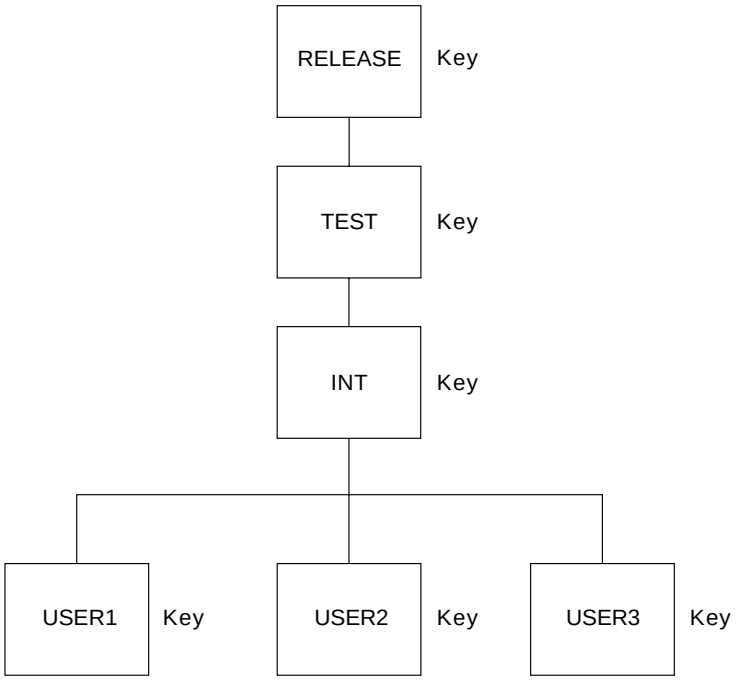


Figure 7. Default (Primary) Project Hierarchy Structure

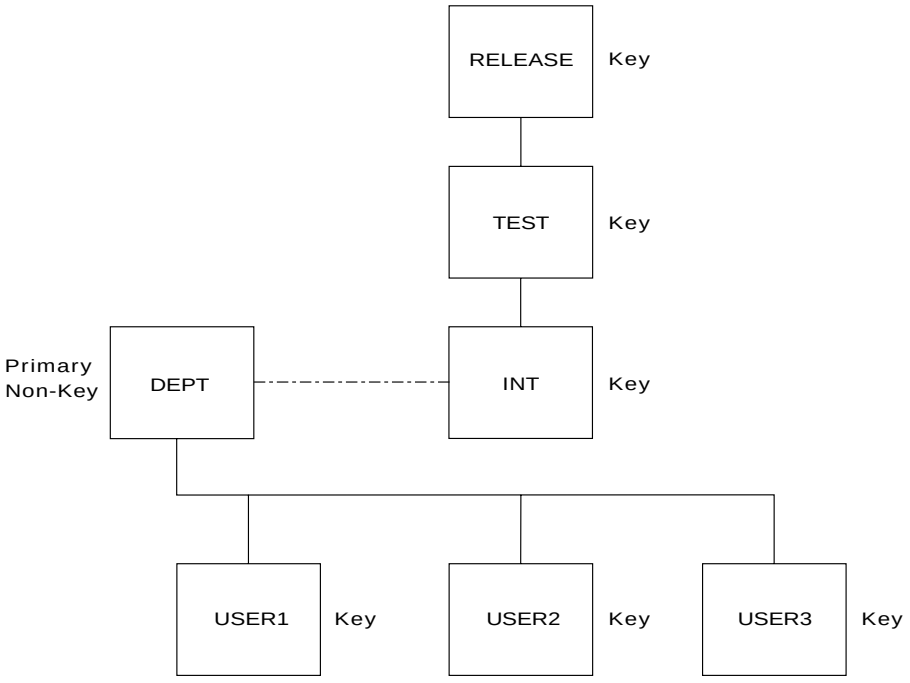


Figure 8. Alternate Project Hierarchy Structure with Primary Non-key Integration Group

In the example, the developers (USER1, USER2, USER3) can use the alternate project definition to promote code into the primary non-key group. You cannot promote up from the primary non-key group, but you can draw down from it.

Promotion to a non-key primary group does not cause deletion of the components from the respective development libraries. Building in the primary non-key group allows the developers to integrate and test pieces of code still under development. Code that is then complete can be promoted through the default project definition from the development libraries into the normal integration group. The promotion to the normal integration libraries causes the components to be deleted from the respective development libraries, but not from the primary non-key group. Deletion from the primary non-key group must be done manually using the SCLM Library Utility, the Delete Group Utility or through SCLM services, such as Delete Group.

Step 2: Identify the Types of Data to Support

This step identifies the types of data required by the applications under development for your project. Some examples of the types of data used are source code, object modules, load modules, and source listings. The list of types developed in this step is used in later steps.

SCLM supports the same kind of data supported by MVS partitioned data sets. The amount of data is also a factor in determining the types of data needed. Different types (such as objects and listings) of data should not reside in the same SCLM type. Determine the number of types you need based on the data you want to maintain for the project. For example, if you want to maintain compiler listings, a listing type is necessary. At a minimum, use four types to produce executable code:

- Source type for application source code
- Object type for generated object code
- Load type for generated load modules
- Architecture type for architecture definition members.

Similar kinds of data can reside in separate types. For example, you can divide source code into assembler source code and Pascal source code. To do this, identify an assembler type and a Pascal type.

Step 3: Establish Authorization Codes

Authorization codes control the movement of data within the hierarchy. The purpose of this step is to assign authorization codes to the hierarchy. Authorization codes restrict the draw down and promotion of members to certain groups within the hierarchy.

At least one authorization code must be defined for a project. If no authorization codes are defined, SCLM will not permit members to be drawn down or promoted. Authorization codes work only on editable types such as source, not on build outputs. Authorization codes are assigned to each group in the hierarchy. Groups can have any number of authorization codes assigned to them. Members are assigned authorization codes when they are registered with SCLM. Members can only exist in groups that have been assigned the same authorization codes as the members.

It is not necessary to define more than one authorization code for the entire project. A single authorization code allows each member under SCLM control to be drawn down to any development group and be promoted to the top of the

hierarchy. If tighter restrictions on the movement of your data are required for your project, you must identify those situations and define additional authorization codes.

An example of when multiple authorization codes can be used is when an application has multiple subsystems being developed in different legs of the hierarchy and you need to ensure that the members of the two subsystems do not get mixed in the development groups in the hierarchy legs. Authorization codes can be set up to prevent the members from one subsystem from being drawn down into the development groups of the other subsystem. This requires at most two authorization codes. For additional possible uses of authorization codes, see "Using Authorization Codes to Control SCLM Operations".

Using the diagram that you drew for Step 1, examine the flow of members and determine if any restrictions on the movement of members are required. Label each group with at least one authorization code. Authorization codes can be up to 8 characters and cannot contain commas.

Using Authorization Codes to Control SCLM Operations

Authorization codes restrict promotions and draw downs on a member-by-member basis for source code only. This section discusses some uses of authorization codes.

First, some facts about authorization codes:

- An authorization code is a character string up to 8 characters and cannot contain commas.
- When you create the project definition, you assign zero or more authorization codes to each group.
- Each member of every group within an SCLM-controlled project is assigned one authorization code.
- In order to put a member into a group, the authorization code of that member must match one of the authorization codes that have been assigned to the group.
- When all the authorization codes are removed from a group, no members can be promoted into or out of that group.
- When you promote a member from one group to the next, the member retains its authorization code. Thus, the group being promoted into and the group being promoted from must have a matching authorization code. If, as a result of a promote, an older version of the module was replaced, the authorization code assigned to that older version is not kept.

Figure 9 on page 10 shows a simple hierarchy with four groups: RELEASE, TEST, DEV1 and DEV2. The group RELEASE has been assigned only one authorization code: DEV. Group TEST has two authorization codes: DEV and TESTONLY. Three authorization codes (DEV, PROTO, and TESTONLY) have been assigned to DEV1. Group DEV2 has DEV and L0 as its authorization codes.

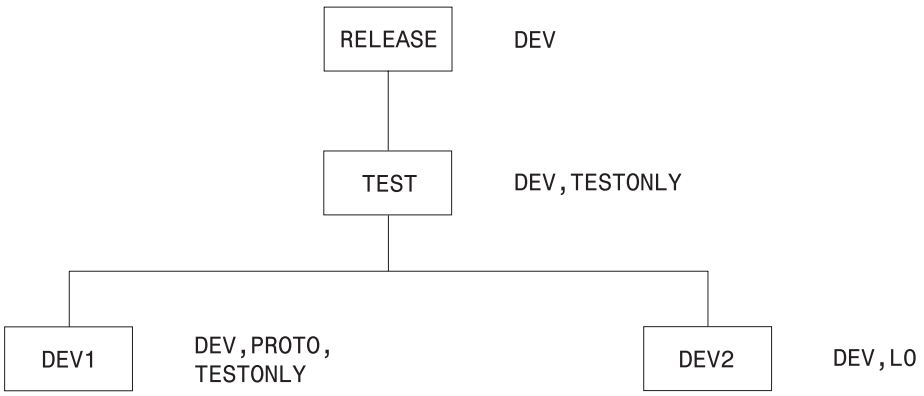


Figure 9. Sample Hierarchy with Authorization Codes

Code this information in the project definition as follows:

```
RELEASE  FLMGROUP   KEY=Y,AC=(DEV)
TEST     FLMGROUP   KEY=Y,AC=(DEV,TESTONLY),PROMOTE=RELEASE
DEV1     FLMGROUP   KEY=Y,AC=(DEV,TESTONLY,PROTO),PROMOTE=TEST
DEV2     FLMGROUP   KEY=Y,AC=(DEV,L0),PROMOTE=TEST
```

In Figure 9, the following relationships exist:

- A member in DEV1 with an authorization code of PROTO cannot be promoted because group TEST does not have PROTO as an authorization code.
- For the same reason, a member in DEV1 with an authorization code of TESTONLY can be promoted to TEST, but cannot be promoted to RELEASE.
- Similarly, a member in DEV1 or DEV2 with an authorization code of DEV can be promoted all the way up to group RELEASE.
- A member in DEV2 cannot have an authorization code of TESTONLY or PROTO; it must be either DEV or L0.
- A member in DEV2 with an authorization code of L0 cannot be promoted because group TEST does not have L0 as an authorization code.

When you edit a member in a development group, SCLM looks at the authorization code you specified on the edit panel and tells you the following:

- If that authorization code is not valid for that development group, you must enter an authorization code that is assigned to that group. If you enter an invalid authorization code and then press the help key, SCLM shows authorization codes for that group.
- If use of that authorization code prevents promotion of that member at some point in the group hierarchy, SCLM gives you the name of the group into which promotion is not allowed.
- If use of that authorization code leads to a potential promotion conflict with another member of the same name, SCLM does not allow the edit. An example of this problem follows.

  SCLM allows you to have two members of the same name and type residing in two different development groups (such as DEV1 and DEV2 in Figure 9) under certain conditions. Each of those members has an authorization code assigned to it. Those codes, along with the authorization codes assigned to the higher groups in the hierarchy, determine how far up the hierarchy each of those members can be promoted. If the two promotion paths do not intersect, SCLM lets you edit those members in those groups. However, if there is at least one group through which both members can be promoted, changes made to one

member would be lost when the other member is promoted. In that case, SCLM does not let you edit the members in those groups.

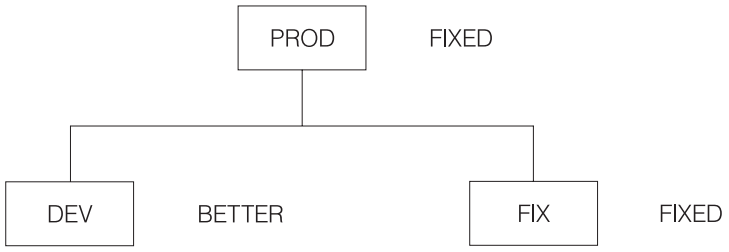
If a member exists in group DEV1, SCLM uses authorization codes to determine whether or not you can edit a member with the same name and type in group DEV2:

Table 1. Authorization Code Allowances

Auth. Code for member in DEV1	Auth. Code for member in DEV2	Allowed?	Why?
DEV	DEV	No	Both members can be promoted through TEST.
DEV	L0	Yes	Promotion paths do not intersect.
PROTO	TESTONLY	No	TESTONLY is not a valid authorization code for DEV2.
PROTO	L0	Yes	Promotion paths do not intersect.
TESTONLY	DEV	No	Both members can be promoted through TEST.
TESTONLY	L0	Yes	Promotion paths do not intersect.

Allowing Parallel Updates

You can use the information in the previous section to set up a project in which you can make modifications to what you have in production (development) while being able to make quick fixes to production modules (maintenance). The simple hierarchy is illustrated in the following example. An actual hierarchy can contain many groups and layers.



Define the groups as follows:

```
PROD     FLMGROUP   KEY=Y,AC=(FIXED)
DEV      FLMGROUP   KEY=Y,AC=(BETTER),PROMOTE=PROD
FIX      FLMGROUP   KEY=Y,AC=(FIXED),PROMOTE=PROD
```

There are three groups: PROD is the production library, DEV is the development library, and FIX is the maintenance library. In practice, there would be a much larger subhierarchy under both DEV and FIX in order to allow for both multiple developers and for testing of applications before moving them to production.

DEV, FIX, and PROD each have a single authorization code, BETTER, FIXED, and FIXED respectively, and could have more. More importantly, no authorization code is assigned to both DEV and PROD. It is this aspect of the project definition that prevents the promotion of any modules from group DEV into group PROD. When the development code is ready to move into production, the authorization code BETTER must be added to the valid authorization codes for the PROD group.

A programmer planning to make changes to a module for the next release of an application draws the module down from PROD into DEV, specifying an authorization code of BETTER on the SCLM EDIT-ENTRY PANEL. Changes are made and tested in DEV.

Suppose that while the module is being changed and tested in the DEV group, a user encounters a problem with the application and another programmer determines that the fix requires a change to the module that has been drawn down to DEV.

The programmer can draw down the module into FIX even though that same module has been drawn down into DEV. This is possible because the promotion paths of the two modules do not intersect: the module in DEV cannot be promoted into PROD because of authorization codes. Therefore, changes made to one module do not overwrite changes made to the other copy.

When the fix has been made to the module in FIX and the application has been rebuilt at that group, the user can run the application from group FIX until the fix has been verified and then promoted to PROD.

Before the fix is promoted, the changes must be incorporated into the copy of the modules in DEV. This is a manual change made by the current owner of the modules in DEV with the assistance of the person who made the changes in FIX.

Keep in mind that although authorization codes can be used to restrict promotion paths, they do not provide security against modifications to SCLM-controlled data made outside of the SCLM environment. You should use RACF* (or the functional equivalent) for that purpose.

Step 4: Allocate the PROJDEFS Data Sets

The PROJDEFS data sets are used to store the project definition data for an individual project. The purpose of this step is to allocate the PROJDEFS data sets.

The PROJDEFS data sets are partitioned data sets with the following naming convention:

```
project_id.PROJDEFS.*
```

Because the project definition is written using S/370 assembler language, SCLM requires that the load data set be named:

```
project_id.PROJDEFS.LOAD
```

When a user invokes SCLM for a specific project, SCLM uses the current assembled version of the project definition located in the LOAD data set.

The data sets containing the project definition's source and object code are not required by SCLM to follow the PROJDEFS naming convention, but it is recommended to make maintaining the project definition easier. Therefore, following the naming convention would produce the following data sets:

```
project_id.PROJDEFS.SOURCE
project_id.PROJDEFS.OBJ
```

Allocate the PROJDEFS data sets using the attributes defined in Table 3 on page 18. The PROJDEFS data sets should be protected from access by general users. Protecting the PROJDEFS data sets is discussed in "Step 7: Protect the Project Environment" on page 24.

Step 5: Allocate the Project Partitioned Data Sets

The project partitioned data sets are used to store the user application data. These data sets are organized into a hierarchy and controlled by the project definition. Allocate the project partitioned data sets using either the ISPF Data Set Utility (option 3.2) or a JCL process. Use the information in this step to determine the names, number, and physical characteristics of the project partitioned data sets.

Data Set Naming Conventions

SCLM expects all the project partitioned data sets to use the default naming convention of `project.group.type`. Because some projects cannot use the default naming convention, SCLM allows the project manager to specify an alternate naming convention either for all the project partitioned data sets or for the project partitioned data sets associated with individual groups in the hierarchy.

If your data already exists, the existing data sets can be used in conjunction with SCLM's flexible data set naming capability. The next section provides additional information on using this capability.

Flexible Naming of Project Partitioned Data Sets

With SCLM, product managers can use the SCLM-supplied default data set naming convention or a user-defined naming convention. The default naming convention is PROJECT.GROUP.TYPE. If the SCLM default naming convention is not used, the project manager's convention must use the MVS naming conventions. For example, it is possible to use four or five qualifiers in the data set names instead of the three qualifiers that are used by the SCLM naming convention. (The PROJDEFS data sets are exceptions; these data sets must use the naming convention defined in "Step 4: Allocate the PROJDEFS Data Sets" on page 12.)

To define a naming convention other than SCLM's default naming convention, you must specify data set names that correspond to specific groups or the entire project. While the names of the data sets used by SCLM can use more than three qualifiers, the developers still see the PROJECT.GROUP.TYPE naming convention on the SCLM dialog panels and service calls. The project definition creates a mapping between the PROJECT.GROUP.TYPE name and the user-defined data set names associated with each group in the hierarchy.

Note: This mapping is only maintained while users are executing SCLM functions. If ISPF utilities are used on data controlled by SCLM, the users should know the mapping between the PROJECT.GROUP.TYPE name and the fully-qualified data set name.

The data set names are defined in the project definition with the FLMCNTRL and FLMALTC macros. Each macro has a DSNAME parameter that allows the project manager to specify the data set names for the entire project or for individual groups. The FLMCNTRL macro defines the data set names for the entire project; the FLMALTC macro defines the data set names on a group-by-group basis. See the *ISPF Software Configuration and Library Manager (SCLM) Reference* for an example of how to set up the macros to use flexible naming of partitioned data sets.

The DSNAME parameters on both macros work the same way and can be used within the same project definition. The value specified on the DSNAME parameter is a pattern for the data set name. This pattern must meet MVS naming conventions and can contain the SCLM variables @@FLMPRJ, @@FLMGRP, and @@FLMTYP. If DSNAME is not specified, SCLM uses the default naming

convention of PROJECT.GROUP.TYPE. The use of variable @@FLMTYP is required. SCLM verifies that the variable @@FLMTYP is used on each DSNAME parameter when the project definition is loaded into memory. The variable @@FLMGRP is **very strongly** recommended. The use of these variables minimizes the risk that data set names associated with different groups are the same and prevents data from being overwritten. The variable @@FLMPRJ is optional.

The SCLM variable @@FLMDSN is created from the value of the DSNAME parameter. Therefore, if the data set name pattern is @@FLMPRJ.component_name.@@FLMGRP.@@FLMTYP, the value of @@FLMDSN will be @@FLMPRJ.component_name.@@FLMGRP.@@FLMTYP.

The versioning partitioned data sets can also use a naming convention other than SCLM's default naming convention. The VERPDS parameter on the FLMCNTRL and FLMALTC macros is used to specify the name of the versioning partitioned data sets. SCLM uses a default of @@FLMDSN.VERSION for the names of the versioning data sets. If a pattern other than the default is used, the variables @@FLMGRP and @@FLMTYP must be part of the data set name pattern. Using two variables minimizes the risk that the versioning data set names associated with different groups are the same, and prevents data from being overwritten.

Attention:

SCLM does not guarantee the uniqueness of the data set names or check the validity of values entered on the DSNAME parameter.

Number of Data Sets to Allocate

Normally, a data set should be allocated for every possible PROJECT.GROUP.TYPE combination in the hierarchy. However, if the intent is to develop code in several hierarchies that merge in one main hierarchy, there might be no need to allocate some data sets. Allocating only the data sets that are actually used saves time when creating the hierarchy and minimizes DASD use and catalog entries. See Figure 10 on page 16 for an example of a hierarchy that does not have all data sets allocated.

Only those data sets actually used in the hierarchy must be physically allocated. SCLM functions will execute successfully for hierarchies that contain unallocated data sets, as long as the unallocated data sets are not used. If a data set is not allocated and SCLM attempts to use the data set, an error message is issued.

Data sets can be added at any time. If you leave a data set unallocated and later find you need it, simply allocate the data set then.

Determining When Data Set Allocation Is Necessary: You can leave the data sets for the intermediate groups in your project unallocated until the first time they are needed for a promote. You can also leave the data sets for types that will not be used at a particular group unallocated. As an example, if a developer is responsible for source code but not panels, then you can leave the data set for the type containing panels unallocated for his group.

A data set need not be allocated if an EXTEND type is being used and the hierarchy is designed so that the source code for the EXTEND type is always at a higher group.

For example, consider a project definition with the FLMTYPE macro written as follows:

```
CMNSRC    FLMTYPE
BLDSRC    FLMTYPE  EXTEND=CMNSRC
```

In this situation, the type CMNSRC can contain members referenced by members in the BLDSRC type. However, if the source code in CMNSRC will always be at a higher layer in the hierarchy (for example, IVV), you do not need to allocate data sets for type CMNSRC below the IVV layer in the main hierarchy.

How SCLM Functions Use Data Sets: SCLM uses a data set when it expects that the data set already contains a member (for example, when attempting to delete a member), or when the data set will contain a member (for example, when saving a new member). The following list details how SCLM functions use a data set:

Build	Uses a data set if it contains a member that has a corresponding accounting record and that member is being built or referenced by another member that is being built. Build also uses data sets for output (those referenced by the LOAD, OBJ, or LIST architecture keywords, for example).
Promote	Uses a data set if it contains a member that has a corresponding accounting record and that member is being promoted. If these data sets contain members that need to be promoted, they must be present in the current group and in the group being promoted to; otherwise, an error message is issued. If a promotion occurs from a non-key group to a key group, the corresponding data sets at the previous key group will also be used.
Delete	Uses a data set when deleting a member.
Delete Group	Uses a data set when deleting a member.
Library Management Utility	Uses a data set when deleting a member or when Edit, View or Build are invoked.
Import	Uses a data set when VSAM records are being imported into the hierarchy. The member imported must exist somewhere in the hierarchy view for the group being imported into.
Edit	Uses a data set when storing or retrieving a member.
View	Uses a data set when retrieving a member.
Migrate	Uses a data set to retrieve information about a member that is being migrated into the SCLM hierarchy.
Parse	Uses a data set when parsing a member.

Manipulating VSAM Records for Unallocated Data Sets: A build map can be created for a member that is higher in the hierarchy but for which there is no source data set allocated for the group where the build is occurring. If you delete a data set, the corresponding accounting records and build maps can still exist in the VSAM databases.

Using the following utilities and services, you can browse or delete VSAM records that correspond to an unallocated data set.

Library Management Utility	Browse and delete accounting records and build maps that correspond to an unallocated data set.
Delete	Delete accounting records and build maps that correspond to an unallocated data set.
Delete Group	Delete accounting records and build maps that correspond to an unallocated data set.

Examples of Hierarchies with Unallocated Data Sets: A valid hierarchy that contains unallocated data sets is shown in Figure 10. Member B INCLUDES member C. A build of member B from group USR1 will succeed, although a data set was not allocated for Cmnsrc at the INT group. The build will locate and use member C from the IVV group.

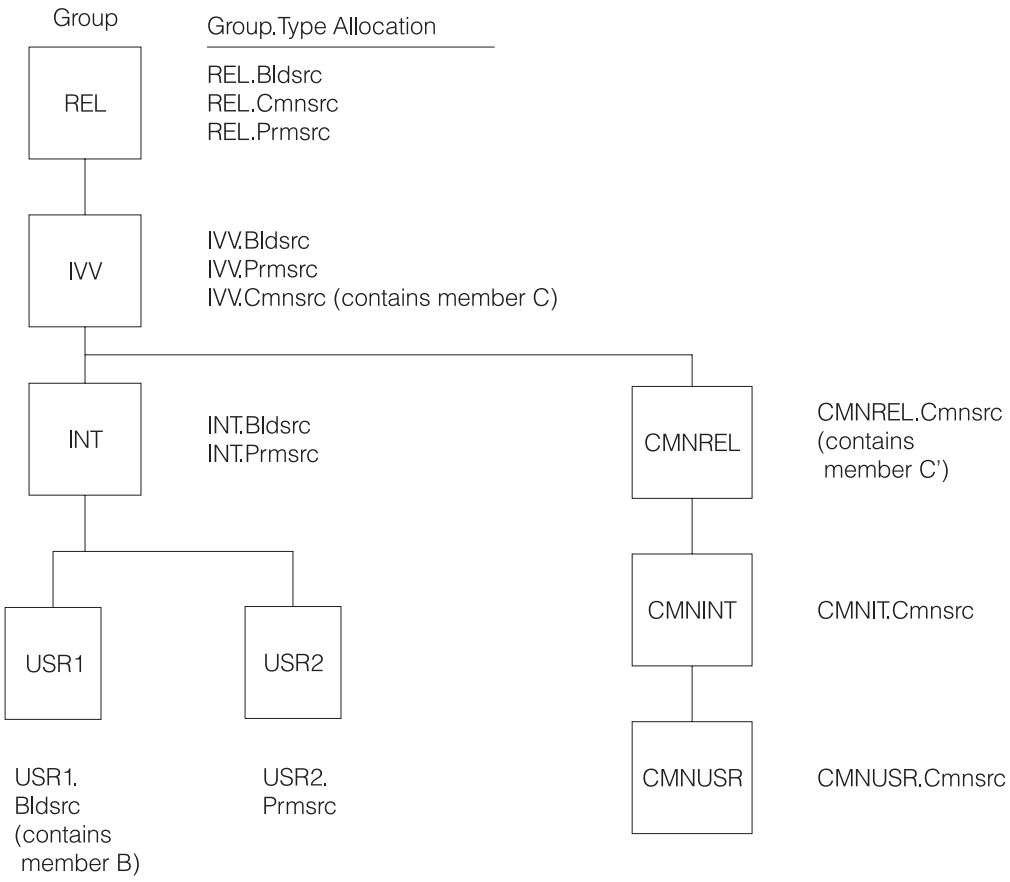


Figure 10. Valid Hierarchy with Unallocated Data Sets

A hierarchy that is not valid for the intended operation is shown in Figure 11 on page 17. A promote of member B from the IVV group, which INCLUDES member C, will fail, because promote will attempt to copy member C in IVV.Cmnsrc to REL.Cmnsrc.

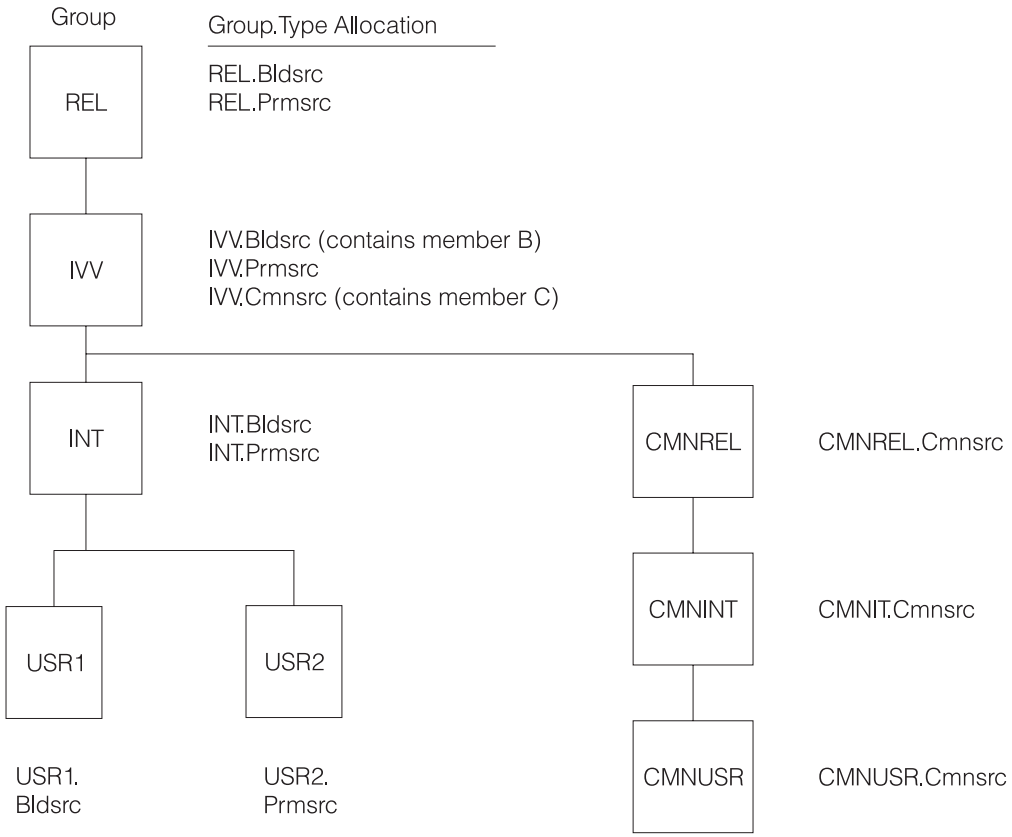


Figure 11. Invalid Hierarchy for Intended Operation

Versioning Partitioned Data Sets

If the versioning capability is going to be used, at least one versioning partitioned data set must be allocated. If you are going to use the VERCOUNT parameter on the FLMCNTRL macro to specify that two or more versions be maintained, then you must specify at least one versioning partitioned data set for each group to be versioned. Otherwise, errors can occur during version retrieval. You can also choose to have a versioning partitioned data set associated with each 'group.type' to be versioned.

The format in Table 2 shows the attributes required for the versioning partitioned data set. All attributes must be coded as shown, with the exception of the LRECL value, which defines the minimum LRECL allocation required for versioning. The LRECL value must be at least 259 and must be 4 bytes more than the LRECL of the largest source data set to be versioned.

Table 2. Versioning Data Set Attributes

LRECL =	259
RECFM =	Variable Blocked (VB)
BLKSIZE =	LRECL + 4 Bytes. Use the optimal block size for your system.

The 4 bytes in the block size calculation are for MVS control information, specifically for the *blocklength* field. For example, with a blocking factor of 10 the block size would be calculated as follows:

```
(259 x 10) + 4 = 2594
```

Project Partitioned Data Sets

This section provides guidance on what data set attributes should be used for the project partitioned data sets. SCLM does not restrict the format of a data set. **However, data sets of the same type must be allocated with the same attributes**. Table 3 shows a list of recommended data set attributes for some typical types. For best performance, use system determined blocksize (blocksize=0). Load module data sets should be allocated with a blocksize of 6144 or greater.

Table 3. Data Set Attributes

Type	RECFM	LRECL
Source	FB	80
Object	FB	80
Load	U	0
Listings	VB	137
Linkedit Maps	FBM	121
Architecture definitions	FB	80
Other Text	FB	80

Space Considerations

SCLM has no special considerations that require the allocation of additional space in the project partitioned data sets. Allocate the size of the project partitioned data sets according to the amount of data that will be stored in them.

Step 6: Allocate and Create the Control Data Sets

Control data sets are used to track and control application programs within the hierarchy. SCLM stores accounting and audit information in VSAM data sets whose names are defined in the project definition. VSAM data sets consist of VSAM clusters. A *VSAM cluster* is a named structure consisting of a group of related components. While it is not required that the first qualifier of VSAM data sets match the project name, it makes project maintenance easier. There are seven types of VSAM data sets that can be associated with a project.

Primary Accounting
: The accounting data set contains information about the software components in the project including statistics, dependency information and build maps (information about the last build of the member). At least one accounting data set is required for a project.

Secondary Accounting
: The secondary accounting data set is a backup of the information in the accounting data set.

Export Accounting
: The export accounting data set contains accounting information that has been exported from the accounting data set.

Primary Audit Control
: The audit control data set contains audit information about changes to the software components in the project for groups that have auditing turned on.

Secondary Audit Control
: The secondary audit control data set is a backup of the information in the audit control data set.

Most projects start out with one VSAM data set, the primary accounting data set. Additional data sets can be added as the project evolves and more advanced SCLM capabilities are needed. Additional VSAM data sets are required for Import, Export, Auditing, automatic backup of accounting data and multiple control data set support. In some cases, it is desirable to use multiple VSAM data sets instead of one or two. If this is the case, see "Splitting Project VSAM Data Sets" on page 65 for additional information.

SCLM uses VSAM Record Level Sharing (RLS) to support sharing the VSAM data sets across systems in a sysplex environment. This support requires:

- the Coupling Facility
- DFSMS 1.3 or later
- a VSAM cluster allocated with the proper characteristics for VSAM RLS
- VSAMRLS=YES specified on the FLMCNTRL macro in the SCLM project definition.

Refer to the DFSMS documentation for additional information about the hardware and software requirements to support VSAM RLS.

The VSAM data sets cannot be shared under any other condition. Accessing any of the VSAM data sets from multiple systems when VSAM RLS is not available can result in the corruption of data, system errors, or other integrity problems. To avoid these problems, the project manager must allocate VSAM data sets so that they cannot be accessed from multiple systems.

All VSAM data sets should be REPROed periodically using the IDCAMS reproduction utility. This will reduce fragmentation and optimize the performance of your VSAM data sets.

Create the Accounting Data Sets

The accounting data sets contain information about the application programs in the hierarchy, including statistics, dependency information, and build maps. SCLM functions use the accounting information to control and track members in the project partitioned data sets. Each project must have at least one primary accounting data set.

An optional secondary accounting data set can be created. The secondary accounting data set is a backup for the primary accounting data set and allows for the restoration of accounting information in the case where the primary data set becomes corrupted. This might happen due to a head crash. A unique name for this data set must be chosen. The secondary accounting data set should be put on a different volume than the primary accounting data set. If a secondary data set is used, the performance of SCLM will be degraded, because updates are made to both the primary and secondary data sets. The information in both data sets should be compared periodically to ensure the integrity of the accounting information.

Both the primary and secondary accounting data sets are created the same way. Each accounting data set for the project must be a VSAM cluster. Use the IDCAMS utility to define accounting data sets. If accounting information for different groups

is going to be kept in separate accounting data sets, additional accounting data sets must be created. An example of the JCL used to define an accounting data set follows:

Note: This example is called FLM02ACT and is in the data set ISP.SISPSAMP that is shipped with ISPF. The ISP.SISPSAMP data set also contains a sample for the allocation of the data set for Record Level Sharing. It is called FLM02RLS.

```
//jobname  JOB (wkpkg,dpt,bin),'name'
//* code additional JOBCARD statements here
//*********************************************************************
//*
//*  THIS JCL EXAMPLE DEFINES A VSAM CLUSTER TO BE USED AS THE SCLM
//*    ACCOUNTING FILE FOR A GIVEN PROJECT.
//*  THE HIGH-LEVEL QUALIFIER MUST BE AN ENTRY IN A VSAM USER CATALOG
//*    IN ORDER TO CREATE THIS CLUSTER.
//*  TO SPECIFY THE FILE, CHANGE THE DEFINE CLUSTER STATEMENT BELOW
//*    AS FOLLOWS:
//*
//*    1) ADD THE FOLLOWING LINE OF JCL TO DELETE THE VSAM CLUSTER
//*       BEFORE THE ALLOCATION IF THE DATA SET ALREADY EXISTS
//*       AND IT NEEDS TO BE DELETED:
//*       DELETE 'project.account.file' CLUSTER
//*       ADD THIS STATEMENT BETWEEN THE //SYSIN ALLOCATION AND THE
//*       DEFINE CLUSTER LINE OF JCL.
//*    2) CHANGE ALL project.account.file TO THE DESIRED FILE NAME.
//*        THIS VALUE IS SPECIFIED ON THE FLMCNTRL AND FLMALTC
//*        MACROS.  IF MORE THAN ONE VSAM ACCOUNTING DATA SET IS
//*        SPECIFIED ON THE FLMCNTRL AND FLMALTC MACROS, MULTIPLE
//*        IDCAMS DEFINE STEPS ARE REQUIRED.
//*        ACCOUNTING DATASET NAMES ARE USUALLY CHOSEN IN THE FOLLOWING
//*        MANNER - "PROJECT.ACCOUNT.FILE" (WHICH IS THE DEFAULT
//*        USED IN THE PROJECT DEFINITION IF NONE IS SPECIFIED).
//*    3) MODIFY CYLINDERS (PRIMARY SECONDARY)
//*    4) SPECIFY THE VOLUME VVVVVV ON WHICH IT WILL BE ALLOCATED
//*
//*  A JOB STEP IS THEN EXECUTED TO INITIALIZE THE FILE.
//*
//*********************************************************************
//STEP1   EXEC PGM=IDCAMS
//*
//SYSPRINT  DD SYSOUT=H
//*
//SYSIN     DD *
     DEFINE CLUSTER +
        (NAME('project.account.file') +
         CYLINDERS(4 1) +
         VOLUMES(VVVVVV) +
         KEYS(26 0) +
         IMBED +
         RECORDSIZE(264 32000) +
         SHAREOPTIONS(4,3) +
         SPEED +
```

Figure 12. Accounting File Example (Part 1 of 2)

```
        SPANNED +
        UNIQUE) +
        INDEX(NAME('project.account.file.INDEX') -
        ) +
        DATA(NAME('project.account.file.DATA') -
        CISZ(2048) +
        FREESPACE(50 50) +
        )
/*
//*******************************************************************
//*
//*  INITIALIZE THE ACCOUNTING FILE
//*
//*******************************************************************
//STEP2    EXEC PGM=IDCAMS
//INPUT  DD *
                       SCLM ACCOUNTING FILE INITIALIZATION RECORD
/*
//OUTPUT DD DSN=project.account.file,DISP=SHR
//SYSPRINT  DD SYSOUT=H
//SYSIN     DD *
     REPRO INFILE(INPUT) OUTFILE(OUTPUT)
/*
//*
)CM 5665-402 (C) COPYRIGHT IBM CORP 1980, 1989
```

Figure 12. Accounting File Example (Part 2 of 2)

Space Considerations for the Accounting Data Sets: Each accounting data set requires approximately three cylinders of 3390 DASD for every 1000 partitioned data set members that SCLM controls. The space required varies depending on how much information SCLM will control. If additional space in the data set is desired, modify the space parameter (shown as CYLINDERS in the example JCL).

Create the Export Data Sets

The export control data sets are optional unless the export and import functions are used.

Before using the EXPORT service, you must allocate and define an export accounting data set.

To prepare for the export operation:

1. Define the export accounting data sets to the project using the FLMCNTRL and FLMALTC macros. Do not use data set names that have already been specified for any ACCT or ACCT2 parameters in the FLMCNTRL and FLMALTC macros.

   Note: SCLM variables, including @@FLMPRJ, @@FLMGRP, and @@FLMUID, can be used when you specify the name of the accounting VSAM data sets.

2. Use the EXPACCT parameter on the FLMCNTRL and FLMALTC macros to specify the name of the export accounting data sets. This example illustrates how to use this parameter:

```
        FLMCNTRL ACCT=SCLM.ACCOUNT.DATABASE,                        C
                 EXPACCT=SCLM.EXPORT.ACCOUNT.DATABASE

SAMPLE  FLMALTC  ACCT=SCLM.ACCOUNT.SAMPLE,                         C
                 EXPACCT=SCLM.EXPORT.ACCOUNT.SAMPLE
```

Create the Audit Control Data Sets

The audit control data sets contain information about changes to SCLM-controlled members that are located in groups being audited. The audit control data sets are only required if the audit function is used. You must create the audit control data sets before the audit function is enabled. If auditing is used, each project must have at least one primary audit control data set.

You can create an optional secondary audit control data set. The secondary audit control data set is a backup for the primary audit control data set. It allows you to restore audit control information if the primary audit control data set is corrupted. Choose a unique name for this data set and put it on a different volume than the primary audit control data set. If a secondary data set is used, SCLM's performance will be degraded because updates are made to both the primary and secondary audit control data sets. The information in both data sets should be compared periodically to ensure the integrity of the audit control information.

Use the IDCAMS utility to define audit control data sets. Each audit control data set for the project must be a VSAM cluster. If audit control information for different groups will be kept in separate audit control data sets, you must create additional audit control data sets. The following JCL example defines audit control data sets.

Note: This example JCL is called FLM02VER and is in data set ISP.SISPSAMP that is shipped with SCLM.

```
//jobname  JOB (wkpkg,dpt,bin),'name'
//* code additional JOBCARD statements here
//*********************************************************************
//*
//*  THIS JCL EXAMPLE DEFINES A VSAM CLUSTER TO BE USED AS THE
//*    AUDIT CONTROL DATA SET FOR A GIVEN PROJECT.
//*  THE HIGH LEVEL QUALIFIER MUST BE AN ENTRY IN A VSAM CATALOG
//*    IN ORDER TO CREATE THIS CLUSTER.
//*  TO SPECIFY THE FILE, CHANGE THE DEFINE CLUSTER STATEMENT BELOW
//*    AS FOLLOWS:
//*
//*    1) ADD THE FOLLOWING LINE OF JCL TO DELETE THE VSAM CLUSTER
//*       BEFORE THE ALLOCATION IF THE DATA SET ALREADY EXISTS
//*       AND IT NEEDS TO BE DELETED:
//*       DELETE 'project.version.file' CLUSTER
//*       ADD THIS STATEMENT BETWEEN THE //SYSIN ALLOCATION AND THE
//*       DEFINE CLUSTER LINE OF JCL.
//*    2) CHANGE ALL project.version.file TO THE DESIRED FILE NAME.
//*        THIS VALUE IS SPECIFIED ON THE FLMCNTRL AND FLMALTC
//*        MACROS.  IF MORE THAN ONE VSAM ACCOUNTING DATA SET IS
//*        SPECIFIED ON THE FLMCNTRL AND FLMALTC MACROS, MULTIPLE
//*        IDCAMS DEFINE STEPS ARE REQUIRED.
//*    3) MODIFY CYLINDERS (PRIMARY SECONDARY)
//*    4) SPECIFY THE VOLUME VVVVVV ON WHICH IT WILL BE ALLOCATED
//*
//*  A JOB STEP IS THEN EXECUTED TO INITIALIZE THE FILE.
//*
//*********************************************************************
//STEP1   EXEC PGM=IDCAMS
//*
//SYSPRINT  DD SYSOUT=H
//*
//SYSIN     DD *
     DEFINE CLUSTER +
        (NAME('project.version.file') +
         CYLINDERS(4 1) +
         VOLUMES(VVVVVV) +
         KEYS(40 0) +
         IMBED +
         RECORDSIZE(264 32000) +
         SHAREOPTIONS(4,3) +
         SPEED +
         SPANNED +
         UNIQUE) +
         INDEX(NAME('project.version.file.INDEX') -
```

Figure 13. Audit Control Data Set Example (Part 1 of 2)

```
        ) +
        DATA(NAME('project.version.file.DATA') -
        CISZ(2048) +
        FREESPACE(50 50) +
        )
/*
//*******************************************************************
//*
//*  INITIALIZE THE AUDIT CONTROL FILE
//*
//*******************************************************************
//STEP2    EXEC PGM=IDCAMS
//INPUT  DD *
                        SCLM AUDIT CONTROL FILE INITIALIZATION RECORD
/*
//OUTPUT DD DSN=project.version.file,DISP=SHR
//SYSPRINT  DD SYSOUT=H
//SYSIN     DD *
     REPRO INFILE(INPUT) OUTFILE(OUTPUT)
/*
//*
)CM 5665-402 (C) COPYRIGHT IBM CORP 1980, 1989
```

Figure 13. Audit Control Data Set Example (Part 2 of 2)

Space Considerations for the Audit Data Sets: Each audit data set requires approximately one cylinder of 3390 DASD for every 100 partitioned data set members that SCLM controls. The space required varies depending on how much information SCLM will control. If you require additional space in the data set, modify the space parameter (shown as CYLINDERS in the example JCL).

Step 7: Protect the Project Environment

SCLM provides a controlled environment to maintain and track all software components. However, SCLM is not a security system. You must rely on RACF or an equivalent security system to provide complete environment security. Consider limiting authority to data sets in the hierarchy above the development layer.

The following sections describe the security requirements for the different types of data in the SCLM environment. Use this information to set up the security for the project environment. When this step is complete, the security requirements for the project environment are complete.

PROJDEFS Data Sets

The project definition LOAD data set should be restricted so that only the project manager has UPDATE authority to it. All other developers need READ access to this data set. Developers have no need to update the remaining PROJDEFS data sets and should not have UPDATE access to those data sets. READ access can be given to the other PROJDEFS data sets if this is reasonable for the project.

Project Partitioned Data Sets

- Each developer needs READ authority to all the project partitioned data sets.
- Each developer needs UPDATE authority to the development group(s) that the individual uses to change SCLM-controlled members. UPDATE authority is also required for any groups the developer is allowed to promote into.
- If the SCLM versioning capability is used, each developer needs UPDATE authority to the versioning partitioned data sets.
- If the import/export capability is enabled, each developer needs UPDATE authority to the export data sets.

- We suggest that the project manager have ALTER authority to all the project partitioned data sets.

Control Data Sets

- Each developer in the project needs UPDATE authority to the control data sets that are updated by the developers.
- Each developer needs READ access to the primary and secondary (if used) accounting data sets for all groups in the hierarchy. This authorization is required for SCLM to perform its verification.
- If cross-reference data sets are used in the project, each developer needs READ access to the cross-reference data sets for all groups.
- If the auditing capability is used, each developer needs UPDATE authority to the audit control data sets.

For more information on RACF, refer to *MVS Resource Access Control Facility (RACF) Command Language Reference,* SC28-0733.

Step 8: Create the Project Definition

The project definition defines the development environment for an individual project. The project definition is organized into three parts: the hierarchy definition, project controls, and language definitions.

- The hierarchy definition determines the structure of the hierarchy and how data moves through the hierarchy.
- Project controls define how SCLM operates for the project.
- The language definitions define the languages for the project.

When creating a project definition, it is usually easier to copy a sample project definition and make the necessary project-specific modifications. IBM supplies two sample project definitions with SCLM located in the data set ISP.SISPSAMP. The sample project definitions are named FLM@EXM1, FLM@EXM2 and FLMWBPRJ. FLM@EXM1 is an example project definition that uses several languages, such as COBOL, PL/I, and Script. FLM@EXM2 is an example project definition that shows several languages using Cross System Product, DB2, and IMS support. The FLMWBPRJ project definition example includes languages that are used to build an application on your workstation using SCLM's workstation build capability. Another example project definition (shown on page 32, but not delivered with SCLM) is used throughout this chapter as a reference in explaining how to generate the project definition.

Copy the project definition that is appropriate for your project, FLM@EXM1, FLM@EXM2 or FLMWBPRJ into your project.PROJDEFS.SOURCE data set. All project definitions and language definitions for your project should reside in your project.PROJDEFS.SOURCE data set.

Each part of the project definition uses SCLM macros to define the data so that SCLM understands it. The flexibility of these macros allows you to customize each project definition for specific purposes. *ISPF Software Configuration and Library Manager (SCLM) Reference* describes the use of these macros in detail.

Note: Because these are S/370 Assembler language macros, all rules pertaining to macros apply. In addition, there are some SCLM rules involving the use of the macros.

Alternate Project Definitions

You can generate more than one project definition for a project. Each project definition defines the relationships between groups in the project database and the processes that you can perform on the data in the project database. Each project definition can define a different database structure, specify different control options, or support different languages for the project.

Limit the use of alternate project definitions to satisfying a temporary need for a capability that the default (primary) project definition does not provide. You can use alternate project definitions successfully if they are never used to introduce or update members controlled under the primary project definition. Thus, you could use an alternate project definition to export data from the database definition or reference data in the primary database definition. However, if you use an alternate project definition to restrict an SCLM verification capability for data that is intended for the primary project definition, you can introduce integrity problems.

You can have an unlimited number of alternate project definitions for a project.

Figure 14 on page 27 shows an alternate project definition with a primary non-key integration group (DEPT) defined for the project database structure shown in Figure 7 on page 7.

```
PROJ1    FLMABEG
*
*
*    TYPE SPECIFICATION
*
ARCHDEF  FLMTYPE
DESIGN   FLMTYPE
LIST     FLMTYPE
LOAD     FLMTYPE
OBJ      FLMTYPE
SOURCE   FLMTYPE
*
*
*    GROUP SPECIFICATION, DEFINE THE AUTHORIZATION CODES
*
RELEASE  FLMGROUP AC=(REL),KEY=Y
TEST     FLMGROUP AC=(REL),KEY=Y,PROMOTE=RELEASE
INT      FLMGROUP AC=(REL),KEY=Y,PROMOTE=TEST
DEPT     FLMGROUP AC=(REL),KEY=N,PROMOTE=INT
USER1    FLMGROUP AC=(REL),KEY=Y,PROMOTE=DEPT
USER2    FLMGROUP AC=(REL),KEY=Y,PROMOTE=DEPT
USER3    FLMGROUP AC=(REL),KEY=Y,PROMOTE=DEPT
*
*
*    PROJECT CONTROLS
*
         FLMCNTRL ACCT=PROJ1.ACCOUNT.FILE,                        C
               MAXLINE=75
*
*
*    LANGUAGE DEFINITIONS
*
         COPY  FLM@ARCD     -- ARCHITECTURE   LANGUAGE    --
         COPY  FLM@TEXT     -- TEXT           LANGUAGE    --
         COPY  FLM@SCRP     -- SCRIPT 3       LANGUAGE    --
         COPY  FLM@ASM      -- 370 ASSEMBLER  LANGUAGE    --
         COPY  FLM@COBL     -- COBOL          LANGUAGE    --
         COPY  FLM@FORT     -- FORTRAN IV     LANGUAGE    --
         COPY  FLM@PSCL     -- PASCAL         LANGUAGE    --
         COPY  FLM@PLIO     -- PL/I OPTIMIZER LANGUAGE    --
         COPY  FLM@L370     -- 370 LINKAGE EDITOR         --
*
 FLMAEND
```

Figure 14. Sample Alternate Project Definition

Create the Hierachy Definition

This step discusses the hierarchy definition. When this step is complete, the hierarchy definition of the project definition is complete.

The hierarchy definition defines the project's hierarchy using groups and types. The rules for moving data within the hierarchy are defined with authorization codes. This information was created in Steps 1, 2, and 3. Modify the example project definition using the following macros and the information from Steps 1, 2, and 3 to define the hierarchy.

The macros that are used in the hierarchy definition are shown in the order that they are usually used in the project definition.

Specify the Project Name with FLMABEG: This macro defines the project name. It is required and must be the first macro in the project definition. You can use it only once. The project name must match the first qualifier of the PROJDEFS.LOAD data set.

If you want more than one project definition for a project, keep the project name in the alternate project definitions the same. See "Alternate Project Definitions" on page 26 for more information. In the example on page 32, the FLMABEG macro defines project PROJ1.

Define Authorization Groups with FLMAGRP: Use this macro to define a set (or group) of authorization codes. This macro is optional and needed only if you are defining a large number of authorization codes. You can use it multiple times.

The FLMAGRP provides a way of using an identifier to represent a list of authorization codes. If you decide to use multiple authorization codes for any of the groups in your hierarchy, it might be easier to associate an identifier with the list. If the list needs to be changed at a later date, the changes can be made on the FLMAGRP macros rather than changing the authorization code lists on all the FLMGROUP macros. The FLMAGRP macro must appear before any reference to the authorization group that it defines. The example on page 32 uses only one authorization code and therefore does not need to use FLMAGRP macros.

Define Types with FLMTYPE: Use this macro to define one type in the project hierarchy. At least one occurrence of this macro is required. You can use it multiple times.

Define the types identified in Step 2: Identify the Types of Data to Support using the FLMTYPE macro. For example, in the sample project definition depicted on page 32, type ARCHDEF is defined to contain architecture members.

Define Groups with FLMGROUP: Use this macro to define one group in the project hierarchy. At least one occurrence of this macro is required. You can use it multiple times.

Define the groups identified in Step 1: Determine the Project's Hierarchy by using the FLMGROUP macro. Each group in the hierarchy requires an FLMGROUP statement.

The authorization codes defined in Step 3: Establish Authorization Codes must also be defined now. Use the AC parameter on the FLMGROUP macro to define the authorization codes listed in Step 3: Establish Authorization Codes. The example on page 32 shows a project definition with only one authorization code defined.

End the Definition with FLMAEND: This signifies the end of the project definition. It must be the last macro in the project definition and is required. You can use it only one time.

Set the Project Control Options

The project control options dictate SCLM processing for an individual project. When this step is complete, the project controls of the project definition will be set up for the new project. Use project control options to specify:

- Primary accounting data set
- Secondary accounting data set
- Export accounting data set
- Audit control data set
- VSAM Record Level Sharing

- Versioning partitioned data set
- Project partitioned data set naming conventions
- Maximum lines per page
- Number of versions to keep
- Translator option override
- SCLM temporary data set allocation
- Change code verification routine
- Build and promote user exit routine

The following macros that can be used in the control section of the project definition are shown in the order that they are usually used in the project definition:

FLMCNTRL Use this macro to specify project-specific control options. The options on FLMCNTRL apply to the entire project. This macro is optional unless you change any of SCLM's default control options. You can use it one time.

FLMALTC Use this macro to provide alternate control for individual groups. This macro is used to override certain options on the FLMCNTRL macro for specific groups. The options on the FLMALTC macro apply only to the groups using it. This macro is optional. You can use it multiple times.

FLMATVER Use this macro to enable the audit and version capability and to define the type of data, (audit or audit and versioning, to capture with the capability. If a project is using the versioning capability, it must also use the audit capability. This macro is optional. You can use it multiple times.

Primary Accounting Data Set Specification: The ACCT control option specifies the name of the primary accounting data set. The data set you specify must be the name of the VSAM cluster you want to use. The default accounting cluster name is `project.ACCOUNT.FILE`, where `project` is the 8-character name for the project.

In the example of a project definition on page 32, the primary accounting data set name is `PROJ1.ACCT.FILE`.

Secondary Accounting Data Set Specification: The ACCT2 control option specifies the name of a backup VSAM accounting data set for the project. If a severe problem occurs with the primary accounting data set (for example, a head crash on that disk), you could use this data set as a backup to restore the primary accounting information.

If you use this option, additional VSAM updates to the secondary accounting data set take place and can affect SCLM's performance.

Export Accounting Data Set Specification: The EXPACCT control option specifies the name of the export accounting data set. The data set you specify must be the name of the VSAM cluster you want to use. The following variables can be used in specifying the name of the export accounting data set name:
- @@FLMPRJ
- @@FLMGRP
- @@FLMUID

The EXPACCT control option must have a data set name that is different from the ACCT or ACCT2 control option specified in FLMCNTRL or any FLMALTC macro.

The example project definition found on page 32 does not specify an export accounting data set.

Audit Control Data Sets Specification: The audit control data sets are optional. They only need to be specified if SCLM's auditing capability will be used. The VERS and VERS2 control options are used to specify the audit control data sets created in "Step 6: Allocate and Create the Control Data Sets" on page 18. The VERS control option specifies the primary audit control data set. The VERS2 control option specifies the secondary audit control data set that is a backup for the primary audit control data set. When using the auditing capability, the secondary audit control data set is optional. The FLMALTC macro can be used to specify different audit control data sets on specific groups.

VSAM Record Level Sharing (RLS): The VSAMRLS control option indicates whether or not VSAM Record Level Sharing should be used when the level of DFSMS on the system is 1.3 or later. The default value is NO. The example found in this chapter does not use VSAM Record Level Sharing.

Versioning Partitioned Data Sets Specification: Specifying the names of versioning partitioned data sets is optional. The VERPDS control option allows you to specify the names of partitioned data sets that will contain the versioned data for a project. If the names of the versioning partitioned data sets will be different for specific groups, the FLMALTC macro must be used to associate the names of the versioning partitioned data sets with the specific groups. The following variables can be used in specifying the name of the versioning partitioned data set name:

- @@FLMPRJ
- @@FLMGRP
- @@FLMTYP
- @@FLMDSN

Project Partitioned Data Set Naming Conventions: The DSNAME control option is used to specify a naming convention other than the SCLM default for the project partitioned data sets. The DSNAME option allows the project manager to specify the naming convention for all the data sets in the hierarchy. If the naming convention of the project partitioned data sets will be different for specific groups then the FLMALTC macro must be used so the naming convention for the data sets associated with the specific groups will be changed. For more information on modifying the naming convention for project partitioned data sets see "Flexible Naming of Project Partitioned Data Sets" on page 13.

Maximum Lines Per Page: Use the MAXLINE control option to specify the maximum lines per page for all SCLM-generated reports. The default is 60. The minimum number of lines per page is 35. In the example project definition on page 32, the maximum number of lines per page defaults to 60.

Number of Versions to Keep: Use the VERCOUNT parameter to specify how many versions of a member to keep. The default value of zero, used in the example found in this chapter, indicates that all versions are kept. The number of versions specified using this parameter applies to all types that are versioned. The VERCOUNT parameter on the FLMATVER macro can be used to override this value for specific types.

Valid values are 0 and any integer value greater than or equal to 2. Because that is what is already in the hierarchy, 1 is not a valid value. If you specify a value other than the default and you intend to version multiple groups in the hierarchy, either

use the FLMALTC macro to specify different VERPDS data sets for each group or use the @@FLMGRP variable in the VERPDS name on the FLMALTC macro. Failure to allocate and specify unique VERPDS data sets can result in difficulty retrieving versions.

Translator Option Override: The OPTOVER control option allows you to keep developers from overriding project-defined translator options. If you specify `Y`, developers can override the translator options for any of the languages by using the PARM statement in the architecture members. For additional information on translator options, see Part One of this book.

If you specify `N`, SCLM uses only translator options you specify in the language definition for the translators. Specifying `N` also overrides the OPTFLAG parameter, which allows option override by the translator. The default for the OPTOVER control option is `Y`. In the example project definition on page 32, the OPTOVER option defaults to `Y`.

SCLM Temporary Data Set Allocations: Many installations specify one or more I/O unit names as Virtual Input Output (VIO) devices at system generation time. Use of these devices typically improves system performance by eliminating much of the overhead and time required to move data physically between main storage and an I/O device.

To take advantage of this facility, specify the name of the VIO unit in your project definition as the VIOUNIT parameter on the FLMCNTRL macro. This unit will be used for all temporary data sets under the following conditions:

- IOTYPE = O, P, S, or W
- CATLG = N
- RECNUM <= the MAXVIO parameter.

Some of the temporary data sets used by versioning will use the VIO unit as well as long as the size of the temporary data set to be allocated is less than or equal to the MAXVIO value.

Temporary data set allocations that fail to meet any of the preceding conditions will be allocated using the unit specified via the DASDUNIT parameter on the FLMCNTRL macro.

The default value for MAXVIO is 5000, and the maximum allowable value is 2147483647. A relatively large value should be specified in order to ensure that SCLM temporary data sets are allocated using the VIO unit. If SCLM functions fail for lack of memory (S80A ABEND or S878 ABENDs), try reducing this value.

The size of the temporary data sets allocated for translators is determined by the attributes specified on the FLMALLOC macros in the language definition. The size of the temporary data sets used by versioning is based on the attributes of the source data set being versioned.

User Exit Routine Specification: SCLM provides a number of exit points that you can use to customize SCLM processing or to integrate SCLM with other products. You can specify your own user exit routines in the project definition using the user exit parameters on the FLMCNTRL macro. A sample user exit for use with the Tivoli Service Desk is provided by ISPF. See “Chapter 6. Using SCLM and Tivoli Service Desk for OS/390” on page 127 for more information.

See “Chapter 2. User Exits” on page 51 for more information.

Example Project Definition: Figure 15 shows an example of a project definition. The source for this example can be found in the ISPF sample library, ISP.SISPSAMP, member FLM@EXM1.

```
         TITLE '*** PROJECT DEFINITION FOR PROJECT=PROJ1 ***'
PROJ1    FLMABEG
*
*        ************************************************************
*        *  DEFINE THE AUTHORIZATION CODES                          *
*        ************************************************************
GRP1     FLMAGRP AC=(A1,B1,C1)
GRP2     FLMAGRP AC=(A2,B2,C2)
GRPALL   FLMAGRP AC=(GRP1,GRP2)
*
*        ************************************************************
*        *  DEFINE THE TYPES                                        *
*        ************************************************************
*
ARCHDEF  FLMTYPE EXTEND=SOURCE
COMP     FLMTYPE
DICT     FLMTYPE
DOCS     FLMTYPE
LINKLIST FLMTYPE
LIST     FLMTYPE
LMAP     FLMTYPE
LOAD     FLMTYPE
OBJ      FLMTYPE
OBJ1     FLMTYPE
OBJ2     FLMTYPE
SCRIPT   FLMTYPE EXTEND=SOURCE
SOURCE   FLMTYPE
*
```

Figure 15. Example Project Definition (Part 1 of 3)

```
*        ************************************************************
*        *  DEFINE THE GROUPS                                       *
*        ************************************************************
*
DEV1     FLMGROUP AC=(GRP1),KEY=Y,PROMOTE=TEST
DEV2     FLMGROUP AC=(GRP2),KEY=Y,PROMOTE=TEST
TEST     FLMGROUP AC=(GRP1),KEY=Y,PROMOTE=RELEASE
RELEASE  FLMGROUP AC=(GRPALL),KEY=Y,ALTC=RELDB
*
*********************************************************************
*               PROJECT CONTROLS
*********************************************************************
*
        FLMCNTRL ACCT=PROJ1.ACCT.FILE,                              C
              VERS=PROJ1.VER1.FILE,                                 C
              VERS2=PROJ1.VER2.FILE,                                C
              MAXVIO=999999,                                        C
              VIOUNIT=VIO
*
RELDB   FLMALTC ACCT=PROJ1.ACCT.FILEX,                              C
              VERS=PROJ1.VER1.FILEX,                                C
              VERS2=PROJ1.VER2.FILEX
*
*********************************************************************
*               VERSIONING AND AUDITABILITY                         *
*********************************************************************
*
*
        FLMATVER GROUP=TEST,                                        C
              TYPE=SOURCE,                                          C
              VERSION=YES
*
        FLMATVER GROUP=RELEASE,                                     C
              TYPE=SOURCE,                                          C
              VERSION=YES
*********************************************************************
*               LANGUAGE DEFINITION TABLES
*********************************************************************
*
*
*********************************************************************
* NON-COMPILERS
*********************************************************************
*
         COPY  FLM@ARCD          -- ARCHITECTURE DEF. LANGUAGE --
         COPY  FLM@CLST          -- CLIST             LANGUAGE --
         COPY  FLM@REXX          -- REXX              LANGUAGE --
         COPY  FLM@REXC          -- REXX PARSER AND COMPILER   --
         COPY  FLM@TEXT          -- TEXT              LANGUAGE --
         COPY  FLM@SCRP          -- SCRIPT 3          LANGUAGE --
         COPY  FLM@BOOK          -- SCRIPT/BOOKMASTER LANGUAGE --
*
*********************************************************************
* REXX PARSERS WITH STANDARD COMPILERS
*********************************************************************
*
         COPY  FLM@RASM          -- 370 ASSEMBLER H   LANGUAGE --
         COPY  FLM@RC37          -- 370 C             LANGUAGE --
         COPY  FLM@RCBL          -- COBOL II          LANGUAGE --
*
```

Figure 15. Example Project Definition (Part 2 of 3)

```
*********************************************************************
* STANDARD COMPILERS USING SYSTEM MACRO LIBRARIES
*********************************************************************
*
COBOL    FLMSYSLB SYS1.EXAMPLE.MACROS
         COPY  FLM@ASM            -- 370 ASSEMBLER     LANGUAGE --
         COPY  FLM@ASMH           -- 370 ASSEMBLER H   LANGUAGE --
         COPY  FLM@C370           -- 370 C             LANGUAGE --
         COPY  FLM@CPLK           -- 370 C + PRE-LINK  LANGUAGE --
         COPY  FLM@CLNK           -- 370 C PRE-LINK/LINK-EDIT   --
         COPY  FLM@COBL           -- COBOL             LANGUAGE --
         COPY  FLM@COB2           -- COBOL II          LANGUAGE --
         COPY  FLM@FORT           -- FORTRAN IV        LANGUAGE --
         COPY  FLM@HLAS           -- HIGH LEVEL ASSEM. LANGUAGE --
         COPY  FLM@PSCL           -- PASCAL            LANGUAGE --
         COPY  FLM@PLIC           -- PL/I CHECKOUT     LANGUAGE --
         COPY  FLM@PLIO           -- PL/I OPTIMIZER    LANGUAGE --
*
*********************************************************************
* LANGUAGE DEFINITIONS TO SUPPORT OBJ AND LOAD WITHOUT SOURCE
*********************************************************************
*
         COPY  FLM@OBJ            -- DUMMY LANG DEF TO MIGRATE OBJ --
         COPY  FLM@COPY           -- COPY OBJ TO OUTPUT TYPE    --
*
*********************************************************************
* LINKAGE EDITORS                                                   *
*********************************************************************
*
         COPY  FLM@L370           -- 370 LINKAGE EDITOR         --

*
*********************************************************************
*
         FLMAEND
*
* 5665-402 (C) COPYRIGHT IBM CORP 1992, 1990
```

Figure 15. Example Project Definition (Part 3 of 3)

Define the Language Definitions

Language Definitions define the languages and translators that a project uses. SCLM functions invoke translators (such as compilers, parsers, and linkage editors) based on a member's language. The language definition defines the translators used by each language. Each language can have multiple translators defined for it. The translators can be IBM program products, independent program products, or user-written translators.

IBM provides examples of language definitions for many commonly used languages such as COBOL and PL/I.

Table 4. Language Definitions Supplied with SCLM

Compilers and Linkage Editors	Language Definitions
Architecture definition	FLM@ARCD (noncompiler)
BookMaster	FLM@BOOK (noncompiler)
CICS map groups	FLM@BMS
CLIST	FLM@CLST (noncompiler)
COBOL OS/VS	FLM@COBL

Table 4. Language Definitions Supplied with SCLM (continued)

Compilers and Linkage Editors	Language Definitions
COBOL OS with CICS preprocessing	FLM@CCOB
COBOL OS with DB2 preprocessing	FLM@2COB
COBOL OS with DB2 and CICS preprocessing	FLM@ECOB
COBOL II	FLM@COB2
COBOL II with CICS preprocessing	FLM@CICS
COBOL II with DB2 preprocessing	FLM@2CO2
COBOL II with DB2 and CICS preprocessing	FLM@ECO2
COBOL II with member expansion and CICS preprocessing	FLM@ICO2
COBOL	FLM@RCBL (COBOL parser written in REXX)
C/C++ for MVS	FLM@RCIS (C/C++ parser written in REXX)
C/370	FLM@C370, FLM@RC37 (C/370 parser written in REXX)
C/370 with CICS preprocessing	FLM@CC
C/370 with DB2 preprocessing	FLM@2C
C/370 with DB2 and CICS preprocessing	FLM@EC
C/370 with member expansion and CICS preprocessing	FLM@IC
C/370 with pre-link	FLM@CPLK
C/370 pre-link with link-edit	FLM@CLNK
DB2	See Table 21 on page 277
FORTRAN IV	FLM@FORT
FORTRAN IV with DB2 preprocessing	FLM@2FRT
Object language definition to migrate object modules into SCLM as outputs (non-editable)	FLM@COPY
Object/Load dummy language definition to migrate object and load into SCLM as inputs (editable)	FLM@OBJ
Pascal	FLM@PSCL
PL/I Checkout Compiler	FLM@PLIC
PL/I Optimizer with DB2 preprocessing	FLM@2PLO
PL/I Optimizing Compiler	FLM@PLIO
PL/I Optimizer with CICS preprocessing	FLM@CPLO
PL/I Optimizer with DB2 and CICS preprocessing	FLM@EPLO
PL/I Optimizer with member expansion and CICS preprocessing	FLM@IPLO
REXX	FLM@REXX (noncompiler) FLM@REXC (compiler)

Table 4. Language Definitions Supplied with SCLM (continued)

Compilers and Linkage Editors	Language Definitions
Language Parsers written in REXX	FLM@RASM (Assembler), FLM@RCBL (COBOL), FLM@RC37 (C/370), FLM@RCIS (C/C++ for MVS)
SCRIPT 3	FLM@SCRP (noncompiler)
S/370 Assembler	FLM@ASM
S/370 Assembler with DB2 preprocessing	FLM@2ASM
S/370 Assembler F with CICS preprocessing	FLM@ASMC
S/370 Assembler F with DB2 and CICS preprocessing	FLM@EASM
S/370 Assembler with member and CICS preprocessing	FLM@IASM
S/370 Assembler H	FLM@ASMH, FLM@RASM (Assembler parser written in REXX)
S/370 High Level Assembler	FLM@HLAS
S/370 Linkage Editor	FLM@L370
TEXT	FLM@TEXT (noncompiler)

All the example language definitions are located in the data set ISP.SISPMACS that is shipped with SCLM.

The ISPF Sample and Macro libraries contain a number of files to support SCLM workstation builds. The ISPF Sample Library contains the following:

- FLMWBMIG - Sample migration EXEC for IBM CSET++ for OS/2 “Hello World 6” sample
- FLMWBUSR - Sample USERINFO file
- FLMWBAIO - Sample ACTINFO file for IBM CSET++ for OS/2 “Hello World 6” sample
- FLMWBAIW - Sample ACTINFO file for Borland (TM) C++ “Hello World” sample
- FLMWBAIX - Sample ACTINFO file for IBM CSET++ for AIX
- FLMWBTMP - Sample workstation language definition template
- FLMWBOS2 - High-level architecture definition to build IBM CSET++ for OS/2 “Hello World 6” sample
- FLMWBIPF - Architecture definition to build IBM CSET++ for OS/2 “Hello World 6” help file
- FLMWBDLL - Architecture definition to build IBM CSET++ for OS/2 “Hello World 6” DLL file
- FLMWBEXE - Architecture definition to build IBM CSET++ for OS/2 “Hello World 6” EXE file
- FLMWBWIN - High-level architecture definition to build Borland C++ “Hello World” sample

The Macro Library contains sample language definitions for OS/2 and Windows. The IBM CSET++ for OS/2 language definitions are:

- FLM@WICC - Compile
- FLM@WDUM - Compile dummy object to hold DLLs

- FLM@WEXE - Link EXE
- FLM@WIPF - Build Help
- FLM@WLNK - Link386 to Link the DLL
- FLM@WRC - Resource compile

The Borland (TM) C++ for Windows language definitions are:

- FLM@WBCC - Compile
- FLM@WBRC - Resource Compile
- FLM@WTLK - TLINK OBJ to EXE

The IBM CSET++ for AIX sample language definitions is:

- FLM@WXLC - Compile

This step describes how to define language definitions to the project definition. When this step is complete, all the languages your project will use will be defined.

To define the language definitions:

1. Determine what languages are used in your project.
2. Copy the appropriate example language definitions to the project.PROJDEFS.SOURCE data set allocated in "Step 4: Allocate the PROJDEFS Data Sets" on page 12.
3. Modify the language definitions.

   If you do not find an example language definition that meets your project requirements, you can write a new language definition. For instructions on defining a new language to SCLM, see "Defining a New Language to SCLM" on page 98.

Refer to the *ISPF Software Configuration and Library Manager (SCLM) Reference* for details on the use of each SCLM macro.

Modifying Example Language Definitions: Use the following macros to modify language definitions for specific project requirements.

Table 5. SCLM Macros for Language Definition

FLMSYSLB	Use this macro to define data sets that contain system, project, or language dependencies that are referenced by SCLM members but are not in the SCLM hierarchy themselves. Examples are system macros for Assembler programs and compiler-supplied include files for C programs.
FLMLANGL	Use this macro to define the language to SCLM.
FLMTRNSL	Use this macro to define a translator for a language. It can be used multiple times for a language.
FLMTOPTS	Use this macro to vary the options passed to a build translator based on the group where the build is taking place. Options can be appended to the existing options or replace the options completely. FLMTOPTS macros must follow an FLMTRNSL macro with FUNCTN=BUILD.
FLMTCOND	Use this macro to specify conditional execution of a BUILD translator. Part of the specification can include examination of return codes from previous BUILD translators in the language definition.

Table 5. SCLM Macros for Language Definition (continued)

FLMALLOC	Use this macro for each data set allocation required by a translator. If you are using a ddname substitution list, specify an FLMALLOC macro for each ddname in the correct order. If not, determine the ddnames that are needed by the translator and specify an FLMALLOC macro for each ddname.
FLMCPYLB	Use this macro to identify data sets to be concatenated to a ddname. The data sets must be preallocated. The FLMCPYLB data sets are used as input to the Parse and other translators.
FLMINCLS	Use this macro to associate sets of includes found during the parse of a member with the types in the project definition that contain those includes. FLMALLOC macros then reference this macro to allocate the include libraries for build translators. The FLMINCLS macro can be used multiple times for each language, but each FLMINCLS macro must have a unique name within the language and be associated with at least one FLMALLOC macro. This helps ensure that the includes that are found by build are the same ones found by the translators.

For each language, take the following actions as necessary:

- Specify data sets containing dependencies that are not to be tracked, such as assembler system macros (macro FLMSYSLB).
- Specify the maximum number of includes, change codes, user data records, compilation units, and external dependencies expected in a source member (macro FLMLANGL; keyword BUFSIZE).
- Determine if ddname substitution is needed for the translator. This information can be found in the translator documentation. Adjust the PORDER parameter on the FLMTRNSL macro as needed.
- Verify translator load module names and load data sets for accuracy (macro FLMTRNSL; keywords COMPILE, DSNAME, and TASKLIB).
- Adjust translator return codes to project requirements if nonzero return codes are acceptable (macro FLMTRNSL; keyword GOODRC).
- Update default translator options (macro FLMTRNSL; keyword OPTIONS).
- Verify translator version information (macro FLMTRNSL; keyword VERSION).
- Specify output listings (macro FLMALLOC; keyword PRINT).
- Specify output default types (macro FLMALLOC; keyword DFLTTYP) to match the FLMTYPE type specified in the project definition.
- Verify that system libraries are being allocated for build translators. Either specify ALCSYSLB=Y on the FLMLANGL macro or ensure that the data sets from FLMSYSLB macros are specified on FLMCPYLB macros following IOTYPE=I allocations.
- Specify the include sets for the language to use. You must specify all the include-sets returned by the parser for the language. If you add a new FLMINCLS macro, ensure that it is referenced by at least one FLMALLOC of a build translator. If you remove an FLMINCLS macro, update any FLMALLOC macros that reference it, ensuring that no member's accounting data contains references to that include set.

Figure 16 on page 39 provides an example of an OS/VS COBOL language definition.

```
*********************************************************************
*
*            OS/VS COBOL LANGUAGE DEFINITION FOR SCLM
*********************************************************************
*
         FLMLANGL    LANG=COBOL,VERSION=COBLV1.0,ALCSYSLB=Y           C
               TSLINL=80,                                             C
               TSSEQP='S 1 6 S 73 80'
*
* PARSER TRANSLATOR
*
          FLMTRNSL   CALLNAM='SCLM COBOL PARSE',                      C
               FUNCTN=PARSE,                                          C
               COMPILE=FLMLPCBL,                                      C
               PORDER=1,                                              C
               OPTIONS=(@@FLMLIS,@@FLMSTP,@@FLMSIZ,)
*          (* SOURCE      *)
           FLMALLOC  IOTYPE=A,DDNAME=SOURCE
            FLMCPYLB @@FLMDSN(@@FLMMBR)
*
* BUILD TRANSLATOR(S)
*
*         --COBOL INTERFACE--
          FLMTRNSL   CALLNAM='COBOL',                                 C
               FUNCTN=BUILD,                                          C
               COMPILE=IKFCBL00,                                      C
               VERSION=1.0,                                           C
               GOODRC=0,                                              C
               PORDER=1,                                              C
               OPTIONS=(DMA,PRI,SIZE=512K,APOS,CNT=77,BUF=30K,OPT,XREF)
*
* DDNAME ALLOCATIONS
*
 FLMALLOC  IOTYPE=O,DDNAME=SYSLIN,KEYREF=OBJ,RECNUM=5000,DFLTTYP=OBJ
 FLMALLOC  IOTYPE=I,DDNAME=SYSLIB,KEYREF=SINC
 FLMALLOC  IOTYPE=S,DDNAME=SYSIN,KEYREF=SINC,RECNUM=2000
 FLMALLOC  IOTYPE=W,DDNAME=SYSUT1,RECNUM=5000
 FLMALLOC  IOTYPE=W,DDNAME=SYSUT2,RECNUM=5000
 FLMALLOC  IOTYPE=W,DDNAME=SYSUT3,RECNUM=5000
 FLMALLOC  IOTYPE=W,DDNAME=SYSUT4,RECNUM=5000
 FLMALLOC  IOTYPE=A,DDNAME=SYSUT5
  FLMCPYLB NULLFILE
 FLMALLOC  IOTYPE=A,DDNAME=SYSUT6
  FLMCPYLB NULLFILE
 FLMALLOC  IOTYPE=A,DDNAME=SYSTERM
  FLMCPYLB NULLFILE
 FLMALLOC  IOTYPE=A,DDNAME=SYSPUNCH
  FLMCPYLB NULLFILE
 FLMALLOC  IOTYPE=O,DDNAME=SYSPRINT,KEYREF=LIST,RECFM=FBA,LRECL=133,   C
               RECNUM=5000,PRINT=Y,DFLTTYP=LIST
```

Figure 16. OS/VS COBOL Language Definition Example

In the example in Figure 16, the COBOL language is defined to SCLM by the FLMLANGL macro. The FLMTRNSL parameters specify particular information about the compiler:

- The name of the compiler: COBOL.
- The name of the compiler load module: IKFCBL00.
- The version of the compiler: 1.0.
- The compiler options: DMA, PRI, SIZE=512K, APOS, CNT=77, BUF=30K, OPT, XREF

The FLMALLOC macros following the build FLMTRNSL macro specify each ddname needed by the COBOL compiler. SCLM allocates the ddnames specified on the FLMALLOC macro before invoking the translator (in this example, the COBOL IKFCBL00 load module). The FLMALLOC parameters allow specification of the record format (RECFM), the logical record length (LRECL), the number of records (RECNUM), and other options. An FLMCPYLB macro specifies that a ddname be associated with a null data set.

The language definitions must be defined to the project definition, either by placing the language definitions directly into the project definition or having the language definitions copied into the project definition when the project definition is assembled. It is easier to maintain the project definition if each language definition is kept in a separate member and copied into the project definition when the project definition is assembled. The example project definition on page 32 uses this method of including the language definitions.

Step 9: Assemble and Link the Project Definition

Assemble all project definitions with the SCLM macro set using the standard IBM S/370 Assembler. Once assembled, link the object code using the standard IBM S/370 linkage editor and store the load module into the project.PROJDEFS.LOAD data set. All project definitions must reside in the project.PROJDEFS.LOAD data set to allow SCLM to be invoked correctly. SCLM accesses the project definition's load module when SCLM is invoked. If the project definition is updated, reassembled, and relinked while the current load module is being used, the active invocation of SCLM will not be affected.

Make sure all project definition load modules are reentrant. Nonreentrant project definition load modules can cause error conditions. Specify the RENT option during link edit. The load module name of the default project definition for a project must match the project identifier specified on the FLMABEG macro. Alternate project definitions can have any load module name, but all alternate project definitions must have the same project identifier, specified on the FLMABEG macro, as the default project definition.

The SCLM macro set performs some verification of the project definition during assembly. When warning or error conditions are detected, the macros issue *MNOTES*, which are SCLM-specific diagnostic comments. The MNOTES produced by SCLM are listed in *ISPF Messages and Codes*. If the text of an MNOTE is missing, verify that the FLMABEG macro appears at the top of the project definition and is referenced correctly. The return code from the assembler indicates the following:

0 The SCLM macros detected no errors.

4 The SCLM macros detected a potential error. The project definition might be valid, but might not reflect the desired options. Review the assembler listing for details.

8 The SCLM macros detected errors. Do not use the project definition until you correct the errors identified in the assembler listing.

Other The assembler detected errors. Examine the assembler listing for the error messages and consult the assembler's user guide for additional information. Do not use the project definition until you correct the errors identified in the assembler listing.

Assemble and Link Example

The following example illustrates JCL that assembles and links a project definition. This example can be found in member FLM02PRJ in the data set ISP.SISPSAMP that is shipped with SCLM.

```
//jobname  JOB (wkpkg,dpt,bin),'name'
//* code additional JOBCARD statements here
//*
//ASMPROJ  PROC  PROJID=,PROJDEF=
//*--------------------------------------------------------------------*
//*   ASSEMBLE AND LINK A PROJECT DEFINITION                           *
//*                                                                    *
//*   PROC PARAMETERS:                                                 *
//*                                                                    *
//*      PROJID  -  HIGH-LEVEL QUALIFIER FOR PROJECT                   *
//*      PROJDEF -  PROJECT DEFINITION MEMBER NAME                     *
//*                                                                    *
//*   NOTE: MODIFY SYSLIB DSNAMES TO GET THE SCLM RELEASE MACROS       *
//*   AND ANY LANGUAGE DEFINITIONS YOU NEED.                           *
//*--------------------------------------------------------------------*
//ASM      EXEC PGM=IEV90,REGION=400K,PARM=OBJECT
//SYSLIB   DD DSN=&PROJID;.PROJDEFS.SOURCE,DISP=SHR
//         DD DSN=ISP.SISPMACS,DISP=SHR
//SYSPRINT DD SYSOUT=H
//SYSPUNCH DD DUMMY
//SYSIN    DD DSN=&PROJID;.PROJDEFS.SOURCE(&PROJDEF),DISP=SHR
//SYSUT1   DD UNIT=SYSALLDA,SPACE=(CYL,(2,2))
//SYSLIN   DD DSN=&&INT,DISP=(,PASS),UNIT=SYSDA,SPACE=(CYL,(5,5,0)),
//            DCB=(BLKSIZE=400)
//*--------------------------------------------------------------------*
//LINK     EXEC PGM=IEWL,PARM='RENT,LIST,MAP',REGION=512K
//SYSPRINT DD SYSOUT=H
//SYSLIN   DD DSN=&&INT,DISP=(OLD,DELETE)
//OBJECT   DD DSN=&PROJID;.PROJDEFS.OBJ,DISP=SHR
//SYSLIB   DD DSN=&PROJID;.PROJDEFS.LOAD,DISP=SHR
//SYSUT1   DD UNIT=SYSALLDA,SPACE=(CYL,(2,2)),DISP=NEW
//SYSLMOD  DD DISP=SHR,DSN=&PROJID;.PROJDEFS.LOAD(&PROJDEF)
//SYSGO    DD DISP=SHR,DSN=&PROJID;.PROJDEFS.OBJ(&PROJDEF)
//         PEND
//*--------------------------------------------------------------------*
//ASMLINK  EXEC  PROC=ASMPROJ,PROJID=SCLM,PROJDEF=SCLM
//
```

Project Manager Scenario

This section describes the steps required to define and install an SCLM project. By completing the steps outlined in the following sections, the project manager can create a project that is under SCLM control. The sample project can also be defined using the SCLM sample project utility (Option 10.7). Once the project has been created, it can be used as a model for building other SCLM projects.

The project manager must perform all the steps described in this chapter before developers can follow the programmer scenario described in Part One of this book.

Prerequisites for Defining an SCLM Project

Before beginning the project definition phase of this activity, you must have the following software, space, and tools available:

- z/OS V1R1.0 ISPF with SCLM installed on an MVS system.
- PL/I Optimizing Compiler IEL0AA Version 4.0 or equivalent. (Optional if defining the project with the SCLM sample project utility.)
- Disk space to contain the data sets for the project. The project requires 265 tracks on 3390 DASD.

- Access to data set `ISP.SISPSAMP`.

  This data set is available as part of the ISPF product. It contains the project definition for this scenario and other examples. Check with the person at your site who installs ISPF to find out the name of this data set and how to allocate it.

  The member FLM01PRJ in this data set is the definition for the sample project definition used for this scenario.
- Access to data set `ISP.SISPMACS`.

  This macro library is shipped with the ISPF product and contains the macros used to assemble the project definition.
- ISPF knowledge at the user level (edit and utilities are used).
- VSAM installed.
- Rudimentary VSAM knowledge. (Not required if defining the project with the SCLM Sample Project utility.

Example Project Overview

This SCLM project contains all the required components of SCLM projects in general and serves as a model for future projects. A description of the components of the project follows.

Figure 17 shows three layers in the SCLM project hierarchy: development, test, and release.

- The development layer promotes to the test layer, and the test layer promotes to the release layer.
- The development layer is composed of the groups DEV1 and DEV2. You can think of these groups as being assigned to two separate developers. The SCLM hierarchy looks like Figure 17.

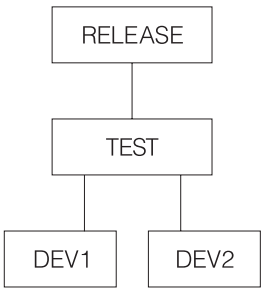


Figure 17. Example Project Hierarchy

Figure 18 on page 43 shows six modules in the hierarchy: FLM01MD1, FLM01MD2, FLM01MD3, FLM01MD4, FLM01MD5, and FLM01MD6. These are the programs that the developers edit in order to install fixes and new features.

- FLM01MD2 is written in PL/I and uses the PL/I optimization compiler.

  Note: Module FLM01MD2 and the language definition for the PLI Optimizing Compiler are not included if the project is defined using the SCLM sample project utility.
- The other five modules are written in S/370 Assembler. They include a member named FLM01EQU that contains the register equates commonly used in assembly language programs.

- The modules are compiled or assembled by the BUILD function into an application named FLM01AP1. SCLM performs this operation using the architecture definitions contained in the ARCHDEF data sets.
- FLM01AP1 does not directly call any language translators. It references other architecture members. The Build process creates the load modules FLM01LD1, FLM01LD2, FLM01LD3, and FLM01LD4.

  Note: Load module FLM01LD2 is not created if the project is defined using the SCLM sample project utility.
- FLM01AP1, FLM01SB1, and FLM01SB2 are high-level architecture members. They do not call any language translators. FLM01LD1, FLM01LD2, FLM01LD3, and FLM01LD4 are LEC architecture members. FLM01CMD is a CC architecture member, and FLM01ARH is an architecture member that is directly copied into FLM01LD3 and FLM01LD4.

  Note: Architecture member FLM01LD2 is not included if the project is defined using the SCLM sample project utility.

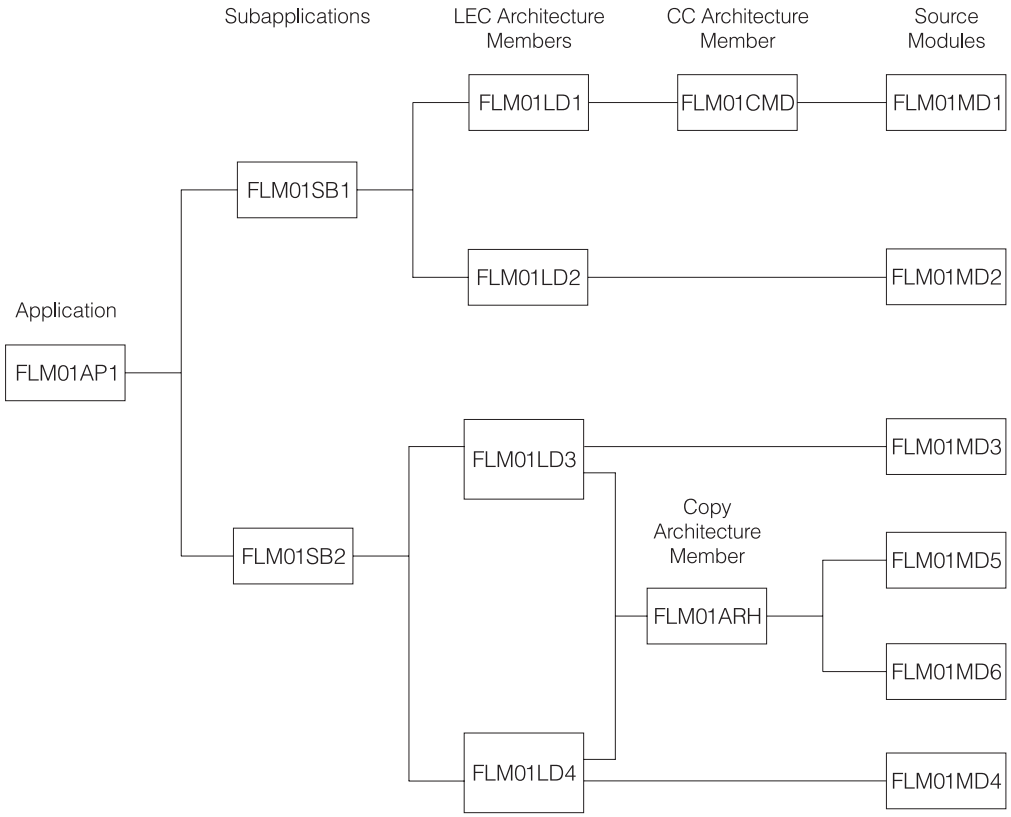


Figure 18. Example Project Architecture

Note: Source module FLM01MD2 and architecture member and load module FLM01LD2 are not included if the project was defined using the SCLM sample project utility (Option 10.7).

Preparing the Example Project Hierarchy

Use the following steps to install the example project data sets on your system. Follow the steps in the order listed and exactly as they are described. When you have completed all of the steps, you will have an SCLM project database with which you can experiment to better understand how SCLM works. If you

encounter any errors during the following steps, use the TSO, ISPF, and SCLM messages to correct the problem. You can also define the sample project using the SCLM Sample Project utility (Option 10.7).

In the descriptions that follow, the default naming convention (PROJECT.GROUP.TYPE) is used. Assume for these examples that the project name is PROJ1. If you use a different name, be sure to inform those users who plan to complete the programmer scenario.

1. Sign on to TSO.
2. At the Ready prompt, start ISPF.
3. Using the ISPF Data Set Utility, allocate the following partitioned data set with space in blocks (10,50), with 10 directory blocks, and with record format FB, LRECL 80:

   ```
   PROJ1.PROJDEFS.SOURCE
   ```

   This partitioned data set will contain the source code for the library structure as defined in the project definition.
4. Using the ISPF Data Set Utility, allocate the following partitioned data set with space in blocks (10,50), with 10 directory blocks, and with record format FB, LRECL 80:

   ```
   PROJ1.PROJDEFS.OBJ
   ```

   This partitioned data set will contain the object code for the library structure as defined in the project definition.
5. Using the ISPF Data Set Utility, allocate the following partitioned data set with space in blocks (10,50), with 10 directory blocks, and with record format U, LRECL 0, BLKSIZE 6144:

   ```
   PROJ1.PROJDEFS.LOAD
   ```

   This partitioned data set will contain the load module for the library structure as defined in the project definition. This member is named PROJ1.

   Note: Depending on the ISPF configuration for your site, you might receive warning or error messages when attempting to edit an SCLM project using the ISPF editor.
6. Use the ISPF Move/Copy Utility to copy the following members from ISP.SISPSAMP into PROJ1.PROJDEFS.SOURCE: FLM01ASM, FLM01PLI, FLM01PRJ, FLM01SCR, FLM01370, FLM02ALL, and FLM02ACT.
7. Member FLM02ALL of PROJ1.PROJDEFS.SOURCE is a background job that allocates all of the data sets needed for this example application. You must provide a job card and change any other information that is specific to your location; for example, change all the occurrences of USERID to PROJ1 and alter the job card. After you have made these changes, submit the job.

   If this job allocates all the required data sets, you can skip to Step 9. Use the ISPF Data Set List Utility to determine whether or not the data sets were allocated.

   If the required data sets have not been allocated, you can allocate them by following Step 8.
8. If Step 7 fails, or if you choose not to use the FLM02ALL JCL member, follow these steps to allocate the required data sets.
   a. Using the ISPF Data Set Utility, allocate the following partitioned data sets with space in blocks (10,50), with 10 directory blocks, and with record format FB, LRECL 80:

```
PROJ1.DEV1.SOURCE
PROJ1.DEV2.SOURCE
PROJ1.TEST.SOURCE
PROJ1.RELEASE.SOURCE
```

These partitioned data sets will contain the source code for the project.

b. Using the ISPF Data Set Utility, allocate the following partitioned data sets with space in blocks (10,50), with 10 directory blocks, and with record format FB, LRECL 80:

```
PROJ1.DEV1.ARCHDEF
PROJ1.DEV2.ARCHDEF
PROJ1.TEST.ARCHDEF
PROJ1.RELEASE.ARCHDEF
```

These partitioned data sets will contain the architecture definition for the project.

c. Using the ISPF Data Set Utility, allocate the following partitioned data sets with space in blocks (30,100), with 10 directory blocks, and with record format VB, LRECL 137:

```
PROJ1.DEV1.SOURCLST
PROJ1.DEV2.SOURCLST
PROJ1.TEST.SOURCLST
PROJ1.RELEASE.SOURCLST
```

These partitioned data sets will contain the listings from the compilations and assemblies of the modules.

d. Using the ISPF Data Set Utility, allocate the following partitioned data sets with space in blocks (15,50), with 10 directory blocks, and with record format FB, LRECL 80:

```
PROJ1.DEV1.OBJ
PROJ1.DEV2.OBJ
PROJ1.TEST.OBJ
PROJ1.RELEASE.OBJ
```

These partitioned data sets will contain the object code from the compilations and assemblies of the modules.

e. Using the ISPF Data Set Utility, allocate the following partitioned data sets with space in blocks (7,13), with 10 directory blocks, and with record format U,LRECL 0, BLKSIZE 6144:

```
PROJ1.DEV1.LOAD
PROJ1.DEV2.LOAD
PROJ1.TEST.LOAD
PROJ1.RELEASE.LOAD
```

These partitioned data sets will contain the load modules from the link edits of the modules.

f. Using the ISPF Data Set Utility, allocate the following partitioned data sets with space in blocks (5,20), with 10 directory blocks, and with record format FB, LRECL 121:

```
PROJ1.DEV1.LMAP
PROJ1.DEV2.LMAP
PROJ1.TEST.LMAP
PROJ1.RELEASE.LMAP
```

These partitioned data sets will contain the load maps from the link edits of the modules.

9. Using the ISPF Library Utility, rename member FLM01PRJ in PROJ1.PROJDEFS.SOURCE to PROJ1. This member contains the source code for the project definition for PROJ1.
10. Using ISPF Edit, edit PROJ1.PROJDEFS.SOURCE(PROJ1). Change all occurrences of USERID to PROJ1.
11. Using ISPF Edit, edit PROJ1.PROJDEFS.SOURCE(FLM01ASM). Change all system macro library references to the library of macros at your location.

    You must change the members FLM01PLI, FLM01SCR, and FLM01370 so that libraries, assemblers, and assembler options match the libraries and products in use at your location. The changes are specified in the samples delivered.

    Note: If you make changes to these members after Step 14 while installing this example project, reassemble and relink the data set PROJ1.PROJDEFS.SOURCE(PROJ1). If you are not sure this step is required, reassemble and relink.
12. Using ISPF Edit, edit PROJ1.PROJDEFS.SOURCE(FLM02ACT). Be sure that the job card contains valid accounting information. Change all occurrences of USERID to PROJ1.

    This member contains JCL that constructs the VSAM cluster used to contain the accounting information used by SCLM. You also need to alter the volumes for IDCAMS for your location, and you might need to make additional changes to conform to requirements at your location.
13. Submit the JCL in PROJ1.PROJDEFS.SOURCE(FLM02ACT). You know that your job has completed successfully when the PROJ1.ACCOUNT.FILE VSAM cluster is created.

    This is the VSAM data set that contains the SCLM accounting information for the project. This job deletes the cluster and then creates the cluster. Because the cluster does not exist the first time you submit the job, you receive a return code of 8 in the listing data set.
14. Use ISPF Foreground Assembler H to assemble PROJ1.PROJDEFS.SOURCE(PROJ1).

    Be sure that the SCLM macro library used at your location is in the concatenation sequence for the libraries used by the assembler. Specify the macro library in the **Additional Input Libraries** field on the Foreground Assembly panel.

    Look at the listing and confirm that no statements were flagged.
15. Use the ISPF Foreground Linkage Editor to link edit PROJ1.PROJDEFS.OBJ(PROJ1). This constructs the load module PROJ1.PROJDEFS.LOAD(PROJ1) that is executed by SCLM to control the library.

    Verify that the link occurred without errors.
16. Use the ISPF Move/Copy Utility to copy the following members from ISP.SISPSAMP into PROJ1.DEV1.SOURCE (these are the source members for the application and are moved into PROJ1.RELEASE.SOURCE later): FLM01EQU, FLM01MD1, FLM01MD2, FLM01MD3, FLM01MD4, FLM01MD5, and FLM01MD6.
17. Use the ISPF Move/Copy Utility to copy the following members from ISP.SISPSAMP into PROJ1.DEV1.ARCHDEF (these are the architecture definition members and are moved into PROJ1.RELEASE.ARCHDEF later): FLM01AP1, FLM01ARH, FLM01CMD, FLM01LD1, FLM01LD2, FLM01LD3, FLM01LD4, FLM01SB1, and FLM01SB2.

Understanding the Sample Project Definition

This section examines the project definition used for the library in the sample project. Typically, the project manager is responsible for developing and maintaining the project definition.

1. Select the View option from the SCLM Main Menu and type:

`PROJ1`	in the **Project** field
`DEV1`	in the **Group** field

   Press Enter.

   Type `'PROJ1.PROJDEFS.SOURCE(PROJ1)'` in the **Data Set Name** field, and press Enter to examine the member that contains the project definition for PROJ1. The macros are:

FLMABEG	FLMABEG initializes the project definition by defining the project name as PROJ1.
FLMTYPE	FLMTYPE defines each type. The type values are **ARCHDEF** architecture definitions **SOURCE** source code **SOURCLST** listings from compilers and assemblers **OBJ** object code **LMAP** load module maps **LOAD** executable load modules The type names were chosen arbitrarily for this sample project.
FLMGROUP	FLMGROUP defines each group. The PROMOTE keyword defines the library structure. Note that DEV1 and DEV2 are promoted to TEST and TEST is promoted to RELEASE.
FLMCNTRL	FLMCNTRL identifies the default VSAM data sets for the project. The VSAM data sets store library control information about the members in the project hierarchy.
COPY	COPY identifies members to be copied into the project definition. The members identified are the architecture definition language, assembler language, PL/I language, link edit language, and SCRIPT language definitions.
FLMAEND	FLMAEND ends the project definition.

   An additional developer, DEV3, can be added with another FLMGROUP macro, as shown in the following example:

```
DEV3     FLMGROUP AC=(P),KEY=Y,PROMOTE=TEST
```

   The project definition specifies the names of the partitioned data sets used by the project (for example, PROJ1.DEV1.SOURCE), the library structure for the groups (for example, DEV1 members are promoted to TEST), and the languages to be used (for example, architecture definition, ASM, PL/I, and link edit).

2. View the PROJ1.PROJDEFS.SOURCE members:

FLM01ASM	ASM language definition
FLM01PLI	PLIO language definition
FLM01370	linkage editor language definition

Note the following points about these members:

FLMSYSLB	This macro can be used to define a set of libraries that contain project and/or system macros or includes.
FLMLANGL	This macro specifies the language identifier.
FLMTRNSL	This macro is used once for each translator to be invoked for a language. The SCLM parser is invoked when the keyword FUNCTN specifies PARSE. The SCLM parser stores statistics (for example, lines-of-code counts) and dependency information (for example, includes and copy statements). The build translator is invoked when the keyword FUNCTN specifies BUILD. In FLM01370, the linkage editor IEWL is invoked. The build fails unless the return code is equal to, or less than, the value specified by the keyword GOODRC (0 in this example).
FLMALLOC	This macro is used to allocate data sets and ddnames required by translators.

Preparing the Example Project Data

The following steps prepare the example project data. You should follow the steps in the order listed and exactly as they are described. When you have completed all of the steps, all necessary data will reside at the RELEASE group. At this point, you or other SCLM users can use the data to experiment with and understand SCLM.

1. Select the SCLM option from the ISPF Primary Option panel.
2. Select the Utilities option from the SCLM Main Menu. Type:

PROJ1	in the **Project** field
DEV1	in the **Group** field

   Leave the **Alternate** field blank.

3. From the Utilities panel, select the Migration option. Type:

`SOURCE`	in the **Type** field
`FLM01MD2` (the PL/I module)	in the **Member** field
`1`	in the **Mode** field
`PLIO`	in the **Language** field
`1`	in the **Process** field
`1`	in the **Messages** field
`4`	in the **Listings** field

   Press Enter to begin processing. The migration utility registers new modules (in this case, FLM01MD2) into an SCLM library by creating accounting records for them.

4. When the migration is complete, you receive the message MIGRATION UTILITY COMPLETED with RETURN CODE = 0. The Migration Utility panel reappears. Type:

`*`	in the **Member** field
`ASM`	in the **Language** field

Press Enter to begin processing.

Notice that you did not have to type **EX** on the command line or re-enter a value in the Process field. The value is carried from panel to panel and is maintained as is until you change it.

The Migration Utility registers the SCLM accounting information for the remaining new modules (in this example, all are assembler language modules). Each time you use the Migration Utility, you can only migrate modules written in the same language. This example migrates FLM01MD2 first. After its migration, the other modules can be referenced as a group by using the asterisk (*). Because FLM01MD2 was migrated first, SCLM does not migrate it again when an * is specified.

5. When the migration is complete, you receive the message MIGRATION UTILITY COMPLETED with RETURN CODE = 0. The Migration Utility panel reappears. Type:

`ARCHDEF`	in the **Type** field
`*`	in the **Member** field
`ARCHDEF`	in the **Language** field

Press Enter to begin processing.

6. Return to the SCLM Main Menu. Select the Build option and press Enter.
7. On the Build panel, type:

`DEV1`	in the **Group** field
`ARCHDEF`	in the **Type** field
`FLM01AP1`	in the **Member** field
`/` (slash)	in the **Error Listings only** field
`1`	in the **Mode** field
`2`	in the **Scope** field
`1`	in the **Messages** field
`1`	in the **Report** field
`3`	in the **Listings** field

Press Enter. All modules in the project are assembled or compiled. SCLM updates the accounting information to indicate that a build operation was performed on each module. The Build Messages and Build Report reappears. The build should complete with a RETURN CODE = 0. The Build panel reappears.

If all of the site-dependent changes to the system macro library references were not made in 10 on page 46, build errors can occur during this step. If this happens, correct the macros, reassemble and link-edit the project definition, and repeat this step.

8. Return to the SCLM Main Menu. Select the Promote option and press Enter.
9. On the Promote panel, type:

`DEV1`	in the **From Group** field
`ARCHDEF`	in the **Type** field
`FLM01AP1`	in the **Member** field
`1`	in the **Mode** field
`1`	in the **Scope** field
`1`	in the **Messages** field
`1`	in the **Report** field

Press Enter. SCLM copies all members for all types at the DEV1 group to the TEST group and then purges all members from the DEV1 group. The Promote Messages and Promote Report appears. The Promote should complete with a RETURN CODE = 0. The Promote panel reappears.

10. On the Promote panel, type:

`TEST`	in the **From Group** field
`ARCHDEF`	in the **Type** field
`FLM01AP1`	in the **Member** field
`1`	in the **Mode** field
`1`	in the **Scope** field
`1`	in the **Messages** field
`1`	in the **Report** field
`EX`	on the command line

Press Enter. SCLM copies all members for all types at the TEST group to the RELEASE group and then purges all members from the TEST group. The Promote Messages and Promote Report appears. The Promote should complete with a RETURN CODE = 0. The Promote panel reappears.

All of the modules are located in the RELEASE group, and the SCLM example project, PROJ1, is now ready to use. This scenario illustrates the status of a current release of a product that does not have any maintenance, test, or development activities underway.

Chapter 2. User Exits

SCLM provides a number of exit points that you can use to customize SCLM processing or to integrate SCLM with other products. SCLM does not provide the user exit routines to be invoked at these exit points. You can specify your own user exit routines in the project definition using the user exit parameters on the FLMCNTRL macro.

There can be performance implications associated with the specification of an exit routine depending on the processing performed by the exit routine. You can write a user exit routine in any language, including REXX. The exit routine can use any of the SCLM services to retrieve additional information that is not returned by the exit.

Writing and compiling a program to be re-entrant, then specifying RENT and REUS on the link-edit makes the invocation of the routine more efficient.

Table 6 lists the exits supplied by SCLM, along with the FLMCNTRL parameter used to specify an associated user exit routine. The ***initial*** and ***verify*** exits are invoked before any real processing (change to data) occurs, and can be used to perform tasks such as verifying a user's authority to perform a given function.

The ***promote copy***, ***promote purge***, and **all** ***notify*** exits are invoked after processing has completed and can be used to perform tasks such as putting an entry into a log file, generating a report, or sending notification to a specified set of users.

All of these exit points can be used to integrate SCLM with other products as well as to enable customized processing. For example, a Verify Change Code Exit routine might be used to query an external change management product to ensure that an open problem request exists for a change being made, and that the user making the change is authorized to do so. The SCLM sample bridge to the Tivoli Service Desk/390 is an example of this type of exit routine.

The following are the available exits, along with the FLMCNTRL parameters used to specify an associated user exit routine.

Table 6. Exits and Exit Routine Specifications

Exit	Exit Routine Specification	When used
Change Code Verification Exit	VERCC	Invoked at the start of an SCLM Edit session (before the member list is presented), when a member is saved in Edit, by the Migrate and Import utilities, and by the Migrate, Import, Save, and Store services.
Verify Change Code Exit	CCVFY	Invoked after the input parameters have been verified for Edit and during SPROF processing.

Table 6. Exits and Exit Routine Specifications (continued)

Exit	Exit Routine Specification	When used
Save Change Code Exit	CCSAVE	Invoked after a member has been saved, but before the SCLM accounting information is updated for the member. This includes a Save performed as a result of using the Edit and Migrate dialogs and the Edit, Store, Save, and Migrate services.
Build Initial Exit	BLDINIT	Invoked at the very beginning of Build before any verification or processing occurs.
Build Notify Exit	BLDNTF or BLDEXT1	Invoked after Build processing completes.
Promote Initial Exit	PRMINIT	Invoked at the very beginning of Promote before any verification or processing occurs.
Promote Verify Exit	PRMVFY or PRMEXT1	Invoked at the end of the Verification phase of Promote, but before the Copy and Purge steps are processed.
Promote Copy Exit	PRMCOPY or PRMEXT2	Invoked at the end of the Copy phase of Promote processing.
Promote Purge Exit	PRMPURGE or PRMEXT3	Invoked at the end of Promote after the Verification, Copy, and Purge phase have all been completed.
Audit/Version Delete Verify Exit	AVDVFY	Invoked after the input parameters have been verified for an audit record and version, but before the actual Delete takes place.
Audit/Version Delete Notify Exit	AVDNTF	Invoked after the audit record and version have been deleted.
Delete Initial Exit	DELINIT	Invoked at the very beginning of the Delete Group before any verification or processing occurs.
Delete Verify Exit	DELVFY	Invoked after the input parameters have been verified for a Library Utility Delete or the Delete service, but before the member is actually deleted.
Delete Notify Exit	DELNTF	Invoked after the Delete has taken place for Delete Group, Library Utility Delete, or the Delete service.

Specify the Change Code Verification Routine

SCLM provides three exits you can use for verifying change codes, integrating with change management systems, or practically any other Edit, Migrate, Save, or Store processing you might want to perform. The three exits are as follows:

- The **verifyl change code** exit (CCVFY), which enables you to verify a change code, a language, a user id, or other values. The exit routine is invoked at Edit verification and SPROF processing. It is invoked during SPROF processing when either the language or the change code has changed. A blank change code is acceptable. A non-zero return code from the exit routine stops processing immediately.
- The **save change code** exit (CCSAVE), which occurs before SCLM write accounting data to the accounting data set for Edit, Migrate, Save, or Store

processing. The routine is invoked during Save. This includes Edit save processing, the Migrate Utility, and the Edit, Store, Save, and Migrate services. A blank change code is acceptable. A non-zero return code from the exit routine stops processing immediately.

- The **change code verification routine** (VERCC), which is useful for verifying change records. A non-blank change code is required. If you supply this routine to SCLM, it is used by the SCLM Editor, Migration, and Import utilities, as well as the Edit, Store, Save, Import, and Migrate services.

  When SCLM invokes the change code verification routine just prior to the edit, SCLM ignores non-zero return codes and allows the edit to begin. If the change code verification routine does not have all the information it needs, the verification routine should return a return code of 8, and the change code verification routine will be invoked again when the member is processed. When a verification routine fails during a save, you have two options:

  – You can use the CREATE edit command to make a non-SCLM controlled copy of the editing session and then use the migrate utility to bring the member back under SCLM control.
  – You can use SPROF from SCLM Edit to change/add the change code.

You can specify any or all of these routines for your project. If you specify a change code verification exit and a verify or save change code exit routine (or both), then the change code verification exit routine is invoked first. The verify or save change code exit routine is only invoked if the change code verification completes successfully. The exception is during SPROF processing where the verify change code exit routine is called without first invoking the change code verification exit routine when only the language has changed.

All three of these exit routines are invoked in the same way.

SCLM passes a string of seven parameters separated by commas to the exit routines. Register 1 contains the address of the input data. The first halfword of the input data is the length of the input string. Immediately following the halfword length is the input parameter string. The return code from the routine is the only parameter passed back. The return code is returned in Register 15. SCLM allows a member to be edited or saved only if it receives a return code of 0 from the exit routine. SCLM informs you if it detects a non-zero return code.

A project can use any combination of the parameters to determine whether or not an update should be permitted. The format and description of parameters SCLM passes to the verification routine are as follows:

Table 7. Initial and Save Change Code Exit Routine Parameters

OPTION LIST	Up to 255-character (including delimiters) parameters specified on the FLMCNTRL macro using the CCVFYOP for options to the verify change code exit routine and CCSAVOP for those passed to the save change code exit routine. Delimit this string so that the SCLM parameters that follow can be identified by the exit routine.
GROUP	The 8-character name of the group in which the member is being created or modified (capitalized, left-justified, blank-padded).
TYPE	The 8-character name of the member type being created or modified (capitalized, left-justified, blank-padded).
MEMBER	The 8-character name of the member that is being created or modified (capitalized, left-justified, blank-padded).

Table 7. Initial and Save Change Code Exit Routine Parameters (continued)

LANGUAGE	The 8-character name of the language specified for the member (capitalized, left-justified, blank-padded).
USERID	The 8-character user ID of the developer performing the modification (capitalized, left-justified, blank-padded).
AUTHCODE	The 8-character authorization code for the member (capitalized, left-justified, blank-padded).
CHANGE CODE	The 8-character change code that has been entered (capitalized, left-justified, blank-padded).

Change Code Verification Routine Example

The following example shows a simple program written in Pascal to perform minimal verification. This routine verifies that the change code of SCLM was entered. A return code of 0 indicates that the change code is valid. A return code of 8 indicates that the change code failed verification. The example assumes that the option list is empty.

The example calls the Pascal PARMS function to retrieve the string of input parameters. The example calls the Pascal RETCODE procedure to pass the verification routine return code to SCLM in register 15. The Pascal PARMS function and the RETCODE procedure follow the IBM 370 subroutine linkage convention.

```
PROGRAM EXITCCV;
(*********************************************************************)
(*  Change Code Verification User Exit                               *)
(*********************************************************************)
(*  Inputs:                                                          *)
(*    PARMS -                                                        *)
(*      option list   -  Options list (if specified on FLMCNTRL).    *)
(*      group         -  Group where the change is being made.       *)
(*      ,type         -  Type containing the member being changed.   *)
(*      ,member       -  Member being changed.                       *)
(*      ,language     -  Language of member being changed.           *)
(*      ,userid       -  User ID performing the change.              *)
(*      ,authcode     -  Authorization code of the member.           *)
(*     ,change code   -  Change code being used for the change.      *)
(*********************************************************************)
(*  Outputs:                                                         *)
(*    return_code     -  Return code in register 15.                 *)
(*                       0 -  Change code is valid.                  *)
(*                       8 -  Change code is invalid.                *)
(*********************************************************************)
(*  Process:                                                         *)
(*   This program verifies that a change code of 'SCLM' has been     *)
(*  entered.                                                         *)
(*********************************************************************)

VAR
     comma_index      :   INTEGER;
     i                :   INTEGER;
     input_data       :   STRING(320);
     return_code      :   INTEGER;
```

```
BEGIN  (* program EXITCCV *)

     (*  Initialize the variables.  *)
     input_data  :=  PARMS;
     return_code :=  0;

     (*  Parse until you get the change code. *)
     FOR
        i := 1 to 6
     DO
        BEGIN
           comma_index := INDEX(input_data,',');
           input_data  := SUBSTR(input_data,comma_index+1);
        END;  (*FOR*)

     (*  If the change code is not equal to 'SCLM', signal an error.  *)
     IF TRIM(input_data) <> 'SCLM'
     THEN
        BEGIN
           return_code := 8;
        END;  (*IF*)

     (*  Set the return code.  *)
     RETCODE(return_code);

END.               (* EXITCCV *)
```

Specify the Build and Promote User Exit Routines

Two user exits can be specified for build. SCLM invokes the initial build user exit before any build processing begins. The build notify user exit is invoked at the end of a build.

Four user exits can be specified for promote. SCLM invokes the initial promote user exit before any promote processing begins. SCLM invokes the promote verification user exit, the promote copy user exit, and the promote purge user exit routines at the end of the promote verification, copy, and purge phases, respectively.

Build and promote user exits are defined to the project definition using the following parameters on the FLMCNTRL macro.

Initial Build User Exit	BLDINIT
Build Notify User Exit	BLDNTF or BLDEXT1 (old format)
Initial Promote User Exit	PRMINIT
Promote Verify User Exit	PRMVFY or PRMEXT1 (old format)
Promote Copy User Exit	PRMCOPY or PRMEXT2 (old format)
Promote Purge User Exit	PRMPRURGE or PRMEXT3 (old format)

Build and Promote User Exit Routine Requirements

If you specify a user exit option parameter, SCLM passes it to the user exit routine, followed by a string of 10 parameters separated by commas. The address of this input data is contained at the address stored in register 1. The first halfword of the input data is the number of characters comprising the input data string. Immediately following this halfword length is the input parameter string itself.

The user exit routine must pass back a return code value to SCLM in register 15. A return code of zero is considered to be successful and processing continues. A non-zero return code from the user exit routine causes build or promote to end with a return code 8. Whether or not processing continues after the user exit depends on the return code value passed back by the user exit routine and the exit routine being invoked. Non-zero return code values from user exit routines are handled in the following ways:

- Both the build notify user exit (BLDNTF) and the promote purge phase user exit (PRMPURGE) can return any value as processing has already been completed at the time the exit is invoked.
- Any non-zero value returned by the initial build user exit (BLDINIT) or the initial promote user exit (PRMINIT) causes processing to stop.
- The processing that occurs after the promote verification phase user exit (PRMVFY) has been invoked depends on the promote mode in effect. In conditional mode, a return code greater than 4 causes promote processing to stop. In unconditional mode, any return code other than 20 allows promote processing to continue.
- The processing that occurs after the promote copy phase user exit (PRMCOPY) has been invoked depends only on the return code value returned. Any return code other than 20 allows normal promote processing to continue.

The format and description of the parameters passed from SCLM through all user exits are:

Table 8. User Exit Parameters

OPTION LIST	Up to 255 characters (including delimiters) (blank-padding is not performed for this parameter). Parameter is specified in the FLMCNTRL macro using macro parameters BLDINIOP, BLDNTFOP, PRMINIOP, PRMVFYOP, PRMCPYOP, and PRMPRGOP. Delimit this string so that the SCLM parameters that follow can be identified by the user exit routine.
'xxxxxxxx'	An 8-character literal value indicating the exit type (capitalized, left-justified, blank-padded). Valid types are: **BINITIAL** Build Initial (BLDINIT) **BUILD** Build Notify (BLDNTF) **PINITIAL** Promote Initial (PRMINIT) **PVERIFY** Promote Verify (PRMVFY) **PCOPY** Promote Copy (PRMCOPY) **PPURGE** Promote Purge (PRMPURGE).
PROJECT	The 8-character name of the project (capitalized, left-justified, blank-padded).
LIBDEF	The 8-character name of the project definition (capitalized, left-justified, blank-padded).
USERID	The 8-character value of the user's logon ID (capitalized, left-justified, blank-padded).

Table 8. User Exit Parameters (continued)

FROM GROUP	The 8-character name of the group (capitalized, left-justified, blank-padded). The group is the "from group" for the promote and the "build group" for the build.
TYPE	The 8-character name of the type (capitalized, left-justified, blank-padded).
MEMBER	The 8-character name of the member (capitalized, left-justified, blank-padded).
SCOPE	The 8-character name of the scope (capitalized, left-justified, blank-padded). Valid scopes are as follows: **Build scope** Limited, normal, subunit, extended. **Promote scope** Normal, subunit, extended.
MODE	The 13-character name of the mode (capitalized, left-justified, blank-padded). Valid modes are as follows: **Build mode** Forced, conditional, unconditional, and report only. **Promote mode** Conditional, unconditional, and report.
TO GROUP	The 8-character name of the group (capitalized, left-justified, blank-padded). The group is the "to-group" for the promote exit routines. This parameter is blank for the build exit routine.

Build allocates the following ddnames for internal use:
- BLDEXIT
- BLDLIST
- BLDMSGS
- BLDREPT

Promote allocates the following ddnames for internal use:
- COPYERR
- PROMEXIT
- PROMMSGS
- PROMREPT

Use of these names in user exit routines can cause conflicts.

At the end of an exit routine, free only those ddnames explicitly allocated by the exit routine.

Build and Promote User Exit Output Data Sets

If you specify a build notify or promote verify, copy, or purge user exit routine, SCLM generates a sequential data set containing a record for each member changed or verified by build or promote. This data set is not generated for the initial build or initial promote user exits. Verified members are those eligible for promotion during the promote verification phase. Changed members for build are those members produced due to translator calls. Changed members for promote are those members copied or purged. SCLM puts new data in the data set for the invocation of each exit. User exit routines can use the output data set when called, but the data set is rewritten for later exits and is deleted when the SCLM processor ends.

The data definition names (ddnames) for build and promote exit output data sets are BLDEXIT and PROMEXIT respectively. The attributes of the output data sets are the same for all the exit routines:

RECFM	FB
BLOCK SIZE	3200
LRECL	160

The format of the data set is the same for every exit. The data set contains three 8-character fields and one 16-character status field. A blank separates all fields. The following list defines the fields generated for every exit routine:

Table 9. User Exit Output Data Set Format

GROUP	Specifies the 8-character name of the group beginning in column 1.
TYPE	Specifies the 8-character name of the type beginning in column 10.
MEMBER	Specifies the 8-character name of the member beginning in column 19.
STATUS	Specifies the status beginning in column 28. **BUILT/DELETED** Indicates if the member was built or if it was an obsolete output that was deleted. This field is written by BLDNTF. **PROMOTABLE/NOT PROMOTABLE** Indicates if the member is eligible for promotion. This field is written by PRMVFY. **COPY SUCCESSFUL/COPY FAILED** Indicates if the member was copied. This field is written by PRMCOPY. **PURGE SUCCESSFUL/PURGE FAILED** Indicates if the member was purged. This field is written by PRMPURGE.

The following example shows build user exit output:

```
USER1   TYPE1   MEMBER1  BUILT
USER1   TYPE    MEM1     BUILT
USER1   TYPE2   MEMBER5  BUILT
```

Specify the Audit Version Delete User Exit Routine

There are two audit version delete exit points in SCLM—audit version delete verify and audit version delete notify. These exits are invoked when an audit record or an audit record and its associated version are deleted using either the SCLM Audit and Version Utility, Version Selection dialog (ISPF Option 10.3.8), or the VERDEL service interface.

The use of the audit version delete exits is optional. SCLM does not provide the user exit routines to be invoked by these exit points.

The audit version delete verify exit is invoked after the initial verification of the inputs is done, but before the the actual deletion of the audit and version data takes place.

The audit version notify exit is invoked after the deletion of the audit and version data has been attempted (in the case of a failure) or performed (when the deletion is successful).

These exits can be used to perform logging functions or additional verification, send notifications or coordinate processing with non-SCLM tools.

Audit Version Delete User Exit Routine Requirements

If you specify a user exit option parameter, SCLM passes it to the user exit routine, followed by a string of 11 parameters separated by commas. The address of this input data is contained at the address stored in register 1. The first halfword of the input data is the number of characters comprising the input data string. Immediately following this halfword length is the input parameter string itself.

The user exit routine must pass back a return code value to SCLM in register 15. A return code of zero is considered to be successful and processing continues. A non-zero return code from the first audit version delete exit verify routine (AVDVFY) causes processing to end and the requested audit and version information is not deleted. The second audit version delete notify user exit routine (AVDNTF) can pass back any value in register 15 and processing continues because the delete has already been performed.

The format and description of the parameters passed from SCLM to the audit version delete user exits are:

Table 10. User Exit Parameters

OPTION LIST	Up to 255 characters (including delimiters) (blank-padding is not performed for this parameter). Parameter is specified in the FLMCNTRL macro using macro parameters AVDVFYOP and AVDNTFOP. Delimit this string so that the SCLM parameters that follow can be identified by the user exit routine.
'xxxxxxxx'	An 8-character literal value indicating the exit type (capitalized, left-justified, blank-padded). Valid types are: **ADVERIFY** Audit Version Delete Verify **ADNOTIFY** Audit Version Delete Notify
PROJECT	The 8-character name of the project (capitalized, left-justified, blank-padded).
LIBDEF	The 8-character name of the project definition (capitalized, left-justified, blank-padded).
USERID	The 8-character value of the user's logon ID (capitalized, left-justified, blank-padded).
GROUP	The 8-character name of the group (capitalized, left-justified, blank-padded) for the audit record or audit record and version.
TYPE	The 8-character name of the type (capitalized, left-justified, blank-padded) for the audit record or audit record and version.
MEMBER	The 8-character name of the member (capitalized, left-justified, blank-padded)for the audit record or audit record and version.
DATE	The 10-character NLS formatted date with 4-character year for the audit record or audit record and version.

Table 10. User Exit Parameters (continued)

TIME	The 11-character time for the audit record or audit record and version. The format for the time is HH:MM:SS.hh or HH:MM:SS,hh. In the format, HH is the hour from a 24-hour clock, MM is the minutes, SS is the seconds, and hh is the hundredths of a second.
VERSION MEMBER NAME	The 8-character version member name (capitalized, left-justified, blank-padded) indicates whether or not the requested audit record has an associated version. When an associated version exists, this value is the same as the member name. This value is blank when the requested audit record does not have an associated version.

Specify the Delete User Exit Routine

There are three delete exit points in SCLM—an initial delete exit, a delete verify exit, and a delete notify exit. The initial delete exit is invoked only for the Delete Group service or dialog (ISPF Option 10.3.9). It is invoked during initialization and before any processing is done. The ″group″ (for Delete Group service **only**), ″type″, and ″member name″ values can contain pattern symbols. The purpose of this exit is to enable verification for a certain level, for example, to insure that a suer is authorized to use Delete Group.

The delete verify exit is invoked for Library Utility Delete (ISPF Option 10.3.1) and the Delete service. It is invoked after the input parameters have been verified, but before any processing is performed.

The delete notify exit is invoked for Library Utility Delete, the Delete service, and the Delete Group dialog and service. The exit is invoked after the delete has been attempted (in the case of failure) or performed (when the deletion succeeds).

Delete User Exit Routine Requirements

If you specify a user exit option parameter, SCLM passes it to the user exit routine, followed by a string of 10 parameters separated by commas. The address of this input data is contained at the address stored in register 1. The first halfword of the input data is the number of characters comprising the input data string. Immediately following this halfword length is the input parameter string itself.

The user exit routine must pass back a return code value to SCLM in register 15. A return code of zero is considered to be successful and processing continues. For the delete verify and delete notify exit routines, any return code other than zero indicates failure and processing ends. In the case of the delete notify exit, the delete has already been performed.

The format and description of the parameters passed from SCLM to the delete user exits are:

Table 11. User Exit Parameters

OPTION LIST	Up to 255 characters (including delimiters) (blank-padding is not performed for this parameter). Parameter is specified in the FLMCNTRL macro using macro parameters DELINTOP, DELVFYOP, and DELNTFOP. Delimit this string so that the SCLM parameters that follow can be identified by the user exit routine.

Table 11. User Exit Parameters (continued)

'xxxxxxxx'	An 8-character literal value indicating the exit type (capitalized, left-justified, blank-padded). Valid types are: **DGINIT** Initial Delete **DVERIFY** Verify delete exit invoked for the Delete service or Library Utility Delete **DNOTIFY** Notify delete exit invoked for the Delete service or Library Utility Delete **DGNOTIFY** Notify delete exit invoked for the Delete Group service or dialog
PROJECT	The 8-character name of the project (capitalized, left-justified, blank-padded).
LIBDEF	The 8-character name of the project definition (capitalized, left-justified, blank-padded).
USERID	The 8-character value of the user's logon ID (capitalized, left-justified, blank-padded).
GROUP	The 17-character name of the group (capitalized, left-justified, blank-padded).
TYPE	The 17-character name of the type (capitalized, left-justified, blank-padded).
MEMBER	The 17-character name of the member (capitalized, left-justified, blank-padded).
FLAG	The 8-character delete flag (capitalized, left-justified, blank-padded). Valid delete flags are ACCT, BMAP, TEXT, and OUTPUT. This value is always TEXT for a Library Utility Delete. OUTPUT is valid only for Delete Group.
MODE	The 8-character name of the mode (capitalized, left-justified, blank-padded). Valid modes are EXECUTE and REPORT. This value is valid only for Delete Group. A blank value is passed for the Delete service and Library Utility Delete.

Delete Group allocates the following ddnames for internal use:
- DGEXIT
- DGLIST
- DGMSGS
- DGREPT

Use of these names in a delete user exit routine can cause conflicts. At the end of an exit routine, free only those ddnames explicitly allocated by the exit routine.

Delete User Exit Output Data Set

When a Delete Group is performed and you specify a delete notify user exit routine, SCLM generates a sequential data set containing a record for each member for which a delete is requested. SCLM puts new data in the data set for the invocation of each exit. The delete notify user exit routine can use the output data set when called, but the data set is rewritten for later exits and is deleted when the SCLM processor ends.

The default data definition name (ddname) for the delete exit output data set is DGEXIT. The attributes of the output data set are:

RECFM	FB
BLOCK SIZE	3200
LRECL	160

The data set contains the following fields. A blank separates all fields.

Table 12. User Exit Output Data Set Format

DATA TYPE	Specifies the 8-character name of the type of data. This is equivalent to the section headings in the Delete Group report. Valid types are MEMBER or BUILDMAP. MEMBER is used when an accounting record or an accounting record and PDS member are deleted.
GROUP	Specifies the 8-character name of the group beginning in column 9.
TYPE	Specifies the 8-character name of the type beginning in column 18.
MEMBER	Specifies the 8-character name of the member beginning in column 27.
STATUS	Specifies the 19-character status beginning in column 36. Valid values are: **DELETE SUCCESSFUL** Indicates the requested data was successfully deleted. **DELETE FAILED** Indicates an error occurred and the delete failed. **DELETE WARNING** Indicates a warning was issued. The requested data either did not exist or was successfully deleted. **NOT ATTEMPTED** Indicates that Delete Group was done in report mode. The delete was not attempted.
OUTPUT	Specifies the 1-character OUTPUT indicator beginning in column 56. If the requested data was a build output, then this column contains an asterisk (*).

The following example shows the delete user exit output that is generated when a Delete Group is requested:

```
MEMBER  USER1   TYPE1   MEMBER1  PASSED    *
```

User Exit Routine Example

An example program written in Pascal to perform minimal user exit activity follows. This routine writes the passed parameters to the data set PROMOUT1, copies the user exit output data set contents to the PROMOUT1 data set, and passes a return code of zero (0) to SCLM.

The program calls the Pascal PARMS function to retrieve the string of input parameters. It calls the Pascal RETCODE procedure to pass the verification routine return code to SCLM in register 15. The Pascal PARMS function and RETCODE procedure assume the IBM S/370 subroutine linkage convention.

```
PROGRAM EXIT001;
(*********************************************************************)
(*   Promote User Exit                                               *)
(*********************************************************************)
(*  Inputs:                                                          *)
(*    PARMS -                                                        *)
(*     option list    -  Options specified in FLMCNTRL macro.        *)
(*     exit type      -  PVERIFY, PCOPY, or PPURGE literal.          *)
(*     ,project       -  Name of the project.                        *)
(*     ,libdef        -  Name of the project definition.             *)
(*     ,userid        -  User ID performing the promote.             *)
(*     ,group         -  Group the member is being promoted from.    *)
(*     ,type          -  Type the member is being promoted from.     *)
(*     ,member        -  The member being promoted.                  *)
(*     ,scope         -  NORMAL, SUBUNIT, or EXTENDED literal.       *)
(*     ,mode          -  CONDITIONAL, UNCONDITIONAL, or REPORT.      *)
(*     ,group         -  Group the member is being promoted to.      *)
(*                                                                   *)
(*    PROMEXIT        -  Promote user exit output data set.          *)
(*                                                                   *)
(*********************************************************************)
(*  Output:                                                          *)
(*    PROMOUT1        -  Output text file contains promote log       *)
(*                       info for this promote phase.                *)
(*                                                                   *)
(*    return_code     -  Return code in register 15.                 *)
(*                       0 - Successful.                             *)
(*********************************************************************)
(*  Process:                                                         *)
(*   This program saves the contents of the PROMEXIT file.           *)
(*********************************************************************)
VAR
     out_file         :   TEXT;
     in_file          :   TEXT;
     parm_string      :   STRING(100);
     line             :   STRING(52);

BEGIN  (* program EXIT001 *)

   (*  Open the file for write *)
   REWRITE(out_file,'DDNAME=PROMOUT1');

   (*  Open the file for read  *)
   RESET(in_file,'DDNAME=PROMEXIT');

   (*  Retrieve input parameters and write them to the output file *)
   parm_string := PARMS;


   WRITELN(out_file,'User exit 1 entered.');
   WRITELN(out_file,'Parms=',TRIM(parm_string));
   WHILE NOT EOF(in_file) DO
      BEGIN
         READLN(in_file, line);
         WRITELN(out_file,line);
      END;

   (*  Close both files and set the program return code  *)
   CLOSE(out_file);
   CLOSE(in_file);
   RETCODE(0);
END.
```

Chapter 3. Additional Project Manager Tasks

In addition to the tasks described in Chapter 1. Defining the Project Environment, project managers can perform other tasks associated with defining and maintaining SCLM projects. This chapter describes other areas of responsibility in which project managers are involved. These include:

- Splitting VSAM data sets
- Backing up and recovering the project environment
- Synchronizing and maintaining accounting data sets
- Modifying the Delete Group dialog interface

Splitting Project VSAM Data Sets

You might need to split the project VSAM data sets into multiple data sets because of security requirements, data set size, performance or changes in the way the project is being developed. By using multiple VSAM data sets in conjunction with flexible data set naming, cross-project support (for example, sharing common code) can be achieved.

The following steps make up the basic process for splitting project VSAM data sets:

1. Decide how you want to split the data sets. SCLM allows the VSAM data sets to be split on group boundaries.
2. Back up the data from the existing VSAM data sets for those groups using the new VSAM data sets. There are two ways to back up the data:
   a. You can use the SCLM export utility to export the contents of each group to the new data set. Because the Import utility deletes the contents of the export data set upon a successful completion of the import, you should make a backup of the export VSAM data sets using the IDCAMS reproduction utility (REPRO). By using this method, you do not need to update the contents of the PDS data sets. You only need to copy members from those groups that will be using the new VSAM data set. This method does not copy the audit records.

      Note: Using the REPRO function of the IDCAMS utility, you can split the audit data base at any point to create any number of smaller audit data bases. In order to use these smaller audit data bases, create alternate project definitions that specify the newly created audit data bases.

   b. You can use the IDCAMS REPRO utility to make a copy of each of the VSAM data sets used by the project. This method has the advantage of creating a backup of the project VSAM data sets. All records are copied to the new VSAM data set. While having the copies for all groups in the new VSAM data set is not a problem for SCLM, it does increase the size of the data set. These records can be deleted by setting up an alternate project definition that points only to the new VSAM data set and using the Delete Group service to delete the groups that are not needed in that data set.
3. Make a backup copy of the project definition. This backup copy is needed to delete the data from the original VSAM data sets.
4. Update the project definition to add an FLMALTC macro for the new data sets and ALTC parameters on the groups that will be using those data sets.

5. Allocate the new VSAM data sets.
6. Assemble the new project definition.
7. Restore the data for the new VSAM data set from backup. How you do this depends on what method you used to back up the data:
   a. If you used the Export utility, use the Import utility to restore the data to the new VSAM data sets.
   b. If you used the IDCAMS REPRO utility, use the REPRO utility to restore the data. You can do this before assembling the new project definition because it does not use any SCLM services.
8. Test the new project definition. Here are some suggestions for testing the new project definition:
   - Edit a member at the modified group. Create a new member, and also edit an existing member.
   - Run a build from the modified group.
   - Promote from the modified group.
9. Delete data from the existing VSAM data set for those groups that reference the new VSAM data set. You can do this by using a backup copy of the old project definition and the Delete Group utility for each group that was moved.

   If you used the method of promoting to a new group, this step is not needed.

Backing Up and Recovering the Project Environment

The important point in backing up and recovering the project environment is that all the data remains synchronized. The project partitioned data sets contain related data, and the control data sets contain the control information for the PDS members. Thus, backing up and restoring the project environment means that the project partitioned data sets and the control data sets must be backed up and restored together.

The recommended procedure for backing up the project environment is to run a background job when no one is working within the hierarchy. You should determine how often to run this job. Remember that the topmost group of the hierarchy (the production group) usually contains most of the software and is usually frozen. You should back up the topmost groups whenever new data is promoted into the topmost groups. The lower groups in the hierarchy are subject to change much more often, and the code in the development groups usually changes daily. Perform backups for the lower groups based on your project's requirements. Again, remember that you must back up an entire group as a unit; this includes the project partitioned data sets and the control data sets.

Be careful when recovering a project environment. When you restore a group, it returns to the version that was in effect when you backed it up. This change can affect code below the restored group. Also the control data sets reflect the status of the group when it was backed up.

Synchronizing Accounting Data Sets

The SCLM FLMCNTRL and FLMALTC macros allow you to select dual accounting data sets to be maintained using the ACCT and ACCT2 parameters. If a nonrecoverable problem occurs with one of the primary accounting data sets, use the following JCL to restore the primary accounting data set.

```
//jobname  JOB (wkpkg,dpt,bin),'name'
//*************************************************************
//*                                                           *
//*  JCL TO RESTORE THE PRIMARY ACCOUNTING DATA SET FROM THE  *
//*  SECONDARY ACCOUNTING DATA SET.                           *
//*                                                           *
//*  SPECIFY THE UNCORRUPTED DATA SET AS YOUR INPUT DATA SET  *
//*                                                           *
//*************************************************************
//STEP1    EXEC PGM=IDCAMS
//INPUT    DD DISP=OLD,DSN=PROJ1.ACCOUNT2.FILE
//OUTPUT   DD DISP=OLD,DSN=PROJ1.ACCOUNT.FILE
//SYSPRINT DD SYSOUT=H
//SYSIN    DD *
 REPRO INFILE(INPUT) OUTFILE(OUTPUT)
/*
//
```

You can also use this JCL to initialize a backup data set for a project that is currently running under SCLM. If problems occur with the backup data set, SCLM issues warning messages. You must restore the backup data set when problems occur.

Maintaining Accounting Data Sets

When groups or types are removed from the project definition, some accounting information from those groups or types can remain in the VSAM data sets for that project. In order to avoid having records that are no longer useful in the VSAM data sets, you should use the DELGROUP service to remove the VSAM records for any groups or types that are being removed from the project definition. This step should be performed before the groups and types are removed from the project definition.

If groups or types have been previously removed from the project definition, you can create an alternate project definition that includes a definition for the removed groups and types. This project definition can be used with the DELGROUP service to delete any remaining VSAM records.

Modifying the Delete Group Dialog Interface

Given the power of Delete Group, there are some restrictions in the dialog interface. Explanations for the restrictions and instructions for modifying the dialog to remove such restrictions follow.

The **Group** field is restricted to disallow patterns. To remove this restriction:

1. Edit the panel FLMDDG#P. It is recommended that you update the DTL version instead of the generated panel to avoid losing the changes if the panel is regenerated. Refer to *Dialog Tag Language (DTL) Guide and Reference* for more information.
2. Replace the line:

   ```
   <dtafld datavar=DGLEVEL usage=both
          entwidth=8 pmtwidth=12 >&lib_prompt;
   ```

   with the lines:

   ```
   <dtafld datavar=DGLEVEL usage=both
          deswidth=41 entwidth=9 pmtwidth=12 >&lib_prompt;
     <dtafldd>(Pattern can be used)
   ```

or with the lines:

```
<dtafld datavar=DGLEVEL usage=both
       deswidth=41 entwidth=17 pmtwidth=12 >&lib_prompt;
  <dtafldd>(Pattern can be used)
```

depending upon how you resolve the next restriction. They should be consistent if patterns are allowed.

3. Edit the imbed FLMZDG#P, and replace the line:

   ```
   VER(&DGLEVEL,NB,NAME)
   ```

   with the line:

   ```
   VER(&DGLEVEL,NONBLANK)
   ```

Type and **Member** fields are restricted to 9 characters; FLMCMD and FLMLNK allow up to 17 characters. To remove this restriction:

1. Edit the panel FLMDDG#P. It is recommended that you update the DTL version instead of the generated panel to avoid losing the changes if the panel is regenerated. Refer to *Dialog Tag Language (DTL) Guide and Reference* for more information.
2. Replace the lines:

   ```
   <dtacol entwidth=8 pmtwidth=12
           deswidth=49 fldspace=11 >
   ```

   with the lines:

   ```
   <dtacol entwidth=17 pmtwidth=12
           deswidth=41 fldspace=11 >
   ```

The Delete mode always defaults to Report when the panel appears. To remove this restriction, remove the following lines from the FLMZDG#P panel imbed:

```
&DMODE = 'REPORT'
&DMODEV = '2'
```

Chapter 4. Converting Projects to SCLM

To convert an existing project to an SCLM-controlled project, bring the project groups under control one at a time beginning with the top layer of the hierarchy, which is the production (frozen) group, and work downward. Most projects to be converted already exist in some kind of logical hierarchy. If all production source code resides in one logical place and code under development resides elsewhere, you have at least a two-layer hierarchy. Before migration can begin, you must place the source code to be converted into partitioned data sets.

There are many advantages to using the preceding method. First, you can bring a project under SCLM control in discrete steps, over a period of time. Second, SCLM can locate integrity problems in the existing hierarchy and fix them systematically during the conversion process. Third, SCLM performs the conversion using the same tools that developers use in the normal development process. Thus, you ensure consistency within the hierarchy, and you become familiar with SCLM. Finally, from the conversion process, you receive an indication of the performance that you can expect of SCLM during the development process.

Prerequisites for Existing Hierarchies

The best time for you to begin the conversion process is when the components to be controlled are concentrated in a small number of groups—immediately following a software release, for example. The following actions help you prepare a hierarchy for the conversion process.

- Create the project definition to be used with the converted hierarchy. See Chapter 1. Defining the Project Environment, for details.
- Verify that all partitioned data sets to be controlled are available online. If the data is not in partitioned data sets, allocate partitioned data sets by following “Step 5: Allocate the Project Partitioned Data Sets” on page 13, and copy data from the existing data sets to the partitioned data sets.
- Delete all unnecessary data from the libraries being converted.
- If you intend to use non-key groups in the converted hierarchy, ensure that they do not contain any data prior to conversion.

Create Alternate Project Definitions

You need to create several alternate project definitions to complete the conversion process. Because the SCLM migration utility can only run against development libraries, which are in the lowest layer of the hierarchy, you need an alternate project definition for each layer of the proposed hierarchy. The first alternate project definition you use defines only the topmost group. That group becomes a development group. The second project definition defines the topmost group and those groups that promote into it, and so on. You do not need to define non-key groups in the alternate project definitions you use for the conversion process because they should not contain any members.

Create Architecture Definitions for the Project

Although you can perform the conversion process without architecture definitions, their creation can greatly simplify the conversion process as well as support future development needs. Define a set of architecture members first for the code in the topmost group of the hierarchy. These architecture members must reference only members that are present in the topmost group because only those members are visible during the first group conversion.

To determine which architecture members you need, do the following:

1. Determine whether all the build translators can use the default translator options in the language definitions. If they can, you do not need compilation control architecture members.
2. Determine the contents of every load module to be controlled. The IEHLIST utility prints the names of all objects in a load module.
3. Produce a linkage edit control architecture member for every load module, and reference each object (actually compilable source members) with an INCLD statement. Use the INCL statement in place of INCLD to reference compilation control architecture members if they are created above.
4. Produce high-level architecture members as needed to control any non-translatable data or data that is not included in load modules.
5. Produce a high-level architecture member and reference each linkage edit control architecture member and high-level architecture member defined above with an INCL statement.

The high-level architecture member created in Step 5 now defines, through its dependencies, the entire application architecture.

After you create the architecture members for the topmost group, you might need to add modifications in the lower groups of the hierarchy. Members that were added during the development process and were not moved to the topmost group may require additional architecture members. You must introduce architecture modifications in the group requiring the change. This action allows the architecture for the hierarchy to match the members controlled in the hierarchy. See Part One of this book for a description of the process and syntax for defining architecture members.

Register Existing PDS Members with SCLM

Editable members and noneditable members are processed in separate and unique ways by SCLM.

Editable members, such as source members, are not created by the SCLM build function. Editable members must be registered with SCLM through the migration utility. Both the language associated with the member and a change code (only if you have a change code verification routine) are required as input to the migration utility. TEXT can be used as the language of members that do not need to be compiled, assembled, or processed, such as panels and messages. Call the migration utility for each library containing editable members.

The SCLM Build function creates noneditable members. Object code, listings, and load modules are examples of noneditable members. The SCLM build function must be called to create all of the noneditable members to be tracked within the hierarchy. If all of the customization related to language translators is complete and has been tested, run the build processor in the unconditional mode using the

topmost architecture member for your application. This unconditional build will identify all build errors that exist. If errors are anticipated and the application is large, use architecture members with smaller scopes. For example, use an LEC architecture member rather than an HL. Using the conditional mode of the build processor causes processing to stop when a member containing an error is encountered.

The normal process is to migrate source members into SCLM and then generate the outputs using the SCLM Build function. There may be occasions, however, where you would like to use SCLM to manage object and load modules for which the source code no longer exists. There are two ways of doing this.

The first method uses a 'dummy' language definition with an FLMLANGL macro, but no FLMTRNSL macros. An example of this is provided as member FLM@OBJ in the ISP.SISPMACS data set shipped with SCLM. This language definition allows you to migrate object and load modules into SCLM as editable members in the same manner that source modules are introduced.

Note: Special care must be taken when using versioning in a project that has stored object and load modules in this manner. SCLM will consider the members to be editable and will allow versioning to occur if specified. This may cause errors in SCLM version processing. The second method is a better choice when versioning is being used in the project.

The second method involves migrating the object and load modules into a temporary type and then using the SCLM Build function to copy them to the target type. The SCLM build process will mark the copied object and load modules as non-editable. This solution is a better choice for projects with versioning in use. Member FLM@COPY in the ISP.SISPMACS data set shipped with SCLM can be used to store object modules into SCLM in this manner. It can be modified for use with load modules. This language definition will migrate the members into a temporary type as editable members. SCLM will allow the migrate because, like the FLM@OBJ language definition, there is no FLMTRNSL macro with FUNCTN=PARSE and therefore no parser will be invoked. The FLMTRNSL macro for the Build function calls IEBGENER to copy the modules from one SCLM type to the other as non-editable outputs.

Introducing Fixes to the Converted Hierarchy

During the conversion process, SCLM might discover integrity errors existing in the current development hierarchy. If it encounters these errors in the topmost group of the hierarchy, the errors have an effect on the rest of the conversion process. You can encounter two kinds of errors:

- Dependency errors for editable members. Errors can be caused when an included member or macro cannot be found within the hierarchy. If you want the missing member tracked in the hierarchy, you must copy the correct version of the included member to the group being converted. If you do not want the missing member tracked in the hierarchy, define it to SCLM using the FLMSYSLB macro and the FLMCPYLB macro in the language definition of the member.
- Compile errors, or any similar translator errors in any group, located during the build process. The errors must be corrected before proceeding with the conversion. Use the listings produced by build to locate and correct the errors. After making the correction rebuild the members that contained the errors.

Chapter 5. Language Definition Considerations

SCLM can be tailored to support languages other than those listed in the examples provided with the product. By creating a *language definition* as part of the project definition, you specify to SCLM the languages that will be used for the project. Language definitions provide SCLM with language-specific control information such as the language name and the definition of the language translators.

The language definition describes language-specific processing in two ways.

From a data-flow perspective, the language definition specifies all data sets used as input to or output from various SCLM processes such as Parse, Build, Promote, and Delete.

From a procedural perspective, the language definition specifies the translators (for example, parsers or compilers) that are invoked to process your SCLM-controlled data. The order in which those translators are invoked and the options to be passed to the translators are defined in the language definition.

You must provide SCLM a language definition for each language (PL/I, COBOL, Link Edit, and so on) that you want SCLM to support. In most cases, you can make minor modifications to sample SCLM language definitions provided with the ISPF product.

A language definition consists of a hierarchy of the following definitions:
- System library definitions
- Language identifier definition
- Translator definitions
- Allocation definitions
- Copy library definitions
- Include set definitions.

Because a macro exists for each of these definitions and because each macro accepts a number of different parameters, you can specify a large variety of language definitions. The language definitions provided with the product are examples that can serve as a reference in the construction of language definitions for a specific application and environment.

To determine what modifications you can make to the language definition, become familiar with the parameters of the language definition macros as documented in *ISPF Software Configuration and Library Manager (SCLM) Reference*. Typically, if you want to write a new language definition, you should copy an old language definition and then modify it to meet your specific needs.

In the remainder of this chapter, several language definitions are examined more closely in order to describe some of the implementations of language definitions. Topics discussed in this chapter include:
- Using multiple translators in a language definition
- Invoking user-defined parsers
- Processing conditionally saved components
- Specifying the location of included members
- Tracking dynamic includes
- Using input list translators.

Using Multiple Translators in a Language Definition

You can define one or more translators for a language using the FLMTRNSL macro. The parameters of the FLMTRNSL macro define all the attributes needed to call a given translator. The FLMTRNSL FUNCTN parameter defines the function or purpose for which a translator is called. SCLM uses translators for the following functions:

- Parsing source code to determine statistics and dependency information. SCLM calls these translators when a member is saved in the editor or migrated (dialog function or MIGRATE service) or saved with the SAVE service.
- Translating one form of code into another. Some examples of code translations are:
  - Source code to object code and listings
  - Script input to a formatted document
  - Object code to load modules.

  SCLM calls these translators during the build process.
- Verifying data. A verify translator can be used to perform validation in addition to the default SCLM validation. The verify translator is invoked prior to the translation step (such as compiling and linking) of build, and prior to the copy phase of promote.
- Copying data. SCLM calls these translators during the promote process. The data can be either PDS members controlled directly by SCLM or non-PDS data that includes an intermediate form of compilation units and external data identified to SCLM via a build translator.
- Purging data. SCLM calls these translators during the promote process. The data can be either PDS members controlled directly by SCLM or non-PDS data that includes an intermediate form of compilation units and external data identified to SCLM via a build translator.

The translators required for a language are language-specific. Some languages require parse and build translators while others need parse, build, copy, and purge translators.

Most SCLM-supplied example language definitions have two translators defined. The first identifies the parser to be invoked, and the second identifies the translator to be invoked during a build. Language definitions can be created for the invocation of one or more translators during the parse, build, copy, verify, or purge functions. For each of these functions, the translators are invoked in the order in which they appear in the language definition. Within a function in the language definition, a translator can pass data on to the next translator invoked by that function within the language definition. This capability allows you to customize the SCLM product for unique processing requirements in your project.

When connecting SCLM translators together in a language definition, make sure that they are ordered so that they will execute in the correct sequence. If used for build, you should order the preprocessing and compile steps just as you would in a CLIST or JCL.

If multiple-step language definitions specify more than one translator to be invoked during a build, make sure the DDNAMEs for outputs to be copied into the project hierarchy are unique. If the same DDNAME is used, only the outputs from the last translator will be copied to the hierarchy.

Note: If a translator is defined with an EXLIBID parameter (that is, it is to be used by an external library), SCLM will ignore this translator and not invoke it. SCLM will behave as if this translator does not exist.

Figure 19 shows a language definition that uses multiple translators. The DB2 preprocessor (DSNHPC) creates a COBOL source data set using the SYSCIN ddname. The next translator, the COBOL II compiler IGYCRCTL, reads in the SYSCIN data set. Notice that the receiving translator defines SYSCIN as IOTYPE=U, meaning that SYSCIN has already been allocated in a previous translator step.

```
*********************************************************************
* COBOL II WITH DB2 PREPROCESSOR  - LANGUAGE DEFINITION FOR SCLM
*
* DB2 OUTPUT IS PASSED VIA THE 'SYSCIN' DD ALLOCATION TO COBOL II.
* POINT THE FLMSYSLB MACRO(S) AT ALL 'STATIC' COPY DATASETS.
* CUSTOMIZE THE 'OPTIONS' AND 'GOODRC' FIELDS TO YOUR STANDARDS.
* ADD THE 'DSNAME' FIELD IF THE TRANSLATOR IS IN A PRIVATE LIBRARY.
* WHEN A NEW TRANSLATOR VERSION REQUIRES TOTAL RECOMPILATION FOR THIS
* LANGUAGE, THE 'VERSION' FIELD ON FLMLANGL SHOULD BE CHANGED.
*********************************************************************
* CHANGE ACTIVITY:                                                  *
*                                                                   *
*********************************************************************
*
         FLMLANGL    LANG=DB2COB2,ALCSYSLB=Y
*
* PARSER TRANSLATOR
*
          FLMTRNSL   CALLNAM='SCLM COBOL PARSE',                     C
               FUNCTN=PARSE,                                         C
               COMPILE=FLMLPCBL,                                     C
               PORDER=1,                                             C
               OPTIONS=(@@FLMLIS,@@FLMSTP,@@FLMSIZ,)
*          (* SOURCE      *)
          FLMALLOC  IOTYPE=A,DDNAME=SOURCE
           FLMCPYLB @@FLMDSN(@@FLMMBR)
*
* BUILD TRANSLATORS
*
*         --DB2 PREPROCESSOR INTERFACE--
          FLMTRNSL   CALLNAM='DB2 PREPROCESS',                       C
               FUNCTN=BUILD,                                         C
               COMPILE=DSNHPC,                                       C
               VERSION=1.0,                                          C
               GOODRC=4,                                             C
               PORDER=3,                                             C
               OPTIONS=(HOST(COB2))
*  1       -- N/A --
          FLMALLOC  IOTYPE=N
*  2       -- N/A --
          FLMALLOC  IOTYPE=N
*  3       -- N/A --
          FLMALLOC  IOTYPE=N
*  4       -- SYSLIB --
          FLMALLOC  IOTYPE=I,KEYREF=SINC
*  5       -- SYSIN --
          FLMALLOC  IOTYPE=S,KEYREF=SINC,RECFM=FB,LRECL=80,          C
              RECNUM=2000
*  6       -- SYSPRINT --
          FLMALLOC  IOTYPE=W,RECFM=FBA,LRECL=121,                    C
              RECNUM=9000,PRINT=I
*  7       -- N/A --
          FLMALLOC  IOTYPE=N
*  8       -- SYSUT1 --
          FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=800,RECNUM=9000
*  9       -- SYSUT2 --
          FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=800,RECNUM=9000
* 10       -- SYSUT3 --
          FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=800,RECNUM=9000
```

Figure 19. COBOL II with DB2 Preprocessor (Part 1 of 2)

```
* 11       -- N/A --
           FLMALLOC  IOTYPE=N
* 12       -- SYSTERM --
           FLMALLOC  IOTYPE=A
            FLMCPYLB NULLFILE
* 13       -- N/A --
           FLMALLOC  IOTYPE=N
* 14       -- SYSCIN --
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,                        C
               RECNUM=9000,DDNAME=SYSCIN
* 15       -- N/A --
           FLMALLOC  IOTYPE=N
* 16       -- DBRMLIB--
           FLMALLOC  IOTYPE=P,DDNAME=DBRMLIB,MEMBER=@@FLMONM,           C
               DFLTTYP=DBRM,KEYREF=OUT1,                                C
               RECFM=FB,LRECL=80,RECNUM=5000,DIRBLKS=1
*
*         --COBOL II INTERFACE--
*
          FLMTRNSL   CALLNAM='COBOL II COMPILER',                       C
               FUNCTN=BUILD,                                            C
               COMPILE=IGYCRCTL,                                        C
               VERSION=2.0,                                             C
               GOODRC=0,                                                C
               PORDER=3,                                                C
               OPTIONS=(XREF,LIB,APOST,NODYNAM,LIST,NONUMBER,NOSEQ)
*
* DDNAME ALLOCATION (USING DDNAMELIST SUBSTITUTION)
*
*  1       (* SYSLIN *)
           FLMALLOC  IOTYPE=O,KEYREF=OBJ,RECFM=FB,LRECL=80,             C
               RECNUM=5000,DFLTTYP=OBJ,DDNAME=SYSLIN
*  2       (* N/A *)
           FLMALLOC  IOTYPE=N
*  3       (* N/A *)
           FLMALLOC  IOTYPE=N
*  4       (* SYSLIB *)
           FLMALLOC  IOTYPE=I,KEYREF=SINC,DDNAME=SYSLIB
*  5       (* SYSIN *)
           FLMALLOC  IOTYPE=U,DDNAME=SYSCIN
*  6       (* SYSPRINT *)
           FLMALLOC  IOTYPE=O,KEYREF=OUT2,RECFM=FBA,LRECL=133,          C
               RECNUM=25000,PRINT=Y,DFLTTYP=LIST,DDNAME=SYSPRINT
*  7       (* SYSPUNCH *)
           FLMALLOC  IOTYPE=A
            FLMCPYLB NULLFILE
*  8       (* SYSUT1 *)
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000
*  9       (* SYSUT2 *)
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000
* 10       (* SYSUT3 *)
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000
* 11       (* SYSUT4 *)
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000
* 12       (* SYSTERM *)
           FLMALLOC  IOTYPE=A,DDNAME=SYSTERM
            FLMCPYLB NULLFILE
* 13       (* SYSUT5 *)
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000
* 14       (* SYSUT6 *)
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000
* 15       (* SYSUT7 *)
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000
```

Figure 19. COBOL II with DB2 Preprocessor (Part 2 of 2)

Invoking User-Defined Parsers

SCLM allows you to replace an SCLM-supplied source parser with a user-defined source parser. This option is important when you are defining a new language for a project because such a language is likely to have a syntax unlike any of the languages that the SCLM-supplied parsers can recognize.

When you write a new parser for a language, you must:

1. Define the information tracked by SCLM in terms of the syntax of the language you want to support.
2. Write a program, based on the information you defined, that passes the statistical and dependency information for a member written in this new language to SCLM. This program is called a parser.
3. Tell SCLM how to invoke your parser.

At the end of this section is a parser, written in PL/I and Assembler, for the ISPF skeleton (SKELS) language. See Figure 21 on page 81, Figure 22 on page 89, and Figure 23 on page 89. We will take you through the three preceding steps and use the SKELS parser as an example.

Several user-modifiable parsers, written in REXX, are shipped with SCLM. FLMLRASM (Assembler), FLMLRCBL (COBOL), FLMRC2 (workstation C/C++ and resource files), FLMLRIPF (workstation help files), FLMLRC37 (C/370) and FLMLRCIS(C/C++ for MVS with include set support) are described in *ISPF Software Configuration and Library Manager (SCLM) Reference* Chapter 7. Understanding and Using the Customizable Parsers contains information on modifying the REXX parsers.

Defining Information Tracked by SCLM

SCLM tracks four kinds of information for each module:

- Statistical information

  Statistical information includes such data as the total lines and the number of comments in the module. See Part One of this book for a description of the 10 statistics kept by SCLM.

- Dependency information

  SCLM tracks two types of dependency information. The first is the name of the members that are included by a member. The second is the include set that is used to find the include. This information is used when a member is built or promoted. See "Specifying the Locations of Included Members" on page 91 for more information on the include information kept by SCLM.

- Change code information

  The change code information is a list of change codes associated with members under SCLM control. These change codes are optional unless the project manager has defined a change code verification routine requiring them. Includes and change codes for a member can be viewed with the Library Utility.

- User-defined information

  User-defined information is a list of free-form records derived from the member via the parse translator and stored in the accounting record. When writing a new parser, define exactly how the parser derives this information from a module.

Writing the Parser

Consider these things when you write your own parser:

- If any information is to be passed to the parser from SCLM, it is passed through a single parameter string as if your program had been invoked from TSO as:

  ```
  CALL program 'parameter list'
  ```

- You can use the SCLM variables to pass information to the parser about the module to be parsed.
- You can allocate any files you need (including the module to be parsed) to ddnames or pass the data set names directly through the parameter list.
- SCLM allocates space for an array and a structure. It is up to the parser to place statistical and dependency information in the array and the structure as it parses the module. SCLM can pass the address of the structure and the array to the parser through the parameter list string. If the parser returns a successful return code, SCLM moves the parsed information into the accounting record of the module.

The SKELS parser example consists of four routines. Together, these routines perform the work needed to parse an ISPF skeleton as we have described.

GETPTRS
: Takes the addresses from the parameter list and places them in the appropriate pointer variables.

INITIAL
: Initializes the counter variables and the parse structure (STAT_INFO).

PARSE
: Reads the lines of the skeleton one at a time, and saves any statistical or dependency information it finds.

WRAPUP
: Prepares the parse structure and the parse array (LIST_INFO) to be passed back to SCLM.

Telling SCLM How to Invoke Your Parser

You need to add a few SCLM macros to your project definition for SCLM to invoke your parser. The macros used to define the SKELS parser are shown in Figure 20 on page 80 For your parser, you need:

- An FLMLANGL to define your language (if it is not already there)
- An FLMTRNSL to define your parser
- An FLMALLOC for each ddname required by your parser
- An FLMCPYLB for each data set name you want to specify.

In Figure 20, you can examine the keywords on the macros to see how they are used.

On the FLMLANGL macro, the LANG parameter indicates the string (in this case it is SKELS) that needs to be given to SCLM when you want SCLM to treat a module like a skeleton. The BUFSIZE parameter is the number of elements in the LIST_INFO array that SCLM passes to the parser.

On the FLMTRNSL macro, the COMPILE and DSNAME parameter tell SCLM that the parser can be found in SCLM.PROJECT.LOAD(FLM@SKLS). The OPTIONS parameter contains three SCLM variables: @@FLMSTP, @@FLMLIS, and @@FLMSIZ. When the parser converts the character string values of @@FLMLIS and @@FLMSTP to fullword binary integers, the result is the addresses of the

LIST_INFO array and the STATS_INFO structure, respectively. The value of @@FLMSIZ is the number of bytes allocated for the LIST_INFO array.

The first FLMALLOC macro allocates the module to be parsed to ddname SSOURCE. The SKELS parser looks at this ddname for the skeleton source. The second FLMALLOC macro allocates an error listings file. If an error occurs during the parse, the SKELS parser writes out a message explaining the situation and providing a recommended solution. If the SKELS parser passes back a return code greater than that specified on the GOODRC parameter of the FLMTRNSL macro, the contents of this listings file are written to the edit listings file for the parse. This is the way you can pass messages and information about the parse to your users.

```
/*********************************************************************/
/* ISPF SKELETON LANGUAGE DEFINITION                                 */
/*********************************************************************/
         FLMLANGL    LANG=SKEL,VERSION=V2.3,BUFSIZE=50

  PARSER TRANSLATOR

          FLMTRNSL   CALLNAM='SKEL PARSER',                          C
               COMPILE=FLM@SKLS,                                     C
               DSNAME=SCLM.PROJECT.LOAD,                             C
               FUNCTN=PARSE,                                         C
               PORDER=1,                                             C
               GOODRC=0,                                             C
               VERSION=V1R0M0,                                       C
               OPTIONS='/@@FLMSTP,@@FLMLIS,@@FLMSIZ,'
           (* SOURCE      *)
           FLMALLOC IOTYPE=A,DDNAME=SSOURCE
           FLMCPYLB @@FLMDSN(@@FLMMBR)
           (* LISTING     *)
           FLMALLOC IOTYPE=W,RECFM=VBA,LRECL=133,                    C
               RECNUM=6000,DDNAME=ERROR,PRINT=Y
```

Figure 20. SKELS Parser Definition

```
PROCESS;
/*********************************************************************/
/***                                                               ***/
/***  Program:   PSKELS                                            ***/
/***                                                               ***/
/***  Purpose:   Performs an SCLM parse of ISPF skeletons after    ***/
/***             SCLM edit and during migration of source to SCLM.***/
/***                                                               ***/
/***  Inputs:    A parameter list containing addresses of a        ***/
/***             structure and a variable-length array into which ***/
/***             parse information is placed.  The length of the  ***/
/***             array, in bytes, is also passed.                  ***/
/***                                                               ***/
/***             In addition, source from the member to be parsed ***/
/***             is read from ddname SSOURCE.                      ***/
/***                                                               ***/
/***  Outputs:   The structure and array are filled with parse     ***/
/***             information by this program.  Any error messages ***/
/***             are written to ddname ERROR.                      ***/
/***                                                               ***/
/***  Retcode:   A fullword integer value, indicating the overall ***/
/***             success of the parse, is returned in register 15.***/
/***                                                               ***/
/***             0 = Successful parse; parse information is        ***/
/***                 returned in the structure and array.          ***/
/***                                                               ***/
/***             4 = Variable-length array was too small to hold   ***/
/***                 all of the parsed information.  Not all       ***/
/***                 information was passed back to SCLM.  The     ***/
/***                 number of elements needed is shown in the     ***/
/***                 listings data set.                            ***/
/***                                                               ***/
/***                 To correct this problem, either:              ***/
/***                                                               ***/
/***                   * Increase the value of BUFSIZE in the      ***/
/***                     FLMLANGL macro for this parser, or        ***/
/***                                                               ***/
/***                   * Break the skeleton being parsed into      ***/
/***                     smaller skeletons and use )IM to join     ***/
/***                     them back together.                       ***/
/***                                                               ***/
/***  Logic:     1) Obtain addresses of structure and array from   ***/
/***                parameter list.                                ***/
/***             2) Initialize counters in structure.              ***/
/***             3) For each line of skeleton source:              ***/
/***                a) Increment appropriate counters.             ***/
/***                b) If record starts with )IM, find and save    ***/
/***                   imbedded skeleton name.                     ***/
/***                c) Scan the record for variable names and      ***/
/***                   save in a program array any new names.      ***/
/***                d) If record starts with )DEFAULT, get new     ***/
/***                   '&' and ')' characters.                     ***/
/***             4) Calculate summary statistics.                  ***/
/***             5) Write an 'END ' element to end of parse array.***/
/***             6) Return.                                        ***/
/***                                                               ***/
/*********************************************************************/
```

Figure 21. Parser for ISPF Skeletons (Part 1 of 8)

```
PSKELS: PROC(PARMLIST) OPTIONS(MAIN);
   DCL PARMLIST     CHAR(255) VAR; /* Parameter list               */
   DCL PARMLISTx    CHAR(255) VAR; /* Copy of the parameter list   */
   DCL PAREN        CHAR(1),       /* Contains ')' special char    */
       NAME         CHAR(8),       /* Contains a referenced name   */
       NAMECHRS     CHAR(39),      /* Valid name characters        */
       RECORD       CHAR(80),      /* Output buffer for error list */
       STAT_PTR     POINTER,       /* Points to stats structure    */
       LIST_PTR     POINTER,       /* Points to parse array        */
       NON_COM_READ BIT(1),        /* Prolog flag                  */
       EOF          BIT(1),        /* End-of-file flag             */
       (I,J,K)      FIXED BIN(31), /* Simple counters              */
       USED_ELMTS   FIXED BIN(31), /* Number of parse array        */
                                   /*    elements used so far      */
       LISTLEN      FIXED BIN(31), /* Total number of available    */
                                   /*    parse array elements      */
       RETCODE      FIXED BIN(31); /* Return code                  */
   DCL ADDR         BUILTIN,
       INDEX        BUILTIN,
       LENGTH       BUILTIN,
       MIN          BUILTIN,
       REPEAT       BUILTIN,
       SUBSTR       BUILTIN,
       VERIFY       BUILTIN,
       PLIRETC      BUILTIN;
   DCL SSOURCE      FILE STREAM INPUT;
   DCL ERROR        FILE STREAM PRINT;
   DCL FXB_OV       FIXED BIN(31), /* Fullword integer             */
       PTR_OV       POINTER BASED(ADDR(FXB_OV));
                                   /* Pointer variable overlay on  */
                                   /*    top of a fullword integer */
                                   /*    variable                  */
   %INCLUDE(STATINFO);
   %INCLUDE(LISTINFO);
   RETCODE = 0;
   CALL GETPTRS;
   CALL INITIAL;
   CALL PARSE;
   CALL WRAPUP;
   CALL PLIRETC(RETCODE);
```

Figure 21. Parser for ISPF Skeletons (Part 2 of 8)

```
GETPTRS: PROC;
/*****************************************************************/
/***                                                           ***/
/***  Routine:   GETPTRS                                       ***/
/***                                                           ***/
/***  Purpose:   Converts the information passed to this program  ***/
/***             into addresses and array length information.     ***/
/***                                                           ***/
/***  Inputs:    A varying length string containing parameters in ***/
/***             the following format:                         ***/
/***                                                           ***/
/***               '<stat_ptr>,<list_ptr>,<length>,'           ***/
/***                                                           ***/
/***                  where stat_ptr is the EBCDIC representation ***/
/***                                of the address of the static ***/
/***                                portion of the account       ***/
/***                                record for this member,      ***/
/***                        list_ptr is the EBCDIC representation ***/
/***                                of the address of the        ***/
/***                                dynamic portion of the       ***/
/***                                account record, and          ***/
/***                        length   is the number of bytes      ***/
/***                                allocated to the dynamic     ***/
/***                                portion of the account       ***/
/***                                record.  This value is equal ***/
/***                                to 228 times the number of   ***/
/***                                elements in that array.      ***/
/***                                                           ***/
/***             Note that this format is consistent with the     ***/
/***             OPTIONS keyword on the FLMTRNSL macro that       ***/
/***             specifies how to invoke this parser.             ***/
/***                                                           ***/
/***  Outputs:   The three variables, STAT_PTR, LIST_PTR and      ***/
/***             LISTLEN are set from the values in the           ***/
/***             parameter list.                               ***/
/***                                                           ***/
/***  Logic:     1) Find the first comma.                         ***/
/***             2) Convert the contents of the character string  ***/
/***                before that comma into integer format.  For   ***/
/***                example, the string '19,' would be converted  ***/
/***                into an integer (X'00000013')         ***/
/***             3) Find the next comma.                          ***/
/***             4) Convert the contents of the character string  ***/
/***                before that comma into integer format.        ***/
/***             5) Find the last comma.                          ***/
/***             6) Convert the contents of the character string  ***/
/***                before that comma into integer format.        ***/
/***                                                           ***/
/***  Note:      We take advantage of PL/I's ability to convert   ***/
/***             a number in character string format into a       ***/
/***             fullword binary value.                           ***/
/***                                                           ***/
/*****************************************************************/
   PARMLISTX = PARMLIST;
   I = INDEX(PARMLIST,',');
   FXB_OV = SUBSTR(PARMLIST,1,I-1);
   STAT_PTR = PTR_OV;
   PARMLIST = SUBSTR(PARMLIST,I+1,LENGTH(PARMLIST)-I);
```

Figure 21. Parser for ISPF Skeletons (Part 3 of 8)

```
   I = INDEX(PARMLIST,',');
   FXB_OV = SUBSTR(PARMLIST,1,I-1);
   LIST_PTR = PTR_OV;
   PARMLIST = SUBSTR(PARMLIST,I+1,LENGTH(PARMLIST)-I);

   I = INDEX(PARMLIST,',');
   LISTLEN = SUBSTR(PARMLIST,1,I-1);
   LISTLEN = LISTLEN / 228;
END GETPTRS;
INITIAL: PROC;
/*********************************************************************/
/***                                                               ***/
/***  Routine:   INITIAL                                           ***/
/***                                                               ***/
/***  Purpose:   Initializes the counters and variables to be      ***/
/***             used during the parse.                            ***/
/***                                                               ***/
/***  Inputs:    None.                                             ***/
/***                                                               ***/
/***  Outputs:   Initialized variables.                            ***/
/***                                                               ***/
/*********************************************************************/
   STATINFO.LINES.TOTAL      = 0; /* # of lines in the skeleton     */
   STATINFO.LINES.COMMENT    = 0; /* # of lines starting with )CM   */
   STATINFO.LINES.NON_COMMENT= 0; /* # lines not starting w/ )CM    */
   STATINFO.LINES.BLANK      = 0; /* # lines starting with )BLANK   */
   STATINFO.LINES.PROLOG     = 0; /* # lines before 1st noncomment  */
      /**/
   STATINFO.STMTS.TOTAL      = 0; /* = LINES.TOTAL                 */
   STATINFO.STMTS.COMMENT    = 0; /* = LINES.COMMENT               */
   STATINFO.STMTS.CONTROL    = 0; /* # of lines starting with )     */
   STATINFO.STMTS.ASSIGNMENT = 0; /* = 0                           */
   STATINFO.STMTS.NON_COMMENT= 0; /* = LINES.NON_COMMENT           */
      /**/
   USED_ELMTS = 0;
      /**/
   NAMECHRS   = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789@#$';
   PAREN = ')';
END INITIAL;
PARSE: PROC;
/*********************************************************************/
/***                                                               ***/
/***  Routine:   PARSE                                             ***/
/***                                                               ***/
/***  Purpose:   Parses the skeleton and places the result in the ***/
/***             account record structures whose addresses were    ***/
/***             passed to the program.                            ***/
/***                                                               ***/
/***  Inputs:    Skeleton source from ddname SSOURCE.              ***/
/***                                                               ***/
/***  Outputs:   Parse results in structure STAT_INFO and array    ***/
/***             LIST_INFO.                                        ***/
/***                                                               ***/
/***  Logic:     1) Read each record of the skeleton.  For each    ***/
/***                line read, increment the appropriate           ***/
/***                counters.                                      ***/
/***                                                               ***/
/*********************************************************************/
```

Figure 21. Parser for ISPF Skeletons (Part 4 of 8)

```
   OPEN FILE(SSOURCE);
   EOF = '0'B;
   NON_COM_READ = '0'B;
   ON ENDFILE(SSOURCE) EOF = '1'B;
   GET FILE(SSOURCE) EDIT(RECORD) (A(80));
   DO WHILE (¬EOF);
/*********************************************************************/
/***  Perform this loop for each record in the skeleton.          ***/
/*********************************************************************/
/***  Increment total line counter.                               ***/
/*********************************************************************/
      STATINFO.LINES.TOTAL = STATINFO.LINES.TOTAL + 1;
/*********************************************************************/
/***  If the line starts with )IM, save the name of the           ***/
/***  imbedded member in LIST_INFO in an 'INCL' array element.    ***/
/*********************************************************************/
      IF SUBSTR(RECORD,1,3) = PAREN || 'IM' THEN
        DO;
          CALL GETNAME;
          USED_ELMTS = USED_ELMTS + 1;
          IF USED_ELMTS < LISTLEN THEN
            DO;
              LISTINFO(USED_ELMTS).TYPE = 'INCL';
              LISTINFO(USED_ELMTS).DATA = NAME;
            END;
          ELSE;
        END;
      ELSE;
/*********************************************************************/
/***  If the line starts with )DOT, save the name of the          ***/
/***  referenced table in LIST_INFO in a 'USER' array element.    ***/
/*********************************************************************/
      IF SUBSTR(RECORD,1,4) = PAREN || 'DOT' THEN
        DO;
          CALL GETNAME;
          USED_ELMTS = USED_ELMTS + 1;
          IF USED_ELMTS < LISTLEN THEN
            DO;
              LISTINFO(USED_ELMTS).TYPE = 'USER';
              LISTINFO(USED_ELMTS).DATA = 'TABLE: ' || NAME;
            END;
          ELSE;
        END;
      ELSE;
/*********************************************************************/
/***  If the line starts with )CM, increment the comment          ***/
/***  counter.  Otherwise, increment the non-comment counter.     ***/
/*********************************************************************/
      IF SUBSTR(RECORD,1,3) = PAREN || 'CM' THEN
        STATINFO.LINES.COMMENT = STATINFO.LINES.COMMENT + 1;
      ELSE
        STATINFO.LINES.NON_COMMENT = STATINFO.LINES.NON_COMMENT + 1;
```

Figure 21. Parser for ISPF Skeletons (Part 5 of 8)

```
/*********************************************************************/
/***  If the line starts with )BLANK, increment the blank line    ***/
/***  counter.                                                    ***/
/*********************************************************************/
      IF SUBSTR(RECORD,1,6) = PAREN || 'BLANK' THEN
        STATINFO.LINES.BLANK = STATINFO.LINES.BLANK + 1;
      ELSE;
/*********************************************************************/
/***  If the line starts with ), increment the control            ***/
/***  statement counter.                                          ***/
/***                                                              ***/
/***  If the line does not start with ), increment the data       ***/
/***  line counter.                                               ***/
/***                                                              ***/
/***  If this is the first data line, then we have reached the end***/
/***  of the prolog (defined here as the comment lines before the ***/
/***  first data line).  Set the prolog count to the number of    ***/
/***  comments read so far.                                       ***/
/*********************************************************************/
      IF SUBSTR(RECORD,1,1) = PAREN THEN
        STATINFO.STMTS.CONTROL = STATINFO.STMTS.CONTROL + 1;
      ELSE
        DO;
          IF ¬NON_COM_READ THEN
            DO;
              STATINFO.LINES.PROLOG = STATINFO.LINES.COMMENT;
              NON_COM_READ = '1'B;
            END;
          ELSE;
        END;
/*********************************************************************/
/***  If this line starts with )DEFAULT, then the special         ***/
/***  character (the left parenthesis) for control cards might    ***/
/***  have changed.  Get the new character.                       ***/
/*********************************************************************/
      IF SUBSTR(RECORD,1,8) = PAREN || 'DEFAULT' THEN
        DO;
          I = VERIFY(SUBSTR(RECORD,9,72),' ') + 8;
          PAREN = SUBSTR(RECORD,I,1);
        END;
      ELSE;
/*********************************************************************/
/***  End of parse-a-line loop.  If there's another line, read it ***/
/***  and go back through the loop.                               ***/
/*********************************************************************/
      GET FILE(SSOURCE) EDIT(RECORD) (A(80));
   END;
   CLOSE FILE(SSOURCE);
/*********************************************************************/
/***  If there were no non-comment lines, then set the number of  ***/
/***  prolog lines to the number of comment lines.                ***/
/*********************************************************************/
   IF ¬NON_COM_READ THEN
     STATINFO.LINES.PROLOG = STATINFO.LINES.COMMENT;
   ELSE;
END PARSE;
```

Figure 21. Parser for ISPF Skeletons (Part 6 of 8)

```
GETNAME: PROC;
/*********************************************************************/
/***                                                               ***/
/***  Routine:   GETNAME                                           ***/
/***                                                               ***/
/***  Purpose:   Returns the name specified on an )IM or )DOT      ***/
/***             statement.                                        ***/
/***                                                               ***/
/***  Inputs:    An 80-byte record in variable RECORD.             ***/
/***                                                               ***/
/***  Outputs:   The 8-byte name in variable NAME.                 ***/
/***                                                               ***/
/***  Logic:     1) Find the first blank after the )IM or )DOT.    ***/
/***             2) Find the next non-blank after that blank.      ***/
/***             3) Move that non-blank and the next 7 bytes into  ***/
/***                variable NAME.                                 ***/
/***                                                               ***/
/*********************************************************************/
   I = INDEX(RECORD,' ');
   I = VERIFY(SUBSTR(RECORD,I,81-I),' ') + I - 1;
   NAME = SUBSTR(RECORD,I,8);
END GETNAME;

WRAPUP: PROC;
/*********************************************************************/
/***                                                               ***/
/***  Routine:   WRAPUP                                            ***/
/***                                                               ***/
/***  Purpose:   Saves the last of the parse information in the    ***/
/***             SCLM structures and outputs error messages to     ***/
/***             the listing file if the LIST_INFO array was not   ***/
/***             large enough to hold all of the information.      ***/
/***                                                               ***/
/***  Inputs:    None.                                             ***/
/***                                                               ***/
/***  Outputs:   More data in LIST_INFO and STAT_INFO.             ***/
/***                                                               ***/
/***  Logic:     1) Calculate summary information.                 ***/
/***             2) Write an 'END ' element to LIST_INFO.          ***/
/***             3) If there was not enough room in LIST_INFO,     ***/
/***                write out messages that describe the error     ***/
/***                and that indicate how to solve the problem.    ***/
/***                                                               ***/
/*********************************************************************/
   STATINFO.STMTS.TOTAL       = STATINFO.LINES.TOTAL;
   STATINFO.STMTS.COMMENT     = STATINFO.LINES.COMMENT;
   STATINFO.STMTS.NON_COMMENT = STATINFO.LINES.NON_COMMENT;
```

Figure 21. Parser for ISPF Skeletons (Part 7 of 8)

```
    /**/
    /*   WRITE AN END ELEMENT TO LIST ARRAY                          */
    /**/
    USED_ELMTS = USED_ELMTS + 1;
    IF USED_ELMTS < LISTLEN THEN
      DO;
        LISTINFO(USED_ELMTS).TYPE = 'END ';
        LISTINFO(USED_ELMTS).DATA = ' ';
      END;
    ELSE
      DO;
        OPEN FILE(ERROR);
            /**/
        PUT FILE(ERROR) SKIP LIST(
                 'ERROR: INFORMATION RESULTING FROM PARSE DOES NOT ' ||
                 'FIT IN PARSE ARRAYS.');
            /**/
        PUT FILE(ERROR) SKIP LIST(
                 '       PARSE ARRAY ELEMENTS:', LISTLEN);
            /**/
        PUT FILE(ERROR) SKIP LIST(
                 '       ELEMENTS NEEDED:     ', USED_ELMTS);
            /**/
        PUT FILE(ERROR) SKIP(2) LIST(
                 'FIX:   1) INCREASE BUFSIZE VALUE IN FLMLANGL MACRO,');
            /**/
        PUT FILE(ERROR) SKIP LIST(
                 '              - OR - ');
            /**/
        PUT FILE(ERROR) SKIP LIST(
                 '       2) BREAK THIS SKELETON UP INTO SMALLER ' ||
                 'SKELETONS AND IMBED THEM ');
            /**/
        PUT FILE(ERROR) SKIP LIST(
                 '          IN A NEW "TOP LEVEL" SKELETON ');
            /**/
        PUT FILE(ERROR) SKIP(2) LIST(
                 'PARAMETER LIST: ' || PARMLISTX);
            /**/
        LISTINFO(LISTLEN).TYPE = 'END ';
        LISTINFO(LISTLEN).DATA = ' ';
            /**/
        CLOSE FILE(ERROR);
            /**/
        RETCODE = 4;
      END;
  END WRAPUP;
  END PSKELS;
```

Figure 21. Parser for ISPF Skeletons (Part 8 of 8)

```
/*********************************************************************/
/***                                                               ***/
/***  LISTINFO Structure                                           ***/
/***                                                               ***/
/***  Maps the static portion of the account record.               ***/
/***                                                               ***/
/***  The number of elements declared for this array should not    ***/
/***  be greater than the value specified on the BUFSIZE keyword   ***/
/***  on the FLMLANGL macro.                                       ***/
/***                                                               ***/
/*********************************************************************/
   DCL 1 LISTINFO(50)      BASED(LIST_PTR),
         2 TYPE            CHAR(4),
         2 DATA            CHAR(224);
```

Figure 22. LISTINFO Module

```
/*********************************************************************/
/***                                                               ***/
/***  STATINFO Structure                                           ***/
/***                                                               ***/
/***  Maps the static portion of the account record.               ***/
/***                                                               ***/
/*********************************************************************/
   DCL 1 STATINFO          BASED(STAT_PTR),
         2 LINES,
           3 TOTAL         FIXED BIN(31),
           3 COMMENT       FIXED BIN(31),
           3 NON_COMMENT   FIXED BIN(31),
           3 BLANK         FIXED BIN(31),
           3 PROLOG        FIXED BIN(31),
         2 STMTS,
           3 TOTAL         FIXED BIN(31),
           3 COMMENT       FIXED BIN(31),
           3 CONTROL       FIXED BIN(31),
           3 ASSIGNMENT    FIXED BIN(31),
           3 NON_COMMENT   FIXED BIN(31);
```

Figure 23. STATINFO Module

Processing Conditionally Saved Components

SCLM provides a feature to handle translators that, by design, have missing or static outputs. Static outputs help SCLM in its work-avoidance algorithms. Note, however, that SCLM relies on translator return codes to determine which outputs are static.

Example of Processing Conditionally Saved Components

Suppose a translator can determine if a developer changed only comments in the source code and signals that by a return code of 2. The translator creates a listing output to match the current source. However, creating object code for the source is unnecessary because comment changes to source do not alter object code. In this case, the object code is a static output because it did not change. Specifying a NOSAVRC=2 on the FLMALLOC macro corresponding to the object output instructs SCLM not to copy object modules back to the hierarchy when the translator returns a 2. SCLM copies the generated listing back to the hierarchy when the translator returns a 2, if the object modules already exist in the heirarchy.

Components that depend on the object do not need to be rebuilt when only the listing is regenerated. If you specify DEPPRCS=N on the FLMLANGL macro, SCLM rebuilds components dependent on a member only if all its outputs were saved.

```
          FLMLANGL    LANG=XYZ,VERSION=V1,DEPPRCS=N
* BUILD TRANSLATOR(S)
*
          FLMTRNSL   CALLNAM='TRANSLATOR XYZ',                        C
               FUNCTN=BUILD,                                          C
               COMPILE=XYZ,                                           C
               GOODRC=4
*
*         (* SYSIN *)
          FLMALLOC  IOTYPE=S,KEYREF=SINC,RECFM=FB,LRECL=80,           C
             RECNUM=1000,DDNAME=SYSIN
*         (* SYSPRINT *)
          FLMALLOC  IOTYPE=O,KEYREF=LIST,RECFM=VBA,LRECL=133,         C
             RECNUM=30000,PRINT=Y,DDNAME=SYSPRINT,DFLTTYP=LISTING
*         (* SYSLIN *)
          FLMALLOC  IOTYPE=O,KEYREF=OBJ,RECFM=FB,LRECL=80,            C
             RECNUM=5000,DDNAME=SYSLIN,DFLTTYP=OBJ,NOSAVRC=2
```

Figure 24. Sample Language Definition for Conditionally Saved Components

Setting Up the Project Definition

To access this feature, use the FLMALLOC, FLMLANGL, and FLMTRNSL macros:

1. Identify the static outputs and their corresponding FLMALLOCs in the language definition.
2. For each static output:
   - List the translator return code that indicates that the output is not to be saved
   - Specify that return code as the NOSAVRC parameter of the FLMALLOC macro for that output.

   The NOSAVRC must have a nonzero positive value. It is only valid for IOTYPEs O and P.
3. Make sure that the GOODRC on the FLMTRNSL macro corresponding to that translator is greater than or equal to the highest NOSAVRC parameter you specified.
4. Determine whether you want SCLM to rebuild components that depend on a given member only if all its outputs (including the static outputs) were saved. If that is the case, specify DEPPRCS=N on the FLMLANGL macro. If you specify DEPPRCS=Y (or let it default to Y), SCLM rebuilds components that depend on that member whenever the build translator returns a good return code. In the preceding example, DEPPRCS=Y causes SCLM to rebuild components that depend on the given member even when only the listing has changed.

Likewise, the translator can directly store output in an external data set not under SCLM control. For example, the Ada translator controls output stored in Ada sublibraries. Under such circumstances, the build function requires a signal from the translator to detect whether or not some of the external outputs were saved to external data sets. SCLM uses NOSVEXT on the FLMTRNSL macro in the same fashion as the parameter NOSAVRC on the FLMALLOC macro to detect whether or not external outputs were saved.

Specifying the Locations of Included Members

SCLM tracks two pieces of information for each include member that is found by a parser. The first piece of information is the member name of the include; the second is the include set that contains the included member. If no include set is returned by the parser for a member, SCLM assigns that member to the default include set. The name of the default include set is all blanks.

SCLM does not track an include member if it meets all of the following conditions:

- The language definition for the member specifies CHKSYSLB=PARSE. This is the default.
- An accounting record for the include is not found by searching the hierarchy for each type specified on the FLMINCLS for the include set.
- The include is found in one of the data sets specified on an FLMSYSLB macro for the include set.

Includes that meet these conditions are removed from the list of includes stored in the accounting record of the member. Because the include is not being tracked, build and promote do not detect if the include is removed from the FLMSYSLB data sets or added to the project database.

Build ignores an include if it meets all of the following conditions:

- The language definition for the member specifies CHKSYSLB=BUILD.
- An accounting record for the include is not found by searching the hierarchy for each type specified on the FLMINCLS for the include set.
- The include is found in one of the data sets specified on an FLMSYSLB macro for the include set.

Includes that meet these conditions are removed from the list of includes stored in the build map record of the member. Because the include is not being tracked, build and promote will not detect if the include has changed since the last build.

The include information is used by build and promote to determine whether the member is up-to-date. When you build, the includes for an up-to-date member have the same type, date, time, and version as the last time that member was built. When you promote, the includes for an up-to-date member have the same date, time, and version as the last time that member was built. Promote does not search the types listed on FLMINCLS macros for includes. It relies instead on the information in the build map to determine the type name of the included member. If a member is not up-to-date, build attempts to rebuild the member and promote does not allow the member to be promoted to the next group in the hierarchy.

An include set is used to associate an included member name with the type or types in the project that are searched to find a member with that name. The FLMINCLS macro is used to associate an include set with one or more types in the project definition. Types are searched in the order listed on the FLMINCLS macro. Each type is searched from the current group to the top of the hierarchy before the next type in the list is searched.

The number of include sets used by a language is usually related to the number of include ddnames supported by the build translators for that language, where the includes are located in project data sets. If the build translator only supports one include ddname, a single include set is sufficient for that language. On the other

hand, if there are multiple build translators, each supporting an include ddname and the includes are separated into different types for each build translator, multiple include sets would be needed.

If multiple include sets are needed, parsers must return the appropriate include set for each include.

Example

This example shows how pieces of a project might look if it were set up to use multiple include sets.

The following list shows the different types of includes in the project and the location of each include type in the project data sets.

Include Type	Project Types and SYSLIB Data sets to Search
Constants	CONSTANT
Messages	INCLENGL, INCLUDE, PRODX.MSGLIB (syslib data set)
SQL Declarations	DCLGEN, source member's type, source member's extended type
All other includes	INCLUDE, source member's type, source member's extended type, SYS1.SEDCHDRS (syslib data set)

Figure 25 shows how the include section of a source member might be coded:

```
#include <stdio>               /* C standard i/o           */
EXEC SQL INCLUDE SQLDEF1;      /* SQL definitions          */
#include "DD:MESSAGE(prog1)"   /* prog1 specific messages  */
#include "DD:CONSTANT(common)" /* common constants         */
#include "DD:CONSTANT(prog1)"  /* prog1 specific constants */
```

Figure 25. Source member with includes in different include sets

The parser must return the following:

Member	include set
STDIO	
SQLDEF1	SQL
PROG1	MESSAGE
COMMON	CONSTANT
PROG1	CONSTANT

You could then use the language definition in Figure 26 on page 93 for this member.

```
*********************************************************************
*           C370 W/DB2  LANGUAGE DEFINITION FOR PROJECT X           *
*                                                                   *
*********************************************************************
*
CDB2      FLMSYSLB    SYS1.SEDCHDRS
*
          FLMLANGL    LANG=CDB2,VERSION=V1,ALCSYSLB=Y*
* CONSTANT INCLUDES
*
CONSTANT  FLMINCLS TYPES=(CONSTANT)*
* MESSAGE INCLUDES
*
MESSAGE   FLMINCLS TYPES=(INCLENGL,INCLUDE)*
* SQL INCLUDES
*
SQL       FLMINCLS TYPES=(DCLGEN,@@FLMTYP,@@FLMETP)*
* ALL OTHER INCLUDES - DEFAULT INCLUDE SET
*
          FLMINCLS TYPES=(INCLUDE,@@FLMTYP,@@FLMETP)*
* PARSER TRANSLATOR
*
          FLMTRNSL   CALLNAM='C370 REXX PARSER',                    C
               FUNCTN=PARSE,                                        C
               COMPILE=MYCPARSE,                                    C
               DSNAME=SOMEUSR.PARSER.LOAD,                          C
               CALLMETH=TSOLNK,                                     C
               PORDER=1,                                            C
               OPTIONS=(LISTSIZE=@@FLMSIZ,                          C
               LISTINFO=@@FLMLIS,                                   C
               STATINFO=@@FLMSTP)
*           (* SOURCE      *)
           FLMALLOC  IOTYPE=A,DDNAME=SOURCE
            FLMCPYLB @@FLMDSN(@@FLMMBR)*
* BUILD DB2 PREPROCESSOR TRANSLATOR
*
*         --DB2 PREPROCESSOR INTERFACE--
          FLMTRNSL   CALLNAM='DB2 C PREP',                          C
               FUNCTN=BUILD,                                        C
               COMPILE=DSNHPC,                                      C
               VERSION=D220,                                        C
               GOODRC=4,                                            C
               PORDER=3,                                            C
               OPTIONS=(HOST(C),APOST)
```

Figure 26. Language definition to support multiple include sets (Part 1 of 4)

```
*  1       -- N/A --
           FLMALLOC  IOTYPE=N
*  2       -- N/A --
           FLMALLOC  IOTYPE=N
*  3       -- N/A --
           FLMALLOC  IOTYPE=N
*  4       -- SYSLIB --
           FLMALLOC  IOTYPE=I,INCLS=SQL
*  5       -- SYSIN --
           FLMALLOC  IOTYPE=S,KEYREF=SINC,RECFM=FB,LRECL=80,          C
               RECNUM=5000
*  6       -- SYSPRINT --
           FLMALLOC  IOTYPE=W,RECFM=FBA,LRECL=133,                    C
               RECNUM=35000,PRINT=Y
*  7       -- N/A --
           FLMALLOC  IOTYPE=N
*  8       -- SYSUT1 --
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=800,RECNUM=9000
*  9       -- SYSUT2 --
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=800,RECNUM=9000
* 10       -- SYSUT3 --
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=800,RECNUM=9000
* 11       -- N/A --
           FLMALLOC  IOTYPE=N
* 12       -- SYSTERM --
           FLMALLOC  IOTYPE=A
            FLMCPYLB NULLFILE
* 13       -- N/A --
           FLMALLOC  IOTYPE=N
* 14       -- SYSCIN --
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,                      C
               RECNUM=9000,DDNAME=DB2TRANS
* 15       -- N/A --
           FLMALLOC  IOTYPE=N
* 16       -- DBRMLIB--
           FLMALLOC  IOTYPE=P,DDNAME=DBRMLIB,MEMBER=@@FLMONM,         C
               DFLTTYP=DBRM,KEYREF=OUT1,                              C
               RECFM=FB,LRECL=80,RECNUM=5000,DIRBLKS=1
*
* BUILD C370 TRANSLATOR
*
         FLMTRNSL   CALLNAM='C 370',                                  C
              FUNCTN=BUILD,                                           C
              COMPILE=EDCCOMP,                                        C
              DSNAME=SYS1.SEDCCOMP,                                   C
              VERSION=C210,                                           C
              GOODRC=0,                                               C
              PORDER=3,                                               C
              OPTIONS=(XREF,LANGLVL(SAAL2),SOURCE,OPT,TEST(ALL),      C
              MARGINS(1,72),NOGONUM,NOTERMINAL,FLAG(I),SHOWINC)
```

Figure 26. Language definition to support multiple include sets (Part 2 of 4)

```
*
* 1        (* SYSIN *)
           FLMALLOC  IOTYPE=U,DDNAME=DB2TRANS
*
* 2        (* SYSLIN *)
           FLMALLOC  IOTYPE=O,KEYREF=OBJ,RECFM=FB,LRECL=80,            C
               RECNUM=5000,DFLTTYP=OBJ
*
* 3        (* SYSMSGS *)
           FLMALLOC  IOTYPE=A
             FLMCPYLB SYS1.SEDCMSGS(EDCMSGE)
*
* 4        (* SYSLIB *)
           FLMALLOC  IOTYPE=A
             FLMCPYLB SYS1.SEDCHDRS
*
* 5        (* USERLIB *)
           FLMALLOC  IOTYPE=I
*
* 6        (* SYSPRINT *)
           FLMALLOC  IOTYPE=A
            FLMCPYLB NULLFILE
*
* 7        (* SYSCPRT  *)
           FLMALLOC  IOTYPE=O,KEYREF=LIST,RECFM=VBA,LRECL=137,         C
               RECNUM=20000,PRINT=Y,DFLTTYP=LIST
*
* 8        (* SYSPUNCH *)
           FLMALLOC  IOTYPE=A
            FLMCPYLB NULLFILE
*
* 9        (* SYSUT1 *)
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=2000
*
* 10       (* SYSUT4 *)
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=2000
*
* 11       (* SYSUT5 *)
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=2000*
* 12       (* SYSUT6 *)
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=2000
*
* 13       (* SYSUT7 *)
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=2000
*
* 14       (* SYSUT8 *)
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=2000
*
* 15       (* SYSUT9 *)
           FLMALLOC  IOTYPE=W,RECFM=VB,LRECL=137,RECNUM=2000
```

Figure 26. Language definition to support multiple include sets (Part 3 of 4)

```
*
* 16       (* SYSUT10 *)
           FLMALLOC  IOTYPE=A
            FLMCPYLB NULLFILE
*
*          (* CONSTANT *)
           FLMALLOC  IOTYPE=I,DDNAME=CONSTANT,INCLS=CONSTANT
*
*          (* MESSAGE *)
           FLMALLOC  IOTYPE=I,DDNAME=MESSAGE,INCLS=MESSAGE
```

Figure 26. Language definition to support multiple include sets (Part 4 of 4)

Dynamic Include Tracking

The SCLM build processor attempts to resolve all include references to source members *before* it invokes any translator. However, for some translators, the include for a source member cannot be resolved until *after* the translator invocation. Such includes are referred to as *dynamic includes*. SCLM can track dynamic includes if the dynamic includes for a member can be altered only by modification of the member or one of the included members.

To support dynamic includes, SCLM invokes an additional build translator step (FLMTRNSL macro) following the translator that produces the output data set containing a list of dynamic includes. This additional translator should parse the output data set for dynamic includes and store them in memory supplied by the build processor. You pass the address of this memory to the translator by specifying the SCLM variable `@@FLMINC` in the translator options (OPTION parameter on FLMTRNSL macro). `@@FLMINC` is a pointer to a set of includes relating to a specified member. The value of `@@FLMINC` is a string of decimal characters that you must convert to a fullword binary value before using it as an address. The following record layout is used to store the dynamic includes:

```
COUNT    : 4 bytes
TYPE1    : 8 bytes
MEMBER1  : 8 bytes
TYPE2    : 8 bytes
MEMBER2  : 8 bytes
    .
    .
    .
TYPE#    : 8 bytes
MEMBER#  : 8 bytes
```

You must specify the number of dynamic includes in the first 4 bytes as a fullword binary integer, followed by the list of dynamic include member and type names. The amount of memory that the SCLM build processor supplies limits the number of dynamic includes to 1000.

When using dynamic includes, consider the following:

- Be sure to remove any duplicate include references before placing them in the structure pointed to by @@FLMINC.
- Do not return any include references that are actually to external (non-SCLM) libraries. The build step will receive an error (FLM01001) for any members not in the specified SCLM library.

- Deletion of members referenced through a dynamic include causes a build verification error (FLM43001). The build process does not proceed, even when using unconditional mode. If a referenced member is to be deleted, a build using the updated source should be performed before the deletion so that the build map can be updated to remove the reference.
- Dynamic include references to members that are outputs of other members do not cause a relationship to the member that created it, even when using extended mode. Builds and promotes for these must use a high-level archdef whose scope includes both source members.

Input List Translators

SCLM provides support for Build translators that operate on more than one source member in a single invocation. This type of translator is known as an input list translator. SCLM users can use existing translators that support this feature or write new user-defined translators to take advantage of the feature. The IBM Ada/370 Compiler is the only SCLM-supported translator that can use input lists.

The SCLM Input List feature can increase the performance of an SCLM Build. Instead of SCLM calling a translator once for each member to be built, SCLM calls the translator passing a list of members to be built. SCLM attempts to place as many members as possible on each input list, thereby limiting the number of translator invocations. The project manager specifies the maximum number of members passed to a translator on an invocation in the language definition that includes the translator. This feature is most useful when using translators that have a high startup overhead to run. Fewer invocations mean increased speed for the SCLM Build process.

An input list translator receives a file that contains a list of data sets that a Build action is performed against. It returns a file that contains a return code for each data set in the input list and, optionally, a set of unique outputs for each data set in the input list.

Two translators, FLMTPRE and FLMTPST, serve as the interfaces between SCLM and the input list translator.

- The FLMTPRE translator generates a list of data sets that an input list translator can use as input.
- The FLMTPST translator passes the return code information that an input list translator provides for every data set on the input list back to SCLM.

Refer to the *SCLM Reference* for more information on FLMTPRE and FLMTPST.

Note: The input list feature of the Build function is designed to work with direct translations of source members only (source members referenced with an INCLD statement). Using the input list feature with source members controlled by CC or Generic architecture definitions will produce undefined results (source members referenced with a SINC statement).

Configuring the Input List Translators

Use the following macros to configure the input list translators to fit your needs:

- FLMLANGL

  Set the following parameters:

  – INPLIST=Y

- MBRLMT to the maximum number of members that can be included in the same invocation of the translator.
- SLOCLMT to the maximum number of source lines to be processed on a single invocation of the translator.

- FLMTRNSL

  Set the following parameters:
  - INPLIST=Y
  - MBRRC to the maximum good return code for each member in the input list. MBRRC defaults to 0 and is optional.
- FLMALLOC

  Set the following parameters:
  - MALLOC to designate which outputs of a translator have multiple unique instances.
  - IOTYPE to O or P.

    SCLM only saves outputs with IOTYPE=O in the hierarchy. For IOTYPE=O, you must also specify the FLMCPYLB macro and the data set name on FLMCPYLB must contain the @@FLMMBR variable somewhere in the variable string to enable SCLM to find the member-specific outputs. When IOTYPE=O is specified, the input list translator is expected to allocate the output data sets necessary for each member.

    Temporary data sets allocated with IOTYPE=P can be used as work data sets for the translators, but they cannot be stored in the hierarchy.
  - ALLCDEL to designate which output data sets were defined by the translator and should be deleted by SCLM.

Defining a New Language to SCLM

This section describes the control structures used to manage SCLM processes and illustrates how to define a new language to SCLM. An example is included to show the statements needed to define the control structures and SCLM macros. The example refers to a fictitious compiler, the Finnoga 4, to show how to gather the information you need and how to specify that information to SCLM in the form of language definition macros.

Using DDnames and DDname Substitution Lists

Many translators support a ddname substitution list; this contains ddnames, which are passed as a parameter to the translator. In Figure 29 on page 114, the ddname in position 5 is the ddname from which the compiler reads the source to be compiled. The ddname occupying that position in the ddname substitution list is usually called SYSIN. You can override the default ddname by placing another ddname in position 5 of the ddname substitution list. The compiler then reads from the other ddname. Table 13 on page 99 lists the various ddnames used by the Finnoga 4 compiler described in this example. The position number indicates the position of the ddname in a ddname substitution list. In addition, Table 13 on page 99 gives a brief description of the data sets allocated to the ddnames.

Note that some position numbers do not have a ddname associated with them.

SCLM allows a maximum of 512 characters for the ddname substitution list. Because every FLMALLOC for a given translator causes an 8-character ddname to be put into the ddname substitution list, when the PORDER > 1, a given translator may have a maximum of 64 FLMALLOCs.

Ddname substitution lists are usually documented in the programming guide for specific compilers and linkage editors. Note that it is rare for two different compilers to have the same ddname substitution list mappings.

Compilers are not required to support a ddname substitution list in order to be defined to SCLM. However, ddname substitution list support makes it easy to link or string two different compilers or preprocessors together. In "Defining a Preprocessor to SCLM" on page 111, you will see how a ddname substitution list is used to pass the outputs of a preprocessor to a compiler.

Compiler Options

Assume that there are four Finnoga 4 compiler options that you can use:

- SOURCE or NOSOURCE
- MACRO or NOMACRO
- OPTIMIZE or NOOPTIMIZE
- OBJ().

It is not critical at this point to understand what these options mean to the compiler, just which options are to be used for each compile. You should always specify SOURCE, NOMACRO, and OBJ(), but you must specify the OPTIMIZE parameter on a module-by-module basis.

Table 13. DDname Substitution List Example

Position Number	DDname	Description of data set(s) allocated
1	SYSLIN	A partitioned data set into which the Finnoga 4 compiler writes the object module. The OBJ keyword in the compiler's option string specifies the member name to use.
2	<none>	<none>
3	<none>	<none>
4	SYSLIB	One or more partitioned data sets through which the Finnoga 4 compiler searches for INCLUDE members.
5	SYSIN	A sequential data set that contains Finnoga 4 source to be compiled.
6	SYSPRINT	A sequential listings data set. The Finnoga 4 compiler writes out a copy of the source that was compiled along with any error, warning, and informational messages.
7	<none>	<none>
8	FINLIB	A data set that contains information needed by the Finnoga 4 compiler. This data set comes with the compiler.
9	<none>	<none>

Table 13. DDname Substitution List Example (continued)

Position Number	DDname	Description of data set(s) allocated
10	SYSUT1	A sequential work data set.
11	SYSUT2	A sequential work data set.

Defining a New Language: Step-by-Step

The following list briefly describes the process required to write a new SCLM language definition:

1. Define the language name to SCLM.
2. Define include-sets for the language to identify the locations of included members.
3. List the various programs (parsers, compilers, and so on) used to parse and build your source.
4. For each program (or translator), look up the ddname substitution list (usually in the Programmer's Guide for the compiler), or list the ddnames used by the program.
5. For each program or translator, write an FLMTRNSL macro followed by FLMALLOC macros (one for each ddname to be allocated for the translator). Use the information in the program documentation to determine which IOTYPE value to specify as well as which other FLMALLOC keywords are appropriate.
6. Write a sample architecture definition and send it to your users. Describe to your users how to convert a JCL file of linkage editor control statements into architecture definitions.
7. Place the application under SCLM control.

This section is an illustration of the process for defining a language to SCLM. As you progress through the definition, you will code SCLM macros with the information SCLM needs to control Finnoga 4 modules. You will place this code into a member of the PROJDEFS.SOURCE data set called @FINNOGA. Language definitions such as @FINNOGA are usually referenced in the code for a project definition by means of the COPY statement.

Step 1.

Define the language.

The first step is to tell SCLM that you are defining a new language. To do so, code the following FLMLANGL macro:

```
FLMLANGL LANG=FINNOGA,VERSION=FINN4
```

In this example, values are specified for two parameters. The default values are used for the other parameters.

Parameter	Description
LANG=	Specifies the language name a user must enter on the SPROF panel or on the Migrate Utility panel to request that this language definition be used to drive build and parse operations of the Finnoga 4 modules.
VERSION=	Identifies the specific release of the current Finnoga 4 compiler. If you install a new

release or version of the Finnoga 4 compiler, you can set this parameter to a different value so that SCLM can mark all Finnoga 4 modules needing to be rebuilt. You must then re-assemble and link your project definition.

Step 2.

Define include sets for the language to identify the locations of included members.

After the language is defined, you can specify where SCLM finds included members for the Finnoga 4 language. In the following example, the FLMINCLS macro is used to list the types that are searched for includes:

```
FLMINCLS TYPES=(INCLUDE,@@FLMTYP)
```

In this example, the TYPES parameter of the FLMINCLS macro is used to tell SCLM where to look for includes. Since no name is specified, this definition applies to the default include set.

Parameter	Description
FLMINCLS name	Specifies the name of the include set that uses this definition. If no name is specified (as in this example), the definition is associated with the default include set. An include set defines a search path for all includes associated with that include set. Multiple include sets can be specified in a language definition if the parser and compiler support distinguishing one kind of include from another. For the parser, this means that the syntax of the language must support determining which include set an include belongs to. For the compiler, this means that a separate ddname must be used for each different include set (kind of include). Two include sets are useful when the standard language includes are kept in one Type and the “EXEC SQL” includes are kept in another Type. A parser can be written to determine which include set each include is in. The language definition then associates a ddname from the build translators with the appropriate include set name.
TYPES=	Specifies the name(s) of the types which are searched to find includes. In this case, the “INCLUDE” type is searched first. The @@FLMTYP SCLM variable indicates that the type of the member that is processed by the Finnoga 4 compiler is to be searched next. For example, if ’EXAMPLE.USERX.SOURCE(PROGA)’ is going to be compiled, SCLM looks for

includes first in the datasets associated with the INCLUDE type and then the SOURCE type.

Step 3.

Specify the programs that process the modules.

Next, identify the programs that are used to parse and build the Finnoga 4 modules. There are usually two such programs: a parser and the compiler. For each of these programs, code an FLMTRNSL macro and the appropriate FLMALLOC macros and FLMCPYLB macros.

Assume that you have written your own parser and that it is in the data set SCLM.PROJDEFS.LOAD(FINPARSE). The parser requires an option string @@FLMSIZ,@@FLMSTP,@@FLMLIS, and reads the source from ddname SOURCE.

Add this to your language definition:

```
FLMTRNSL   CALLNAM='FINNOGA PARSER',                          C
      FUNCTN=PARSE,                                           C
      COMPILE=FINPARSE,                                       C
      DSNAME=SCLM.PROJDEFS.LOAD,                              C
      PORDER=1,                                               C
      OPTIONS=(@@FLMSIZ,@@FLMSTP,@@FLMLIS)
```

The parameters included in this example are described as follows:

Parameter	Description
CALLNAM=	A character string that appears in messages during the specified FUNCTN (in this case PARSE). This value will assist in recognizing which translator was executing during the specified FUNCTN.
FUNCTN=	The value PARSE tells SCLM that this program is to be invoked whenever you parse a module with language FINNOGA.
COMPILE=	Member name of the load module for the Finnoga 4 parser. Note that the keyword "COMPILE" actually identifies the load module name of a translator (which may or may not be a compiler).
DSNAME=	Names the partitioned data set that contains the Finnoga 4 parser load module. DSNAME is required when the data set containing the desired module is not in the system concatenation. DSNAME is similar to a STEPLIB. When more than one data set is to be searched, the TASKLIB parameter can be used in conjunction with, or as a replacement for, the DSNAME parameter.
PORDER=	The value 1 tells SCLM that this program expects an options string but not a ddname substitution list.
OPTIONS=	Specifies the options string to be passed to the parser. Strings that start with `@@FLM` are SCLM variables, and they are replaced by their current values before the string is passed to the parser.

Since the parser reads its source from a ddname, you must tell SCLM how to allocate that ddname. To do this, use an FLMALLOC macro and an FLMCPYLB macro.

```
FLMALLOC  IOTYPE=A,DDNAME=SOURCE
FLMCPYLB  @@FLMDSN(@@FLMMBR)
```

A description of the parameters follows:

Parameter	Description
IOTYPE=A	Tells SCLM to allocate a ddname to one, or a concatenation of, specific data set(s). Each of those data sets are subsequently identified by using an FLMCPYLB macro.
DDNAME=	Identifies the ddname to be allocated.
@@FLMDSN(@@FLMMBR)	Identifies the member to be parsed. When the two SCLM variables are resolved, you get the member of the data set in which you are interested.

Now you can tell SCLM how to invoke the Finnoga 4 compiler. To do so, use an FLMTRNSL macro followed by one or more FLMALLOC and FLMCPYLB macros.

```
FLMTRNSL   CALLNAM='FINNOGA 4',                              C
      FUNCTN=BUILD,                                          C
      COMPILE=FNGAA40,                                       C
      PORDER=3,                                              C
      GOODRC=0,                                              C
      OPTIONS='SOURCE,NOMACRO,OBJ(@@FLMMBR),',               C
      PARMKWD=PARM1
```

You can specify only a few of the parameters and let SCLM supply default values for the others:

Parameter	Description
CALLNAM=	Names the compiler. This name appears in build messages.
FUNCTN=	Tells SCLM that this program gets invoked whenever you want to build a member with language FINNOGA.
COMPILE=	Identifies the load module name for the Finnoga 4 compiler.
DSNAME=	If you do not specify a DSNAME value, SCLM assumes that the load module can be found in the system concatenation.
PORDER=	The value 3 tells SCLM to pass an options string and a ddname substitution list to the Finnoga 4 compiler.
GOODRC=	The value 0 indicates that SCLM is to consider this build unsuccessful if the compiler completes with any return code greater than 0.
OPTIONS=	Specifies the options string to be passed to the compiler. At compiler run time, the SCLM variable @@FLMMBR is resolved to the member name being built.

PARMKWD=	The value PARM1 specifies the concatenation of the contents of the PARM1 parameters in the architecture definition to the preceding options string. Use the PARM1 parameter to specify the OPTIMIZE/NOOPTIMIZE option for each member. An example of this is provided later in this section.

As discussed previously, the Finnoga 4 compiler uses 7 ddnames and also supports a ddname substitution list. The preceding parser invocation definition showed how to define a translator (the parser) that does not use a ddname substitution list. The following SCLM FLMALLOC macros are used by SCLM to construct the ddname substitution list shown in Table 13 on page 99.

When you use a ddname substitution list, you must define the ddnames in the order in which they are expected to appear in the ddname substitution list by the translator. The first ddname defined is placed by SCLM into position 1 in the ddname substitution list. The second ddname specified is placed into position 2 in the ddname substitution list, and so on.

Note that you do not have to specify any ddnames in the following example macros. SCLM will create temporary unique ddnames and place them into the ddname substitution list positions. Because of the way ddname substitution lists work, the compiler uses those temporary ddnames instead of the standard documented ddnames (like SYSIN).

The first ddname in the Finnoga 4's ddname substitution list is SYSLIN. It is allocated to a partitioned data set into which the compiler places the object module.

```
FLMALLOC IOTYPE=P,KEYREF=OBJ,DFLTTYP=OBJ,RECFM=FB,LRECL=80,   C
    RECNUM=5000
```

The parameters specified in this macro are described as follows:

Parameter	Description
IOTYPE=P	The compiler is written in such a way that a partitioned data set must be allocated to this ddname. The compiler will write to a member of this partitioned data set. SCLM creates a temporary PDS and allocates it to a temporary ddname (since no DDNAME keyword was specified). This example illustrates two points. It shows how to define a temporary PDS for output from a translator and emphasizes that each compiler (or parser) that you define to SCLM may be slightly different from any other translator you have defined to SCLM. Always refer to the translator documentation when defining a translator to SCLM.
KEYREF=OBJ	To save what is written to this ddname and keep it under SCLM control, SCLM must be able to determine the member name and the SCLM-controlled data set name in which it is to save this output module. If SCLM is building an

architecture definition, it determines the project, group, type and member as follows:

- The high-level qualifier is the project identifier that was previously specified.
- The group is the level at which the build is taking place. The group name is the second qualifier.
- SCLM looks at the architecture definition being built and retrieves the member and type from the architecture statement associated with the keyword OBJ. The type name is the third qualifier.

DFLTTYP=OBJ

To save what is written to this ddname and keep it under SCLM control, SCLM must be able to determine the member name and the SCLM-controlled data set name in which it is to save this output module. If SCLM is building a source member, it determines the project, group, type and member as follows:

- The high-level qualifier is the project identifier that was previously specified.
- The group is the level at which the build is taking place.
- The type is the value of the DFLTTYP= keyword.
- The member name defaults to the name of the member being built.

If SCLM is building an architecture definition (and not a source member directly) then the DFLTTYP= value is ignored. Instead, SCLM uses the type associated with the KEYREF= value.

RECFM=FB

Specifies the record format of the temporary data set that SCLM creates. In this example, the record format is fixed block.

LRECL=80

Specifies the record length, in characters, of the temporary data set that SCLM creates.

RECNUM=5000

Tells SCLM to allocate enough space in this data set to hold 5000 records (records that are fixed block and 80 characters in length).

Positions 2 and 3 in the ddname substitution list are not used. Create two FLMALLOC macros with IOTYPE=N to tell SCLM to fill those name fields with hex zeros and to continue to the next ddname.

```
FLMALLOC IOTYPE=N
*
FLMALLOC IOTYPE=N
```

The ddname in position 4 of the ddname substitution list must be allocated to one or more partitioned data sets. This ddname is used by the Finnoga 4 compiler to find included members. The FLMINCLS macro described earlier needs to be referenced here to ensure that the compiler is picking up includes from the correct data sets. Since IOTYPE=I allocations default to the default include set shown earlier, this is automatically done. If another name was used on the FLMINCLS

macro, that name needs to be referenced here using the INCLS parameter. IOTYPE=I allocates a ddname with a concatenation of all the PDS's in the hierarchy starting with the group specified for the BUILD and ending with the top, or production level, group. First the hierarchy for the INCLUDE type is allocated, followed by the type of the first SINCed member from the architecture definition, or, if no architecture definition is used, the type of the member being built.

```
FLMALLOC IOTYPE=I,KEYREF=SINC
```

The parameters used with this macro are as follows:

Parameter	Description
IOTYPE=I	Allocate this ddname to a concatenation of SCLM-controlled data sets. The types used in the concatenation are determined by the FLMINCLS macro referenced by the INCLS= parameter on the FLMALLOC macro. In this case, there is no INCLS= parameter so the default FLMINCLS (or include set) is used. A hierarchy of datasets is concatenated for each type specified for the referenced FLMINCLS macro. The hierarchy begins at the group where the build is taking place and extends to the top of the project's hierarchy. In this case, the concatenation first contains all of the data sets for the INCLUDES type followed by the data sets for the value substituted into the @@FLMTYP variable. See the KEYREF= parameter to determine the value which is substituted into the @@FLMTYP and @@FLMETP variables.
KEYREF=SINC	If you are building an architecture definition, refer to the first SINC statement in that architecture definition for the type that is substituted into the @@FLMTYP macro. The value for @@FLMETP comes from the EXTEND= parameter of the FLMTYPE macro for that type. If you are not building an architecture definition, the type is the type of the member being built.

The next ddname in the ddname substitution list is allocated to the source to be compiled:

```
FLMALLOC IOTYPE=S,KEYREF=SINC
```

The parameters used in the example are as follows:

Parameter	Description
IOTYPE=S	Tells SCLM to allocate a temporary sequential data set.
KEYREF=SINC	If you are building a source module directly, SCLM copies that member to this temporary data set. If you are building a CC architecture definition, SCLM copies the members listed on the SINC statement to this data set.

Next, define the SYSPRINT ddname to SCLM.

```
FLMALLOC  IOTYPE=O,KEYREF=LIST,RECFM=VBA,LRECL=125,          C
     RECNUM=5000,PRINT=Y,DFLTTYP=FINLIST
```

This definition contains the following parameters:

Parameter	Description
IOTYPE=O	Specifies that the compiler writes to this ddname using a sequential data set. SCLM creates a temporary sequential data set and allocates it to a temporary ddname (since this is part of a ddname substitution list).
KEYREF=LIST	Refers SCLM to the LIST record in the architecture definition being built. That record contains the member name and type into which the listing is saved after a successful build. (SCLM copies the data from the temporary data sets into members of the PDS's controlled by SCLM after a successful build.)
DFLTTYP=FINLIST	Specifies the data set type into which this listing is written whenever a Finnoga 4 module is built directly or when using INCLD in an architecture definition.
PRINT=Y	Specifies that this is a listing that should be copied to the Build List data set after the build process completes.

Although the next position in the ddname substitution list is not used, you still need to tell SCLM what to put there. Create another FLMALLOC with IOTYPE=N:

```
FLMALLOC IOTYPE=N
```

Next, specify the FINLIB data set allocation to SCLM. Specifically, indicate that the Finnoga 4 library resides in a data set named SYS1.FINNOGA.LIB:

```
FLMALLOC IOTYPE=A
FLMCPYLB SYS1.FINNOGA.LIB
```

Finally, note that position 9 in the ddname substitution list, like position 7, is not used:

```
FLMALLOC IOTYPE=N
```

The last two ddnames in the ddname substitution list for the Finnoga 4 compiler are temporary work data sets. Use IOTYPE=W for temporary work data sets, such as SYSUT1, SYSUT2, and so on. In addition, specify the record format and length of the two files, as shown in the following example:

```
FLMALLOC IOTYPE=W,LRECL=4000,RECFM=F,RECNUM=4000
*
FLMALLOC IOTYPE=W,LRECL=4000,RECFM=F,RECNUM=4000
```

When you have completed all the steps described previously, you will have a language definition similar to the following one. (This language definition contains comments to explain the flow of operations.) When you are ready to reassemble your project definition, add a COPY statement in your main project definition file to pull in these macros.

```
*********************************************************
*  FINNOGA 4 LANGUAGE DEFINITION
*********************************************************
*
         FLMLANGL LANG=FINNOGA,VERSION=FINN4
*
*********************************************************
*  TYPES TO SEARCH FOR INCLUDES
*********************************************************
*
         FLMINCLS TYPES=(INCLUDE,@@FLMTYP)
*
*********************************************************
*  PARSE TRANSLATOR DEFINITION
*********************************************************
*
         FLMTRNSL   CALLNAM='FINNOGA PARSER',                          C
               FUNCTN=PARSE,                                           C
               COMPILE=FINPARSE,                                       C
               DSNAME=SCLM.PROJDEFS.LOAD,                              C
               PORDER=1,                                               C
               OPTIONS=(@@FLMSIZ,@@FLMSTP,@@FLMLIS)
*
*   -- SOURCE --
*
           FLMALLOC  IOTYPE=A,DDNAME=SOURCE
            FLMCPYLB  @@FLMDSN(@@FLMMBR)
*********************************************************
*  BUILD TRANSLATOR DEFINITION
*********************************************************
*
         FLMTRNSL   CALLNAM='FINNOGA 4',                               C
               FUNCTN=BUILD,                                           C
               COMPILE=FNGAA40,                                        C
               GOODRC=0,                                               C
               PORDER=3,                                               C
               OPTIONS='SOURCE,NOMACRO,OBJ(@FLMMBR),',                 C
               PARMKWD=PARM1
*
*   -- (1) OBJECT
*
           FLMALLOC IOTYPE=P,KEYREF=OBJ,DFLTTYP=OBJ,RECFM=FB,LRECL=80, C
               RECNUM=5000
*
*   -- (2) NOT USED
*
           FLMALLOC IOTYPE=N
*
*   -- (3) NOT USED
*
           FLMALLOC IOTYPE=N
*
*   -- (4) INCLUDE LIBRARIES
*
           FLMALLOC IOTYPE=I,KEYREF=SINC
*
*   -- (5) SOURCE
*
           FLMALLOC IOTYPE=S,KEYREF=SINC
*
*   -- (6) LISTING
*
           FLMALLOC  IOTYPE=O,KEYREF=LIST,RECFM=VBA,LRECL=125,         C
               RECNUM=5000,PRINT=Y,DFLTTYP=FINLIST
```

Figure 27. Finnoga 4 Language Definition (Part 1 of 2)

```
*
*   -- (7) NOT USED
*
           FLMALLOC IOTYPE=N*
*   -- (8) FINNOGA COMPILER LIBRARIES
*
           FLMALLOC IOTYPE=A
            FLMCPYLB SYS1.FINNOGA.LIB
*
*   -- (9) NOT USED
*
           FLMALLOC IOTYPE=N
*
*   -- (10) WORK FILE
*
           FLMALLOC IOTYPE=W,LRECL=4000,RECFM=F,RECNUM=4000
*
*   -- (11) WORK FILE
*
           FLMALLOC IOTYPE=W,LRECL=4000,RECFM=F,RECNUM=4000
*
*5665-402 (C) COPYRIGHT IBM CORP 1980, 1989
```

Figure 27. Finnoga 4 Language Definition (Part 2 of 2)

Showing Users How to Write CC Architecture Definitions

Once you have written the language definition, and assembled and link-edited the project definition, your users can use SCLM to build their Finnoga 4 applications. To do so, however, they must know what information to supply in their architecture definitions. Table 14 lists the SCLM-controlled inputs and outputs for the Finnoga 4 build. It includes the ddnames of the data sets that are input to and output from the Finnoga 4 compiler. In addition, a KEYREF value and brief description of each ddname is given.

Table 14. DDnames and KEYREFs

ddname	KEYREF	Description of data set(s) allocated
SYSLIN	OBJ	A partitioned data set into which the Finnoga 4 compiler writes the object module. The OBJ keyword in the compiler's option string specifies the member name to use.
SYSLIB	SINC	One or more partitioned data sets through which the Finnoga 4 compiler searches for include members.
SYSIN	SINC	A sequential data set that contains Finnoga 4 source to be compiled.

Table 14. DDnames and KEYREFs (continued)

ddname	KEYREF	Description of data set(s) allocated
SYSPRINT	LIST	A sequential listings data set. The Finnoga 4 compiler writes out a copy of the source that was compiled along with any error, warning, and informational messages.

In addition, the PARM1 parameter is used in the FLMTRNSL macro for the Finnoga 4 compiler.

When your users write CC architecture definitions for their Finnoga 4 applications, they must include each of the preceding KEYREFs. A typical Finnoga 4 CC architecture definition looks like this:

```
SINC   PROG   SOURCE
SINC   SUB1   SOURCE
OBJ    PROG   OBJ
LIST   PROG   FINLIST
PARM1  OPTIMIZE
```

This CC architecture definition, along with the language definition previously written, tells SCLM to compile the concatenation of Finnoga 4 members PROG and SUB1 in data set type SOURCE. The resulting object module and listing are to be saved in data set types OBJ and FINLIST, respectively. When the source is compiled, you want to use the OPTIMIZE compiler option.

You do not have to specify the modules that are included from ddname SYSLIB. Simply allocate SYSLIB to the proper libraries (with an IOTYPE=I) and the compiler will find the included members.

This simple template is all you have to give to your users. When they edit their Finnoga 4 source, they need to specify FINNOGA as the language name. Then they create their architecture definitions like the preceding one. SCLM and the language definition you created will perform the rest of the work.

Convert Your JCL Decks to Architecture Definitions

Suppose your Finnoga 4 users have a library of JCL that they have been using to compile their Finnoga 4 source. The following example uses a sample Finnoga 4 compile job and shows how you would write an architecture definition with the information in the JCL. The JCL deck that you use might look like this:

```
//JOB  ...
//FINNOGA EXEC PGM=FNGAA40,
//        PARM='SOURCE,NOMACRO,OBJ(PROG1),NOOPTIMIZE'
//SYSLIN   DD DSN=USER02.PRIVATE.OBJ,DISP=OLD
//SYSLIB   DD DSN=USER02.PRIVATE.FINNOGA,DISP=SHR
//SYSIN    DD DSN=USER02.PRIVATE.FINNOGA(MAIN),DISP=SHR
//         DD DSN=USER02.PRIVATE.FINNOGA(SUB1),DISP=SHR
//         DD DSN=USER02.PRIVATE.FINNOGA(SUB2),DISP=SHR
//SYSPRINT DD SYSOUT=A
//FINLIB   DD DSN=SYS1.FINNOGA.LIB,DISP=SHR
//SYSUT1   DD UNIT=SYSDA,VOL=SER=,DCB=(LRECL=4000,RECFM=F),
//         SPACE=(TRK,(10,10))
//SYSUT2   DD UNIT=SYSDA,VOL=SER=,DCB=(LRECL=4000,RECFM=F),
//         SPACE=(TRK,(10,10))
```

In this example, you want SCLM to control the modules that are input or output through ddnames SYSIN, SYSLIN, and SYSPRINT. For the Finnoga 4 language definition, the keywords SINC, OBJ and LIST have been assigned to those modules. You create the architecture definition by listing the modules involved in the build and identifying their roles with the keywords SINC, OBJ, and LIST. In addition, you tell SCLM to concatenate the NOOPTIMIZE option to the end of the OPTIONS string being passed to the translator using the PARM1 keyword.

```
SINC    MAIN   SOURCE
SINC    SUB1   SOURCE
SINC    SUB2   SOURCE
OBJ     PROG1  OBJ
LIST    MAIN   FINLIST
PARM1   NOOPTIMIZE
```

Now you are prepared to move this application under SCLM control:

1. Copy the members MAIN, SUB1, and SUB2 from 'USER02.PRIVATE.FINNOGA' to a development group in the SCLM project hierarchy. In this example, the data set type is SOURCE. You should also copy over any included source members.
2. Use the SCLM Migration Utility to migrate your source members using the language name FINNOGA (the name specified on the FLMLANGL macro).
3. Use the SCLM editor to create the architecture definition. Unless you have modified the ARCHDEF language definition, the language of this architecture definition should be ARCHDEF. SCLM asks for the language name when you first enter the SAVE or END edit command.

Your user is now ready to compile this application using SCLM. The source members are under SCLM control as are the architecture definitions. The object module and the Finnoga 4 listing have not yet been created. To build this application, select Build (option 10.4) from the SCLM Main Menu and enter the project, group, type, and member name of the architecture definition (archdef).

Defining a Preprocessor to SCLM

Suppose that some of your Finnoga 4 users run a preprocessor step on their Finnoga 4 source before compiling it. How do you define that two-step build process to SCLM? Using another fictitious product, the Panda Universal Preprocessor (PUPP), you can specify that some Finnoga 4 source is to be run through PUPP before it gets compiled.

Again, you need to list the ddnames used by the translator you want to define. In this case, assume that PUPP uses three ddnames:

Table 15. DDnames Used by a Hypothetical Preprocessor

DDname	Description of file(s) allocated
SYSIN	A sequential data set containing the Finnoga 4 source to be preprocessed.
SYSOUT	A sequential data set to which the preprocessed Finnoga 4 source is written. You want to compile the contents of this data set.
SYSPRINT	A listing data set containing Panda Universal Preprocessor messages and warnings.

In this example, the ddnames are not numbered because you will not use the PUPP ddname substitution list. Instead, you will use the ddname substitution list supported by the Finnoga 4 compiler to link the two build steps together.

Your users want SCLM to keep the listing data set produced by PUPP, but they do not want to keep the intermediate copy of the preprocessed source (the output in SYSOUT). The preprocessed source should be passed to the Finnoga 4 compiler and then deleted.

Because you want to preprocess some but not all of the Finnoga 4 source, you should define two different build processes to SCLM. You have already defined the latter build process (for language FINNOGA), and you will not change that language definition. For the two-step build process, however, you will create a new language definition with a different language name. The users must assign the correct language name to each Finnoga 4 source member.

The new language definition is very much like the first language definition, so you can copy the first definition into a second PROJDEFS.SOURCE member and modify it there.

The new language definition (copied from the first definition) has two FLMTRNSL macros: one for the parser, and the other for the Finnoga 4 compiler. You will add a third FLMTRNSL for the preprocessor, using the same macros and keywords as you used in the previous example. Enter this example before the FLMTRNSL for the Finnoga 4 compiler and after the last FLMALLOC for the parser. The order of execution is then parse, preprocess, and compile.

```
         FLMTRNSL   CALLNAM='PANDA U PREP',                          C
               FUNCTN=BUILD,                                         C
               COMPILE=PANDA01,                                      C
               GOODRC=0,                                             C
               PORDER=1,                                             C
               OPTIONS='NOTRACE'
*
*   -- SOURCE
*
            FLMALLOC IOTYPE=S,KEYREF=SINC,DDNAME=SYSIN
*
*   -- PREPROCESSED SOURCE
*
            FLMALLOC IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000,          C
                DDNAME=SYSOUT
*
*   -- LISTING
*
            FLMALLOC  IOTYPE=O,KEYREF=OUT1,RECFM=VBA,LRECL=125,       C
                RECNUM=5000,PRINT=Y,DFLTTYP=PUPLIST,DDNAME=SYSPRINT
*
```

Figure 28. Panda Universal Preprocessor

The following list describes the keywords that change so you can invoke the new language definition:

Keyword	Description
FUNCTN=	Identifies this translator as a build translator. There are now two build translators in this language definition: one for PUPP and one for the Finnoga 4 compiler. Define the PUPP translator first and the

	Finnoga 4 translator second to tell SCLM the order in which the translators are to be invoked.
OPTIONS=	Specifies the options string to be passed to the PUPP compiler. In this case, you do not want the trace option activated.
DDNAME=	Specify the DDNAME= keyword because you are not using a ddname substitution list to pass ddnames to PUPP. This parameter specifies which ddnames to allocate (the ddnames that PUPP uses).
IOTYPE=W	Specifies that ddname SYSOUT is to be allocated as a work file. In this example, the users do not want to save the processed source. When the build completes, this file is deleted. In a later step, this file gets passed to the Finnoga 4 compiler.
KEYREF=OUT1	Specifies that the listing PUPP writes to ddname SYSPRINT is to be saved under SCLM control. You usually use KEYREF=LIST for this purpose. However, KEYREF=LIST is already being used by the translator definition for the Finnoga 4 compiler. Because you have already used the standard set of CC archdef keywords, you must use the OUTx keywords. OUTx keywords are used to identify additional build outputs. You can use OUT0, OUT1,...,OUT9 to specify additional outputs that SCLM is to control.
PRINT=Y	This listing and the Finnoga 4 listing are both written to the build listing data set.

Passing the Source to the Compiler

You must next make one change to the macros that define how to invoke the Finnoga 4 compiler. The source to be compiled no longer comes directly from the SCLM-controlled source libraries. Instead, you want SCLM to take the preprocessed source that PUPP writes to ddname SYSOUT and pass it to the Finnoga 4 compiler. This requires a change to the FLMALLOC macro that defines the ddname that gets put into the SYSIN position in the ddname substitution list for the Finnoga 4 compiler. The new macro is illustrated as follows:

```
*
*   -- (5) SOURCE
*
      FLMALLOC IOTYPE=U,DDNAME=SYSOUT
```

You use a different IOTYPE value (IOTYPE=U) to indicate that the ddname to be placed in the ddname substitution list has already been allocated in a previous build step. In this case, DDNAME=SYSOUT tells SCLM to place the name SYSOUT in position 5 of the ddname substitution list and go on to the next ddname. When the Finnoga 4 compiler runs, it reads the source from ddname SYSOUT.

The new language definition is shown in Figure 29 on page 114. Note that the new language has been specified on the FLMLANGL macro.

```
*********************************************************
*  FINNOGA 4 LANGUAGE DEFINITION
*********************************************************
*
         FLMLANGL LANG=FINPUPP,VERSION=FINN4
*
*********************************************************
         TYPES TO SEARCH FOR INCLUDES
*********************************************************
*
         FLMINCLS TYPES=(INCLUDE,@@FLMTYP)
*
*********************************************************

*  PARSE TRANSLATOR DEFINITION
*********************************************************
*
         FLMTRNSL   CALLNAM='FINNOGA PARSER',                          C
               FUNCTN=PARSE,                                           C
               COMPILE=FINPARSE,                                       C
               DSNAME=SCLM.PROJDEFS.LOAD,                              C
               PORDER=1,                                               C
               OPTIONS=(@@FLMSIZ,@@FLMSTP,@@FLMLIS)
*
*   -- SOURCE --
*
           FLMALLOC  IOTYPE=A,DDNAME=SOURCE
            FLMCPYLB  @@FLMDSN(@@FLMMBR)
*
*********************************************************
*  BUILD TRANSLATOR DEFINITION
*********************************************************
*
* PREPROCESSOR STEP
*
         FLMTRNSL   CALLNAM='PANDA U PREP',                            C
               FUNCTN=BUILD,                                           C
               COMPILE=PANDA01,                                        C
               GOODRC=0,                                               C
               PORDER=1,                                               C
               OPTIONS='NOTRACE'
*
*   -- SOURCE
*
           FLMALLOC IOTYPE=S,KEYREF=SINC,DDNAME=SYSIN
*
*   -- PREPROCESSED SOURCE
*
           FLMALLOC IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000,           C
               DDNAME=SYSOUT
*
*   -- LISTING
*
           FLMALLOC  IOTYPE=O,KEYREF=OUT1,RECFM=VBA,LRECL=125,        C
               RECNUM=5000,PRINT=Y,DFLTTYP=PUPLIST,DDNAME=SYSPRINT
*
* COMPILE STEP
*
         FLMTRNSL   CALLNAM='FINNOGA 4',                               C
               FUNCTN=BUILD,                                           C
               COMPILE=FNGAA40,                                        C
               GOODRC=0,                                               C
               PORDER=3,                                               C
               OPTIONS='SOURCE,NOMACRO,OBJ(@FLMMBR)',                  C
               PARMKWD=PARM1
```

Figure 29. Finnoga/PUPP Language Definition (Part 1 of 2)

```
*
*   -- (1) OBJECT
*
           FLMALLOC IOTYPE=P,KEYREF=OBJ,DFLTTYP=OBJ,RECFM=FB,          C
               LRECL=80,RECNUM=5000
*
*   -- (2) NOT USED
*
           FLMALLOC IOTYPE=N
*
*   -- (3) NOT USED
*
           FLMALLOC IOTYPE=N
*
*   -- (4) INCLUDE LIBRARIES
*
           FLMALLOC IOTYPE=I,KEYREF=SINC
*
*   -- (5) SOURCE
*
           FLMALLOC IOTYPE=U,DDNAME=SYSOUT
*
*   -- (6) LISTING
*
           FLMALLOC  IOTYPE=O,KEYREF=LIST,RECFM=VBA,LRECL=125,         C
               RECNUM=5000,PRINT=Y,DFLTTYP=FINLIST
*
*   -- (7) NOT USED
*
           FLMALLOC IOTYPE=N
*
*   -- (8) FINNOGA COMPILER LIBRARIES
*
           FLMALLOC IOTYPE=A
            FLMCPYLB SYS1.FINNOGA.LIB
*
*   -- (9) NOT USED
*
      FLMALLOC IOTYPE=N
*
*   -- (10) WORK FILE
*
      FLMALLOC IOTYPE=W,LRECL=4000,RECFM=F,RECNUM=4000
*
*   -- (11) WORK FILE
*
      FLMALLOC IOTYPE=W,LRECL=4000,RECFM=F,RECNUM=4000
*
*5665-402 (C) COPYRIGHT IBM CORP 1980, 1989
```

Figure 29. Finnoga/PUPP Language Definition (Part 2 of 2)

The following example illustrates an architecture definition to build a program using two translators:

```
SINC    PROG7  SOURCE
OBJ     PROG7  OBJ
LIST    PROG7  FINLIST
OUT1    PROG7  PUPLIST
PARM1   NOOPTIMIZE
```

Figure 30. Architecture Definition Example

The only difference between this archdef and the Finnoga 4 CC archdef is the presence of the OUT1 keyword. This keyword specifies the type and member into which the PUPP listing is saved. In addition to specifying the OUT1 keyword in their archdefs, users who use this language definition to build their Finnoga 4 source must also remember to specify the language name FINPUPP for that Finnoga 4 source in the FLMLANGL macro statement.

Converting JCL to SCLM Language Definitions

Many sites use Job Control Language (JCL) to run preprocessors, compilers, linkage editors, and other tools used in the development process. SCLM supports developers and project managers through the use of language definitions that tell SCLM how to parse, build, and promote members of an SCLM-controlled data set. Language definitions can also specify additional translators to execute for the COPY, PURGE, and VERIFY functions. Because the SCLM language definitions provide an easier method of implementing processing control than JCL does, many sites have found it beneficial to convert their JCL to SCLM language definitions. To ease the conversion process, SCLM provides sample language definitions that you can tailor to the special needs of your site.

This section explains how to construct SCLM language definitions to replace existing JCL decks. Examples illustrate the basic principles underlying a successful migration from JCL to SCLM and also demonstrate methods for avoiding potential problems and conflicts.

Before You Begin

Before you attempt to convert your existing JCL decks to SCLM language definitions, you must obtain and review ″expanded″ listings of the JCL. The ″expanded JCL″ listings allow you to determine the actual values of the symbolic parameters in the JCL; these values include data set names, options, and other information that is required for successful translation to an SCLM language definition. In addition, you will need to know the order in which programs are executed in the JCL, and the condition codes that are expected from each program. Your system administrator should be able to help you locate the information that you need.

You should also review the information provided about SCLM macros in *ISPF Software Configuration and Library Manager (SCLM) Reference* paying special attention to the following macros and their parameters:

- FLMTRNSL
- FLMTCOND
- FLMALLOC
- FLMCPYLB
- FLMINCLS
- FLMTOPTS

Capabilities and Restrictions

There are two basic equivalencies that you will use to convert JCL cards to SCLM macro statements:

- Every JCL EXEC card with PGM=abc will correspond to an FLMTRNSL macro with COMPILE=abc in your language definition. Conditional execution of BUILD translators may be addressed through use of the FLMTCOND macro.

- Every JCL DD card will correspond to an FLMALLOC macro and/or an FLMSYSLB macro associated with an FLMALLOC macro in your language definition.

  In the case of STEPLIB, the JCL DD card will correspond to the DSNAME parameter in the FLMTRNSL macro. A STEPLIB concatenation of more than one data set would use the TASKLIB parameter. The TASKLIB parameter is set to the ddname associated with the data set concatenation. FLMCPYLBs are used to specify the data sets on an FLMALLOC macro with DDNAME set to the TASKLIB ddname. When both DSNAME and TASKLIB are specified, the DSNAME data set is searched first, followed by the TASKLIB data sets, followed by the system concatenation.

  In the case of SYSLIB-type ddnames for a compiler, the data sets must be specified FLMSYSLBs. Then either ALCSYSLB=Y must be specified on the FLMLANGL macro and/or FLMCPYLBs must be specified for the appropriate FLMALLOC macros. For an example of this, refer to the COBOL (FLM@COB2) or C/370 (FLM@C370) language definitions supplied with SCLM.

Three areas of restrictions can prevent a simple, one-to-one translation of JCL cards to SCLM macro statements:

- Backward referencing of data definition names (DDs)

  If a JCL DD card uses the "refer back" technique to reference a previous DD card (other than the card in the preceding step), or if a DD card refers to a data set using a ddname that differs from the data set's ddname in a prior step, conversion to an SCLM language definition can involve the use of an intermediate translator or a ddname substitution list in order to allocate the correct data set name for the program. (An intermediate translator is not needed if the succeeding translator supports DDNAME substitution lists; in this case, the succeeding translator can "hard code" the DDNAME and use IOTYPE=U on the FLMALLOC macro.)
- Complex conditional execution

  A JCL deck that specifies skipping all steps after a specified condition code from one or more previous steps is directly converted to appropriate FLMTRNSL macros with appropriate GOODRC values. Other conditional executions of BUILD translators can be addressed by using the FLMTCOND macro. For example, if the JCL is set up to run BUILD translator X if any previous return code is 4, but run Build translator Y if any previous return code is 8, you can use the FLMTCOND macro. FLMTCOND is only valid for use with BUILD translators. Conditional execution of non-BUILD translators can require modification of the translators or interface programs to handle the control of execution.
- TSO Address Space compatibility

  Some programs that run from JCL will not run in the TSO Address Space in which SCLM resides without a special interface translator. IBM has provided interface programs for several common IBM programs with this characteristic. For example, the FLMTMSI (SCRIPT), FLMTMJI (JOVIAL), and FLMTMMI (DFSUNUB0) translators all use the TSO Service Facility IKJEFTSR.

  If you have JCL that runs program XYZ without any errors, but fails when you try to run program XYZ from an FLMTRNSL macro, this may be the problem. You must write a translator to call the program using IKJEFTSR.

The following sections describe how to convert JCL cards and decks into functionally equivalent SCLM language definitions and provide suggested strategies for working around restrictions and conflicts.

Converting JCL Cards to SCLM Macro Statements

This section contains examples of JCL decks and their SCLM language definition equivalents.

Executing Programs

The SCLM FLMTRNSL macro is similar to a JCL EXEC (EXECUTE) card. Figure 31 shows a single JCL card that runs a program named IEFBR14.

```
//STEP1    EXEC    PGM=IEFBR14
```

Figure 31. JCL: Execute IEFBR14

Figure 32 shows an SCLM FLMTRNSL macro that performs the same task as the JCL card in Figure 31.

```
FLMTRNSL   COMPILE=IEFBR14,FUNCTN=BUILD,PORDER=0
```

Figure 32. SCLM: Execute IEFBR14

FLMTRNSL's COMPILE option specifies the name of the program to execute (IEFBR14.) The FUNCTN parameter specifies here that IEFBR14 will be invoked when the user requests a BUILD, and the PORDER value of 0 tells SCLM that neither an option list nor a ddname substitution list will be passed to IEFBR14.

Figure 33 is a slightly more complicated example. We want to use a translator program named GAC to copy the contents of TSOSCxx.DEV1.SOURCE(MEMBER1) into TSOSCxx.DEV1.LIST(MEMBER1). The GAC program itself requires a SYSIN data set, which is empty in this example.

```
//STEP1    EXEC    PGM=GAC
//SYSIN    DD      DUMMY
//INPUT    DD      DSN=TSOSCxx.DEV1.SOURCE(MEMBER1),DISP=SHR
//OUTPUT   DD      DSN=TSOSCxx.DEV1.LIST(MEMBER1),DISP=SHR
```

Figure 33. JCL: Execute GAC

Figure 34 shows the SCLM language definition that performs the same task as the JCL in Figure 33.

```
FLMTRNSL   COMPILE=GAC,FUNCTN=BUILD,PORDER=0
FLMALLOC   IOTYPE=A,DDNAME=SYSIN
 FLMCPYLB   NULLFILE
FLMALLOC   IOTYPE=A,DDNAME=INPUT
 FLMCPYLB   TSOSCxx.DEV1.SOURCE(MEMBER1)
FLMALLOC   IOTYPE=A,DDNAME=OUTPUT
 FLMCPYLB   TSOSCxx.DEV1.LIST(MEMBER1)
```

Figure 34. SCLM Language Definition: Execute GAC

As before, the FLMTRNSL macro is used to specify the name of the program to run. The FLMALLOC and FLMCPYLB statements allocate the existing data sets to ddnames.

Conditional Execution

In Figure 35, program XYZ runs only if the return code from program ABC is less than five.

```
//STEP1     EXEC    PGM=ABC
//STEP2     EXEC    PGM=XYZ,COND=(5,GE)
```

Figure 35. JCL: Conditional Execution

In SCLM, the GOODRC parameter on the FLMTRNSL macro allows you to specify return code values for conditional execution. In Figure 36, the GOODRC parameter for program ABC is set to 4. If ABC ends with a return code greater than four, processing ends; program XYZ will not execute.

```
FLMTRNSL   COMPILE=ABC,FUNCTN=BUILD,PORDER=0,GOODRC=4
FLMTRNSL   COMPILE=XYZ,FUNCTN=BUILD,PORDER=0
```

Figure 36. SCLM Language Definition: Conditional Execution

In Figure 37, program XYZ runs only if the return code from program ABC is less than 5. Program MBS is to execute after program XYZ regardless of the previous return codes.

```
//STEP1     EXEC    PGM=ABC
//STEP2     EXEC    PGM=XYZ,COND=(5,GE)
//STEP3     EXEC    PGM=MBS
```

Figure 37. JCL: Complex Conditional Execution

In SCLM, the GOODRC parameter on the FLMTRNSL macro specifies when to skip all remaining translators in the language definition. In Figure 38 the FLMTCOND macro is used so that execution may skip program XYZ but continue with program MBS.

```
FLMTRNSL   COMPILE=ABC,FUNCTN=BUILD,PORDER=0
FLMTRNSL   COMPILE=XYZ,FUNCTN=BUILD,PORDER=0
  FLMTCOND ACTION=SKIP,WHEN=(*,GE,5)
FLMTRNSL   COMPLIE=MBS,FUNCTN=BUILD,PORDER=0
```

Figure 38. SCLM Language Definition: Complex Conditional Execution

Sample JCL Conversion

This section contains commented sample JCL and language definitions that perform the same tasks: invoking the CICS preprocessor and then invoking the OS COBOL compiler to produce an object module. Figure 39 on page 123 contains the JCL used to accomplish these tasks; Figure 40 on page 125 contains the equivalent SCLM language definition. Each sample contains comments with step numbers. The step descriptions that follow relate a line or command from the JCL to the equivalent SCLM language definition macro, option, or command.

1. The JCL has a job step named TRN, which is the first translator called in this job.

   SCLM uses an FLMTRNSL macro to call this translator. This is the first FLMTRNSL macro for build in the language definition.

2. Job step TRN executes a program called DFHECP$1, the CICS preprocessor for OS COBOL.

SCLM uses the COMPILE=DFHECP$1 statement on the FLMTRNSL macro.

3. The STEPLIB line in job step TRN tells the job where to find the program DFHECP$1.

   SCLM uses the DSNAME option on the FLMTRNSL macro. Both the STEPLIB and DSNAME point to the same data set, CICS.V3R2M1.SDFHLOAD.

4. The SYSIN statement defines the data set that contains the member to compile.

   SCLM uses an FLMALLOC macro to allocate the SYSIN data set to a ddname for the CICS preprocessor. Because we are using PORDER=1, the FLMALLOC macro assigns the ddname, SYSIN, that the CICS preprocessor is expecting.

5. The TRN job step sends the preprocessor listing to the printer using the SYSPRINT statement.

   SCLM uses an FLMALLOC macro to allocate an output data set to the ddname SYSPRINT.

6. The SYSPUNCH line in the TRN step creates the output of the CICS preprocessor and passes it to the next job step (COB) as a temporary file.

   SCLM uses an FLMALLOC macro with IOTYPE=W to allocate a work (temporary) file with the ddname of SYSPUNCH. This work file is passed to the next job step (FLMTRNSL).

7. The JCL has a job step named COB, which is the second translator called in this job.

   SCLM uses an FLMTRNSL macro to call this translator. This is the second FLMTRNSL macro for build in our language definition.

8. The job step COB executes (EXEC PGM=) a program called IKFCBL00, the compiler for OS COBOL.

   SCLM uses the COMPILE=IKFCBL00 statement on the FLMTRNSL macro.

9. To pass compiler options to the OS COBOL compiler, the COB job step uses a PARM= command.

   SCLM uses the OPTIONS= statement on the FLMTRNSL macro to perform the same task.

10. This job has conditional execution for the COB step via the COND(5,GE) JCL command. The COB step will not execute if the return code of the TRN step is greater than 4.

    SCLM sets the GOODRC keyword parameter for the TRN step (CICS preprocessor) equal to 4. Build halts execution of all translators following the TRN step in the language definition if the return code from the TRN step is greater than 4.

11. The STEPLIB statement in job step COB tells the job where to find the program IKFCBL00.

    SCLM uses the DSNAME= option on the FLMTRNSL macro. Both the STEPLIB and DSNAME point to the same data set, IKF.V1R2M4.VSCOLIB.

12. The SYSLIB statement in job step COB tells the job where to find the system type includes.

    The language definition uses the FLMSYSLB macro with IOTYPE=I and the FLMINCLS macro to do the same task.

    SCLM allocates these project data sets allocated for IOTYPE=I before the data sets on the FLMCPYLB macro(s). ALCSYSLB=Y parameter must be specified on the FLMLANGL macro to ensure that the FLMSYSLB data sets are allocated to the IOTYPE=I ddnames.

Because PORDER=3 is being used, the SYSLIB DD is the fourth ddname passed to the compiler in a ddname substitution list. The COBOL compiler uses the fourth ddname as SYSLIB no matter what value is assigned to the DDNAME keyword parameter on the FLMALLOC macro.

13. For each system library specified for the SYSLIB DD, the language definition has an FLMSYSLB macro. In this case both CICS.V3R2M1.SDFHCOB and CICS.V3R2M1.SDFHMAC are specified.
14. The COB job step sends the compile listing to the printer using the SYSPRINT statement.

    SCLM uses an FLMALLOC macro to allocate an output data set to the ddname SYSPRINT.
15. In the COB job step, the SYSIN DD statement identifies the data set that contains the member to compile. This is the output of the CICS preprocessor step TRN.

    SCLM uses an FLMALLOC macro with IOTYPE=U to refer to a ddname from a prior step. The language definition instructs MVS to allocate the data set assigned in the TRN step to the ddname SYSPUNCH.
16. The SYSLIN statement in the COB step identifies the output data set for object code created by the COBOL compiler.

    The language definition uses an FLMALLOC macro with IOTYPE=O to allocate an output file. This FLMALLOC macro is the first in the COB FLMTRNSL because when using PORDER=3, the OS COBOL compiler expects the output data set ddname to be first in a ddname substitution list.
17. The COB step allocates SYSUT1 as a temporary work file for the OS COBOL compiler.

    SCLM's language definition uses an FLMALLOC macro with IOTYPE=W to perform the same task. This must be the eighth file provided to the OS COBOL compiler because PORDER=3 tells SCLM that we are using a ddname substitution list.
18. The COB step allocates SYSUT2 as a temporary work file for the OS COBOL compiler.

    SCLM's language definition uses an FLMALLOC macro with IOTYPE=W to perform the same task. This must be the ninth file provided to the OS COBOL compiler because we are using a ddname substitution list.
19. The COB step allocates SYSUT3 as a temporary work file for the OS COBOL compiler.

    SCLM's language definition uses an FLMALLOC macro with IOTYPE=W to perform the same task. This must be the tenth file provided to the OS COBOL compiler because we are using a ddname substitution list.
20. The COB step allocates SYSUT4 as a temporary work file for the OS COBOL compiler. SCLM's language definition uses an FLMALLOC macro with IOTYPE=W to perform the same task. This must be the eleventh file provided to the OS COBOL compiler because we are using a ddname substitution list.
21. The COB step allocates SYSUT5 as a temporary work file for the OS COBOL compiler.

    SCLM's language definition uses an FLMALLOC macro with IOTYPE=W to perform the same task. This must be the twelfth file provided to the OS COBOL compiler because we are using a ddname substitution list.
22. **SCLM language definition only**

    The language definition uses PORDER=3 for the OS COBOL compiler step (COB) to use a ddname substitution list. A ddname substitution list provides

an ordered list (defined by the translator) of ddnames such that the position of a ddname in the list, and not the actual ddname, is used by the translator for a specific file.

The input file for the compiler must be the output file from the CICS preprocessor. The ddname assigned in the TRN step is SYSPUNCH. Because this file has already been allocated to SYSPUNCH, another way (besides ddname) is needed to pass this file as the input to the compiler. By using PORDER=3, SCLM passes all the files that can be used by the OS COBOL compiler in the order specified for this compiler. To use PORDER=3, a specific parameter string must be built. The language definition must have an FLMALLOC macro for each of these parameters.

Those FLMALLOCs that are tagged for STEP 22 are not applicable for the OS COBOL compiler. SCLM places 8 bytes of hexadecimal zeros into the ddname substitution list for each FLMALLOC with IOTYPE=N.

```
//USERIDC  JOB (AS05CR,T12,C531),'USERID',NOTIFY=USERID,CLASS=A,
//    MSGCLASS=O,MSGLEVEL=(1,1)
//*
//*      THIS PROCEDURE CONTAINS 2 STEPS
//*      1.   EXEC THE CICS PREPROCESSOR
//*      2.   EXEC THE OS/VS COBOL COMPILER
//*
//*      CHANGE THE JOB NAME AND THE ACCOUNTING INFORMATION TO MEET THE
//*      REQUIREMENTS OF YOUR INSTALLATION.
//*
//*      CHANGE 'PROGNAME' TO THE NAME OF THE CICS/COBOL PROGRAM YOU
//*      WANT TO COMPILE.  CHANGE 'USERID' TO YOUR USERID.
//*
//*      CHANGE 'DEVLEV' TO THE GROUP THAT CONTAINS THE PROGRAM TO BE COMPILED.
//*
//* STEP 1: TRN STATEMENT; STEP 2: EXEC PGM STATEMENT
//*
//TRN    EXEC PGM=DFHECP1$,
//*
//* STEP 3: STEPLIB STATEMENT
//*
//       REGION=2048K
//STEPLIB  DD  DSN=CICS.V3R2M1.SDFHLOAD,DISP=SHR//*
//*
//* STEP 4: SYSIN STATEMENT
//*
//SYSIN    DD  DSN=USERID.DEVLEV.SOURCE(PROGNAME),DISP=SHR
//*
//* STEP 5: SYSPRINT STATEMENT
//*
//SYSPRINT DD  SYSOUT=A
//*
//* STEP 6: SYSPUNCH STATEMENT
//*
//SYSPUNCH DD  DSN=&&SYSCIN,;
//             DISP=(,PASS),UNIT=SYSDA,
//             DCB=BLKSIZE=400,
//             SPACE=(400,(400,100))
//*
//* STEP 7: COB STATEMENT;  STEP 8:  EXEC PGM STATEMENT
//* STEP 9: PARM STATEMENT; STEP 10: COND STATEMENT
//*
//COB    EXEC PGM=IKFCBL00,REGION=2048K,COND=(5,GE),
//       PARM='NOTRUNC,NODYNAM,LIB,SIZE=256K,BUF=32K,APOST,DMAP,XREF'
//*
//* STEP 11: STEPLIB STATEMENT
//*
//STEPLIB  DD  DSN=IKF.V1R2M4.VSCOLIB,DISP=SHR
//*
```

Figure 39. JCL: Invoke COBOL Preprocessor and Compiler (Part 1 of 2)

```
//* STEP 12: SYSLIB STATEMENT; STEP 13: DD STATEMENT
//*
//SYSLIB   DD  DSN=CICS.V3R2M1.SDFHCOB,DISP=SHR
//         DD  DSN=CICS.V3R2M1.SDFHMAC,DISP=SHR
//*
//* STEP 14: SYSPRINT STATEMENT
//*
//SYSPRINT DD  SYSOUT=O
//*
//* STEP 15: SYSIN STATEMENT
//*
//SYSIN    DD  DSN=&&SYSCIN,DISP=(OLD,DELETE)
//*
//* STEP 16: SYSLIN STATEMENT
//*
//SYSLIN   DD  DSN=USERID.DEVLEV.OBJ(PROGNAME),DISP=SHR
//*
//* STEP 17: SYSUT1 STATEMENT
//*
//SYSUT1   DD  UNIT=SYSDA,SPACE=(460,(350,100))
//*
//* STEP 18: SYSUT2 STATEMENT
//*
//SYSUT2   DD  UNIT=SYSDA,SPACE=(460,(350,100))
//*
//* STEP 19: SYSUT3 STATEMENT
//*
//SYSUT3   DD  UNIT=SYSDA,SPACE=(460,(350,100))
//*
//* STEP 20: SYSUT4 STATEMENT
//*
//SYSUT4   DD  UNIT=SYSDA,SPACE=(460,(350,100))
//*
//* STEP 21: SYSUT5 STATEMENT
//*
//SYSUT5   DD  UNIT=SYSDA,SPACE=(460,(350,100))
```

Figure 39. JCL: Invoke COBOL Preprocessor and Compiler (Part 2 of 2)

```
*********************************************************************
*          SCLM LANGUAGE DEFINITION FOR
*        OS COBOL WITH CICS PREPROCESSOR 3.2.1
*
* CICS OUTPUT IS PASSED VIA THE CICSTRAN DD ALLOCATION TO OS COBOL.
*
* POINT THE FLMSYSLB MACRO(S) AT ALL 'STATIC' COPY DATASETS.
* CUSTOMIZE THE 'OPTIONS' AND 'GOODRC' FIELDS TO YOUR STANDARDS.
* ADD THE 'DSNAME' FIELD IF THE TRANSLATOR IS IN A PRIVATE LIBRARY.
* WHEN A NEW TRANSLATOR VERSION REQUIRES TOTAL RECOMPILATION FOR THIS
* LANGUAGE, THE 'VERSION' FIELD ON FLMLANGL SHOULD BE CHANGED.
*********************************************************************
*
COBCICS  FLMSYSLB  CICS.V3R2M1.SDFHCOB
*
         FLMLANGL  LANG=COBCICS,VERSION=CICS321,ALCSYSLB=Y
*
* PARSER TRANSLATOR
*
          FLMTRNSL   CALLNAM='SCLM COBOL PARSE',                      C
               FUNCTN=PARSE,                                          C
               COMPILE=FLMLPCBL,                                      C
               PORDER=1,                                              C
               OPTIONS=(@@FLMLIS,@@FLMSTP,@@FLMSIZ,)
*          (* SOURCE      *)
          FLMALLOC  IOTYPE=A,DDNAME=SOURCE
           FLMCPYLB @@FLMDSN(@@FLMMBR)
*
* BUILD TRANSLATORS
*         - CICS PRECOMPILE - STEP NAME TRN
*
* STEP 1
         FLMTRNSL   CALLNAM='CICS PRE-COMPILE',                       C
               FUNCTN=BUILD,                                          C
* STEP 2
               COMPILE=DFHECP1$,                                      C
* STEP 3  (* STEPLIB *)
               DSNAME=CICS.V3R2M1.SDFHLOAD,                           C
               VERSION=2.1,                                           C
* STEP 10 (* COND *)
               GOODRC=4,                                              C
               PORDER=1
* STEP 4  (* SYSIN *)
          FLMALLOC  IOTYPE=S,KEYREF=SINC,RECFM=FB,LRECL=80,           C
               DDNAME=SYSIN
* STEP 5   (* SYSPRINT *)
          FLMALLOC  IOTYPE=O,RECFM=FBA,LRECL=121,                     C
               RECNUM=35000,PRINT=Y,DDNAME=SYSPRINT
*
* STEP 6   (* SYSPUNCH *)
          FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,                       C
               RECNUM=5000,DDNAME=SYSPUNCH
*
* STEP 7   (*COBOL INTERFACE - STEP NAME COB *)
* STEP 8
         FLMTRNSL   CALLNAM='COBOL COMPILE',                          C
               FUNCTN=BUILD,                                          C
               COMPILE=IKFCBL00,                                      C
* STEP 11  (* STEPLIB *)
               DSNAME=IKF.V1R2M4.VSCOLIB,                             C
               VERSION=1.0,                                           C
               GOODRC=4,                                              C
* STEP 22
               PORDER=3,                                              C
```

Figure 40. SCLM Language Definition: Invoke COBOL Preprocessor and Compiler (Part 1 of 2)

```
* STEP 9   (* PARMS *)
              OPTIONS=(NOTRUNC,NODYNAM,LIB,SIZE=256K,BUF=32K,APOST,   C
              DMAP,XREF)* DDNAME ALLOCATIONS
* STEP 16
*  1        (* SYSLIN *)
            FLMALLOC  IOTYPE=O,KEYREF=OBJ,RECFM=FB,LRECL=80,           C
              RECNUM=5000,DFLTTYP=OBJ
* STEP 22
*  2        (* N/A *)
            FLMALLOC  IOTYPE=N
* STEP 22
*  3        (* N/A *)
            FLMALLOC  IOTYPE=N
* STEP 12; STEP 13
*  4        (* SYSLIB *)
            FLMALLOC  IOTYPE=I,KEYREF=SINC
* STEP 15
*  5        (* SYSIN *)
            FLMALLOC  IOTYPE=U,KEYREF=SINC,DDNAME=SYSPUNCH
* STEP 14
*  6        (* SYSPRINT *)
            FLMALLOC  IOTYPE=O,KEYREF=LIST,RECFM=FBA,LRECL=133,        C
              RECNUM=25000,PRINT=Y,DFLTTYP=LIST
* STEP 22
*  7        (* SYSPUNCH *)
            FLMALLOC  IOTYPE=N
* STEP 17
*  8        (* SYSUT1 *)
            FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000
* STEP 18
*  9        (* SYSUT2 *)
            FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000
* STEP 19
* 10        (* SYSUT3 *)
            FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000
* STEP 20
* 11        (* SYSUT4 *)
            FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000
* STEP 22
* 12        (* SYSTERM *)
            FLMALLOC  IOTYPE=N
* STEP 21
* 13        (* SYSUT5 *)
            FLMALLOC  IOTYPE=A
             FLMCPYLB NULLFILE
* STEP 22
* 14        (* SYSUT6 *)
            FLMALLOC  IOTYPE=N

* 5665-402 (C) COPYRIGHT IBM CORP 1980, 1989
```

Figure 40. SCLM Language Definition: Invoke COBOL Preprocessor and Compiler (Part 2 of 2)

Note: For reference purposes, the language definition shown in Figure 40 contains comments with step numbers placed in the middle of commands; for this language definition to run, these comments must be removed.

Chapter 6. Using SCLM and Tivoli Service Desk for OS/390

The Tivoli Service Desk for OS/390 (Service Desk) sample code, shipped as member FLM00CVE in SAMPLIB, illustrates communication between SCLM and Service Desk Version 1.2. The sample is implemented in the REXX programming language and uses the Service Desk REXX high-level API. The sample verifies a programmer's authority to update an SCLM-controlled module based on the SCLM change code provided by the programmer. FLM00CVE retrieves the Service Desk problem record identified by the change code, and verifies:

1. The record exists.
2. The **Problem Status** field is set to OPEN.
3. The **Assignee Name** field is the same as the userid parameter passed by SCLM.

Required Environment

- Tivoli Service Desk for OS/390 (formerly Information/Management) software Version 1.2 or above must be installed on the target MVS system.
- The Service Desk REXX HLAPI (BLGYRXM) must be installed on the system.
- A valid Service Desk session name, class name, and default REXX/HLAPI Record-Retrieve PIDT table must exist. The sample uses session BLGSES00, class MASTER, and table BLGYPRR.
- For software verification purposes, at least one problem record meeting the desired criteria should exist in the Service Desk database.

Description of User Program Interaction

The FLM00CVE REXX Exec can be invoked as a regular MVS Exec; however, it is designed to be invoked as an SCLM change code verification user exit. If invoked as a user exit, the Service Desk specific arguments are passed by the SCLM option list defined in the FLMCNTRL macro and the SCLM-specific arguments are appended to the Service Desk arguments.

Input Parameters

Two different sets of parameters are passed to the sample as one parameter string. User options are specified in the Options entry of the FLMCNTRL macro. SCLM parameters are the standard set of parameters passed to the SCLM Exit.

Option List Format

The option list format is as follows:

```
pica_tabn,
pica_clsn,
pica_sess,
pica_clsc,
pica_dbid,
pica_msgd,
pica_spli,
pica_stxt,
pica_tint,
pica_usrn,
group,
type,
member,
```

```
language,
userid,
auth code,
change code
```

Service Desk Parameters

The required Service Desk parameters are:

pica_tabn
specifies the name of the Service Desk Record Retrieval table. The table defines the fields within a problem record. The default is BLGYPRR (shipped with Service Desk). This must be the name of the table used in your installation.

pica_clsn
specifies the Service Desk Privilege Class record that contains the registered user name authorized to retrieve a problem record. The default is MASTER. This must match your installation. The registered authorized user name is optionally specified in option 10 (see 128).

pica_sess
specifies the name of the Service Desk Session Member (BLGSESxx) load module. the default is BLGSES00. This parameter must match your installation.

The optional Service Desk parameters are:

pica_clsc
specifies the count of pribilege class records that can be maintained in storage during the Service Desk session. The default is **one**, the sample program uses only one privilege class record.

pica_dbid
specifies the Service Desk Problem Record Database number. The default is **5**, the standard Service Desk database.

pica_msgd
specifies the destination for Service Desk API log messages. Messages can be either printed to an APIPRINT data set, returned on the message chain, or both. The default is **C**, return messages on the API message chain. The sample program interprets chained message return code and reason code values to provide English text messages. See “Error Processing” on page 130 for more information.

pica_spli
specifies the number of minutes that the activity log can print transaction results before the API closes and reopens the log. The default is **ten minutes** if message chaining (pica_msgd) is ***not*** selected. Otherwise, it is zero.

pica_stxt
specifies whether text data is to be retrieved from the problem record. Setting this value to NO suppresses text retrieval. The default is **NO** because the sample program does not process text fields in the problem record.

pica_tint
specifies the transaction processing timeout interval. This field specifies the time in seconds that any Service Desk API transaction can process before the API notifies the application of a timeout event. The default is **300** seconds.

pica_usrn
specifies a name registered in the selected Privilege Class (see 128) that is authorized to retrieve problem records. The default is the TSO UserID of the SCLM user.

SCLM Parameters

The SCLM parameters are:

group specifies the MVS data set Group name.

type specifies the MVS data set Type name.

member
specifies the MVS partitioned data set Member name if selected, otherwise blank.

language
specifies the language of the module selected. This is blank for Edit exits.

userid specifies the TSO User ID accessing SCLM. In the sample program, this value is compared to the Service Desk Problem Record Assignee Name field (Service Desk S-word: S0B5A) for authorization to modify the SCLM module.

auth_code
specifies the authorization code of the member being edited.

change code
specifies the Change Code entered by the SCLM user on the appropriate panel. This value is used by the sample program to specify the Service Desk Problem Record Record_ID (RNID) to be retrieved. In the sample program, the Problem Record Current Status field (Service Desk S-word: S0BEE) from the retrieved record is verified against the constant **OPEN** for authorization to modify the SCLM module.

Program Flow

When the FLM00CVE program is invoked, the program flow is as follows:

1. Parse the argument string passed by invocation.
2. Perform the REXX/HLAPI Initialization function (HL01).
3. Perform the REXX/HLAPI Record Retrieve function (HL06).
4. Perform the REXX/HLAPI Termination function (HL02).
5. Verify that the user requesting to change the member has authority to do so based on information contained in the retrieved record.
6. Output error messages if applicable.
7. Return to caller passing return code as exit value.

Each of the steps above performs error-checking and return code analysis independently. If an error is noted, processing might terminate at that time or continue to another step. For example, after Service Desk initialization has completed, Service Desk Termination is attempted regardless of intervening errors, the transaction is not left hanging.

Error Processing

When an error condition is encountered, the program issues an error message, if possible, and terminates processing with the appropriate return code. When a warning condition is encountered, the program issues a warning message and continues processing. When a warning or error is the result of a Service Desk REXX/HLAPI call, a message appropriate to the reason code is displayed. If a Service Desk message chain is available, the associated messages are also displayed.

The program initiates REXX/HLAPI with logging enabled. Error conditions are both printed to the session log and returned to the program in message chains, as appropriate.

For warning message instigated by the Service Desk API interface, the program returns a return code of zero because SCLM considers any non-zero return code as an indication of failure. For API errors with return code 8 or higher, the program issues the appropriate messages and return code 8.

The program specifically tests for and reports the following input parameter errors:

- No input parameters.
- Missing or invalid REXX/HLAPI table name.
- Missing or invalid Service Desk Class name.
- Missing or invalid Service Desk Session ID.
- Missing or invalid User ID.
- Missing or invalid Change Code.
- Problem Record not found in the database.
- Problem Record Problem Status not ″OPEN″.
- Problem Record Assignee Name does not match User ID.
- Input parameters specified as ″Ignored″ are checked for presence and valid format, and a warning message is issued if warranted. However, the return code presented is zero.

Example

This example calls the FLM00CVE Exec through the SCLM verify change code exit.

```
IN FLMCNTRL MACRO:
  CCVFY=FLM00CVE,
  CCVFYDS=PROJ1.SAMPLIB.EXEC,
  CCVFYCM=TSOLNK,
  CCVFYOP=(BLGYPRR,MASTER,BLGSES00,1,5,C,300,NO,360,FLM00CVE,)
```

Where:

CCVFY=FLM00CVE
specifies that the SCLM Verify Change Code exit be used and that member FLM00CVE be invoked.

CCVFYDS=PROJ1.SAMPLIB.EXEC
specifies the MVS data set containing member FLM00CVE. In the example: ″PROJ1.SAMPLIB.EXEC(FLM00CVE)″

CCVFYCM=TSOLNK
specifies that FLM00CVE is invoked using the TSO service facility routine, the default for REXX Exec programs.

CCVFYOP=(exit routine parameters)
specifies the parameters that are passed to the exit program.

Chapter 7. Understanding and Using the Customizable Parsers

Parsers are provided as source code (in REXX) for those customers who need to extend or modify the behavior of the parsers supplied by IBM. This section explains the logic of the parsers as shipped and provides examples of how to modify the parsers to suit your own needs and standards.

The customizable parsers supplied by IBM are:

FLMLRASM Assembler H parser

FLMLRCBL COBOL II parser

FLMLRCIS C/C++ for MVS parser

FLMLRC2 C++ for Windows parser

FLMLRC37 C/370 parser

FLMLRDTL DTL parser

FLMLRIPF OS/2 IPF parser

These parsers can be found in the ISPF sample library, ISP.SISPSAMP.

The Parsers as Shipped

The IBM-supplied parsers are delivered as REXX source. If you do not require any changes to the functions provided, the source modules can be used. The parsers may also be compiled, pre-linked, and link edited (using the IBM Compiler and Library for REXX/370 and the Linkage Editor) for optimum performance.

Use the `CALLMETH=TSOLNK` parameter on the FLMTRNSL macro to directly invoke SCLM translators written in REXX.

Sample Language Definitions

The sample language definitions are provided to demonstrate how to invoke the customizable parsers:

FLM@RASM Assembler H sample language definition

FLM@RCBL COBOL II sample language definition

FLM@RCIS C/370 sample language definition

FLM@RC37 C/370 sample language definition

FLM@DTLC DTL sample language definition

FLM@WBCC C++ for Windows sample language definition

FLM@WIPF OS/2 Help sample language definition

In addition, a sample REXX language definition, FLM@REXC, is provided to compile, pre-link, and link edit REXX source code.

Parser Error Listings

For parsing errors with return codes of 4, 8, or 10, the parsers write error messages to a data set called *userid*.SCLMERR.LISTING. An error message consists of two or three lines. The first line is the error code: 4, 8, or 10. The second line and the third line (if it exists) contain one of the following:

- One or more non-valid input parameters
- A dependency name that is greater than 8 characters in length
- A dependency name that cannot be stored in the dependency buffer because it is full
- A line of source containing an error
- A single quote or double quote that is mismatched and its line number

For additional information, refer to *ISPF Software Configuration and Library Manager (SCLM) Reference*

Modifying the Parsers

This section describes the general design of the customizable parsers and provides several examples of updating the parsers.

The parsers read each line of the source code and process tokens on each line. Tokens are discrete elements on a line of source code; they are language-dependent. For example, consider the following COBOL statement:

```
MOVE 'SMITH' TO NAME.
```

Seven tokens appear in this example: `MOVE`, the two single quotation marks, `SMITH`, `TO`, `NAME`, and the period.

State variables are used to hold the current conditions and expectations created by the processing of prior tokens in order to process the current token. For example, if a single quote is found, the single quote state variable (`state.single`) is turned on. All tokens, regardless of multiple lines, are ignored until the matching single quote is found, or until the end of file is reached. In the COBOL and Assembler parsers, dependency names may be enclosed in quotes; all data after the dependency name is ignored until the matching quote is found. Dependency keywords (`COPY` or `EXEC SQL INCLUDE`) inside quotes are ignored. For example, consider the following COBOL statement:

```
MOVE 'COPY B' TO ACTION.
```

`B` will not be placed into the dependency buffer because `COPY` will not be processed as a dependency keyword.

Because of these state variables, dependencies, comments (in C/370), quotes, and so on can span lines. Concatenation of keywords and dependency names (particularly in COBOL) is not supported by the parsers. If dependency names are split between lines, the partial dependency name will not be added by the REXX parser.

Adding More Elaborate Parsing Error Messages

This section provides an example of modifying a customizable parser to add more complete error messages to the *userid*.SCLMERR.LISTING data set. This support can be added to all three of the REXX parsers. The COBOL parser, FLMLRCBL, will be used in this example.

The **error_listing** routine is used to place the error_string1 and error_string2 strings into the error messages data set. error_string1 and error_string2 are set before invoking **error_listing**. The following list identifies, in order, the routine, the expanded English error message, and the error string to be changed in FLMLRCBL.

Routine **Change Required**

initialization Change:

```
error_string1 = miss_parm1 ' ' ||,
                miss_parm2 ' ' ||,
                miss_parm3
```

to

```
error_string1 = 'MISSING PARAMETER(S): ' ||,
                miss_parm1 ' ' ||,
                miss_parm2 ' ' ||,
                miss_parm3
```

initialization Change:

```
error_string1 = 'LISTSIZE=',
                ||sclm_dep_array_size
error_string2 = '   LISTSIZE < ',
                DEP_ELEM_SIZE
```

to

```
error_string1 = 'LISTSIZE PARAMETER MUST BE AT LEAST',
                DEP_ELEM_SIZE
error_string2 = '
```

initialization Change:

```
error_string1 = 'LISTSIZE=',
                ||sclm_dep_array_size
```

to

```
error_string1 = 'LISTSIZE PARAMETER MUST BE A '||,
                'POSITIVE WHOLE NUMBER'
```

initialization Change:

```
error_string1 = 'LISTINFO=',
                ||sclm_dep_addr
```

to

```
error_string1 = 'LISTINFO PARAMETER MUST BE A '||,
                'POSITIVE WHOLE NUMBER'
```

initialization Change:

```
error_string1 = 'STATINFO=',
                ||sclm_stats_addr
```

to

```
error_string1 = 'STATINFO PARAMETER MUST BE A '||,
                'POSITIVE WHOLE NUMBER'
```

process_line Change:

```
error_string1 = token
```

to

```
error_string1 = 'DEPENDENCY 'token' EXCEEDS 8 '||,
                'CHARACTERS ON LINE '||,
                stats.total_lines
```

add_dep Change:

```
error_string1 = name
```

to

```
error_string1 = 'DEPENDENCY ARRAY CAPACITY EXCEEDED '||,
                'WITH DEPENDENCY 'name
```

termination Change:

```
error_string1 = SINGLE_QUOTE state.single_line
```

to

```
error_string1 = 'MISMATCHED SINGLE QUOTE ON ' state.single_line
```

termination Change:

```
error_string1 = DOUBLE_QUOTE state.double_line
```

to

```
error_string1 = 'MISMATCHED DOUBLE QUOTE ON ' state.double_line
```

termination Change:

```
error_string1 = END_KEYWORD
```

to

```
error_string1 = 'DEPENDENCY ARRAY CAPACITY EXCEEDED,'
error_string2 = 'NOT ENOUGH SPACE TO WRITE END-OF-LIST KEYWORD'
```

Appending to the Error Listing File

If parser errors are found, error messages are written to the *userid*.SCLMERR.LISTING data set. This data set is created (re-written) each time an error is found, each time one of the REXX parsers is invoked. The **allocate_error_listing** routine is used to allocate this data set. The overwriting of this data set is suitable for creating or modifying members with Edit. However, during multiple migrations of members, this data set will be overwritten each time a parser error occurs per parser invocation.

In order to keep all parser errors for all members, modify the **allocate_error_listing** routine to append to the `userid`.SCLMERR.LISTING data set, instead of overwriting it. Change

```
IF SYSDSN(ERRFILE) = 'OK' THEN
   disp = 'OLD'
ELSE
```

to

```
IF SYSDSN(ERRFILE) = 'OK' THEN
   disp = 'MOD'
ELSE
```

With this change, all invocations of the parser will append any error messages to the error file without overwriting the previous contents.

Compiling the Parsers

To increase parser performance, the REXX parsers can be compiled and pre-linked using the IBM Compiler and Library for REXX/370. Using the FLM@REXC language definition, SCLM can be used to compile, pre-link, and link edit the parsers. To compile a parser using FLM@REXC:

1. Add FLM@REXC to your SCLM project definition.
2. Make any required changes to FLM@REXC, such as changing specified data set names.
3. Re-assemble and re-link the project definition.
4. Migrate the parsers into SCLM using the `REXXCOM` language.
5. Build each of the parsers.
6. If necessary, copy the load modules (FLMLRASM, FLMLRCBL, FLMLRC37, FLMLRCIS, FLMLRC2, FLMLRDTL, and/or FLMLRIPF) to common data sets.
7. Change the language definitions to use the load modules instead of the interpreted versions.

   Remember to change the CALLMETH parameter on the FLMTRNSL macro.
8. Re-assemble and re-link the project definition.

Part 2. Developer's Guide

Chapter 8. The Software Configuration and Library Manager

The Software Configuration and Library Manager (SCLM) component of ISPF contains the capabilities of both a Library Manager and a Configuration Manager program.

Library Manager programs control source code, keeping developers from accidentally overwriting each other's code changes and providing a mechanism for moving the source code from one set of development libraries to the next. Also, SCLM can keep back-level versions of source files, with an audit trail of changes and other basic library management functions that you can use in your code development and maintenance processes.

Configuration Manager programs track how all the pieces of an application fit together. Not just the source code, but the object and load modules as well. SCLM adds additional capabilities, such as how test cases and documentation are associated with a source code module. SCLM uses this information to control compiling, linking, and promoting an application. SCLM "builds" are optimized such that only pieces that need to be regenerated when a change is made are built.

SCLM Project Environment

The SCLM *project environment* is made up of data sets used by SCLM to store and control the user application software for an individual project. The project environment contains three types of data associated with an individual project:

- User Application Data
- SCLM Control Data (see "Step 6: Allocate and Create the Control Data Sets" on page 18)
- Project Definition Data (see "Chapter 1. Defining the Project Environment" on page 3)

User Application Data

User application data consists of the application data (programs) being developed for a single project. SCLM stores all user data associated with a single project as members within a hierarchical set of MVS partitioned data sets (ISPF libraries). These data sets are called the project partitioned data sets. Users refer to SCLM-controlled ISPF libraries with an SCLM naming convention containing three levels of qualification, specifically:

```
project_name.group_name.type_name
```

The first qualifier, project_name, is the unique project identifier associated with the hierarchy.

SCLM organizes project data sets into *groups*, the second identifier within the naming convention. Each group represents a different stage or state of the user data within the life cycle of a project. For example, assume a project has three groups named DEV1, TEST, and RELEASE. The DEV1 group represents data being modified. The TEST group represents data being tested. The RELEASE group represents data released for customer use. The groups of a project are organized into hierarchical order to form a tree-like hierarchy.

A group is made up of several data sets that can contain different *types* of data. Types, the third qualifier of the naming convention, are used to differentiate the kinds of data maintained in the groups of a project. For example, source code would be stored in one type and listings in another type. It is better not to mix different data types in SCLM. (Although SCLM allows you to do this, it is not recommended; data with different formats should be stored in different types.)

Thus a user working on an application for project PROJ1 might be assigned to the DEV1 group. The project can be using four different types of data. Therefore the user might have the following project partitioned data sets to work in:

```
PROJ1.DEV1.SOURCE       - all source modules
PROJ1.DEV1.OBJECT       - all compiler object files
PROJ1.DEV1.LISTING      - all compiler listings
PROJ1.DEV1.LOAD         - all executables (link edit output)
```

Note: SCLM can use data sets with names consisting of three levels of qualification as is the practice in many ISPF environments. It can also use data sets with two or more levels of qualification. This is an option that the project manager must enable for a project to use. If this option is used, SCLM developers would still use the `project_name.group_name.type_name` naming convention when performing SCLM functions. See Part Two of this book for more information on this option.

SCLM Hierarchies

The groups within a project are organized in a hierarchical order with each group being subordinate to the group above it. A sample hierarchy is shown in Figure 41.

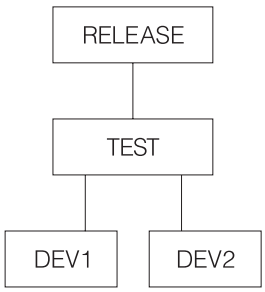


Figure 41. Sample Project Hierarchy

The topmost group is not subordinate to any group and is known as the top group, root group, or the root of the hierarchy. There is only one top group in each hierarchy. The bottom groups in a hierarchy are called development groups. The names for the development groups in Figure 41 are DEV1 and DEV2. All modifications and additions to user-created data must occur in the development groups of the hierarchy. Groups of equivalent rank within the hierarchy are considered to be within the same *layer* of the hierarchy. Most hierarchies have multiple layers.

Changes can be promoted to the next group, TEST, in the example hierarchy. Promote means to copy or move a member or a set of members from one group to the next group in the hierarchy. Each group can only promote members to the group to which it is subordinate. This link between groups is known as the *promote path*. For example in Figure 41 the three promote paths are DEV1 to TEST, DEV2 to TEST, and TEST to RELEASE. Any number of groups can promote into the same group.

Hierarchies are always searched from bottom to top along a path called the *hierarchical view*. The hierarchical view can begin at any group in the hierarchy and follows the promote paths to the topmost group in the hierarchy. Therefore in Figure 41 on page 142, two examples of hierarchical views are DEV1 to TEST to RELEASE and TEST to RELEASE. Thus, when referencing data in the hierarchy, members at lower groups take precedence over members at higher groups. All data existing in groups TEST and RELEASE is accessible from development libraries in groups DEV1 or DEV2. When a change is made to a member in the DEV1 group, this change is not available to the DEV2 group until the changed member has been promoted to the TEST group.

Therefore, within a hierarchy, the user data located at the lower layers of the hierarchy is in a more volatile state than the data at the upper layers. The upper layers of the hierarchy usually contain versions of products ready or nearly ready for release to customers, while the lower layers contain versions of products currently under development.

Key/Non-Key Groups

You can further identify groups in the project hierarchy as *key* groups and *non-key* groups. Key groups are defined as the groups within a hierarchy that contain all the software components of the application under development. A key group is a group into which you move data during a promotion. A project can have as many key groups as you want as long as any hierarchical view has no more than 123 groups. The actual limiting factor is the MVS limit of 123 extents for a concatenated partitioned data set.

SCLM allows a project to specify transition groups between key groups. These groups are known as non-key groups. A non-key group is a group into which you copy (rather than move) data during a promotion. When you promote data in a hierarchy, SCLM does not purge data from a key group until it reaches the next key group. Therefore, in a project with non-key groups, SCLM temporarily duplicates data in the non-key groups and the next lower key group. Figure 42 illustrates the relationship between a key and a non-key group within a project hierarchy.

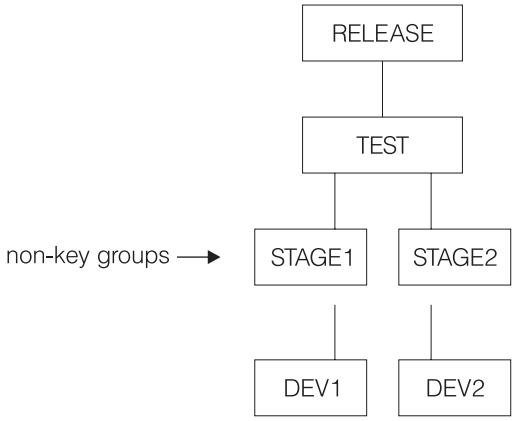


Figure 42. Key and Non-Key Groups Within the Project Hierarchy

As the figure shows, two non-key groups (the STAGE layer) appear between the development groups (the DEV layer) and the test and integration group (the TEST layer.) Developers use the STAGE groups as an interim place into which they promote their work before it moves to the next layer.

Using non-key groups enables you to display the critical elements of the hierarchical structure on ISPF panels. Because ISPF panels allow you to display only four key groups at one time, it is difficult to display the highest group in the hierarchy when you have a complex project that contains many layers.

Select key groups and non-key groups with the following set of guidelines:

- The lowest (development) groups must be key.
- Any group with more than one lower group promoting into it should be key.

Moving Data through the Hierarchy

Data moves within an SCLM hierarchy in two directions, up or down. When SCLM promotes members up the hierarchy from one group to the next group, the following rules apply:

- Copy members from key groups to non-key groups
- Move members from non-key groups to non-key groups
- Move members from key groups to key groups
- Move members from non-key groups to key groups and purge from the previous key group.
- Do not promote data from a primary non-key group.

In general, when SCLM accesses a hierarchy from a particular group, it concatenates only the necessary groups. If the lowest group in the hierarchy to be accessed is non-key, SCLM concatenates it with all the non-key groups above it, up to the next key group. From there, SCLM concatenates only the key groups. If the starting group in the hierarchy to be accessed is key, SCLM concatenates only it and the key groups above it.

The one exception to this concatenation involves non-key groups that have more than one group promoting into them. Non-key groups of this kind are as significant as key groups, and SCLM must also concatenate them in a hierarchy. Groups that must be concatenated when a hierarchy is to be accessed are known as *primary groups*. Thus, all key groups and all non-key groups with more than one group promoting into them are primary groups.

After members are promoted from the development groups to the higher groups in the hierarchy, users can bring members back to the development groups by performing a *draw down*. A draw down copies the member at the higher group to the specified development group. For a member to be drawn down it must be within the *hierarchical view* of the development group. Members can only be drawn down to development groups. SCLM performs an automatic *draw down* when the member is edited.

Chapter 9. Using SCLM Functions

With SCLM functions, you can view, create, update, delete, compile, link, promote, and report on data stored in the database of a project. In addition, you can generate reports with the build, promote, and utilities functions.

You can call SCLM functions in a variety of environments. In addition to the SCLM dialog interface, you can call a subset of SCLM functions independently with a command line processor or a program service interface. Refer to *ISPF Software Configuration and Library Manager (SCLM) Reference* for more information.

This chapter describes the panels you use to access the SCLM functions and the various options you can select from each panel. It also describes the panels that allow you to generate reports and provides several examples of the reports.

This chapter also compares SCLM to ISPF and notes the differences in the EDIT commands and the similarities of the utilities.

You can access all interactive SCLM functions through a set of panels under ISPF dialog management by selecting the SCLM option from the ISPF Primary Option Menu.

If SCLM does not appear on any of your menu panels or on your *Menu* pull-down, you can still access it by typing **TSO SCLM** on any ISPF command line, then pressing Enter. If SCLM is available to your terminal session, the SCLM Main Menu is displayed. If SCLM is not available on your system, a panel (ISRNOSLM) is displayed to inform you that SCLM is not available to your terminal session.

Notes:

1. A virtual region size of 4096K is recommended when you use the SCLM dialog. Increase the virtual region size if you encounter abends related to insufficient memory.
2. SCLM maintains allocations of data sets in the hierarchy between uses of SCLM functions. This enhances the performance of SCLM; however, if data sets in the hierarchy are created, deleted, cataloged or uncataloged while SCLM is active, you should exit SCLM and reselect the SCLM Main Menu.

Name Retrieval with the NRETRIEV command

The ISPF command table contains an entry named NRETRIEV. On enabled panels (such as edit and browse), NRETRIEV retrieves the library names from the current library referral list, or data set or workstation file names from the current data set referral list. The user is responsible for assigning the NRETRIEV command to a PF key.

When the cursor is **not** in the *Other Data Set Name* field, the *volume* field, or the *workstation file name* field, and the NRETRIEV key is pressed, the ISPF library fields are filled in from the current list. As long as the cursor is not placed in these fields, subsequent presses of the NRETRIEV key will retrieve the next library concatenation from the list.

When the cursor **is** in the *Other Data Set Name* field, the *volume* field, or the *workstation file name* field, and the NRETRIEV key is pressed, the other data set

name or workstation name field is filled in from the current data set list. ISPF attempts to determine if the name in the list is a workstation or data set name. As long as the cursor is placed in these fields, subsequent presses of the NRETRIEV key will retrieve the next data set or workstation name from the list.

Use the personal list settings panel to force the NRETRIEV command to verify the existence of a data set before retrieving it. If verification is active, then a check is made to see if a data set name exists before a retrieval attempt. If a volume name is not in the personal list entry, then the catalog is checked to see if the data set name is cataloged. If a volume name exists, an OBTAIN macro is used to check the volume for the data set. Verification does not check ISPF library names or workstation names, and does not check for the existence of PDS(E) members. In the data set list *dsname level* field, verification is inactive and workstation names are never retrieved.

NRETRIEV is enabled on the following options:

- View, including extended move, copy, create, and replace panels
- Edit, including extended move, copy, create, and replace panels
- Library Utility (Option 3.1)
- Data Set Utility (Option 3.2)
- Move/Copy Utility (Option 3.3)
- Data Set List (Option 3.4)
- Reset ISPF Statistics (Option 3.5)
- Hardcopy Utility (Option 3.6)
- Workstation Transfer (Option 3.7.2)
- SuperC (Options 3.12, old and new, and Option 3.14)
- SCLM Options:
  - View (Option 1)
  - Edit (Option 2)
  - Member list (Option 3.1)
  - Migration (Option 3.3)
  - Build (Option 4)
  - Promote (Option 5)

SCLM Considerations for NRETRIEV

The NRETRIEV command is enabled to work in several of the SCLM options. There are certain restrictions and considerations to keep in mind when you choose to use NRETRIEV in SCLM.

SCLM Restrictions

- The NRETRIEV key within SCLM does **not** use the standard reference list or personal lists. Instead, it uses a ***stack*** that is stored internally. The stack is not editable. The stack is saved from session to session as a single-line table called ISRSLIST.

  NOTE: In the SCLM View option, the ***other data set name*** field **does** use the standard reference list because the *other data set name* field has no particular meaning to SCLM.
- In SCLM, there is no validation of saved or retrieved names. That means that if you type in a library name and press enter, it is added to the list of saved names, even if SCLM does not process it. This contrasts with the standard

reference list processing, which does not add a data set or library name until the data set or library is successfully allocated.

- On name retrieval (when the NRETRIEV key is pressed) there is no validation of the existence of data sets or libraries.
- The regular NRETRIEV command is screen independent (it uses a separate list indicator for each screen in split screen mode). There is only 1 position locator for SCLM lists. This means that split screens with SCLM NRETRIEV will use the same pointer into the list. An NRETRIEV on screen 1 followed by an NRETRIEV on screen 2 will get list entries 1 and 2 respectively.

Stack Management for SCLM

A library name (or concatenation) is added to the *saved library* list by pressing enter on a panel that supports saving names. If the library or concatenation exists in the list already, it is moved to the top of the list. Where the ***project*** field, or the first ***group*** field is an output field (SCLM options 2, 3, 4, and 5), the output fields are not used in the comparison between what was typed on the panel and what is already in the list. This enables you to work in different but similar projects.

In other words, on the edit screen that has both the **project** and **group1** as output fields, the concatenation:

```
SCLM Library:
   Project...: PDFTDEV
   Group ....: DGN      ....STG    ....INT    ....SVT
   Type .....: ARCHDEF
   Member ...:
```

would match

```
SCLM Library:
   Project...: PDFTOS25
   Group ....: JSM      ....STG    ....INT    ....SVT
   Type .....: ARCHDEF
   Member ...:
```

Similarly, where groups 2, 3, and 4 are not used, those groups are not used when checking to see if the name already exists.

If a match is found, the existing entry in the list is moved to the top of the list.

SCLM Main Menu

Figure 43 on page 148 shows the seven SCLM primary functions from the SCLM Main Menu.

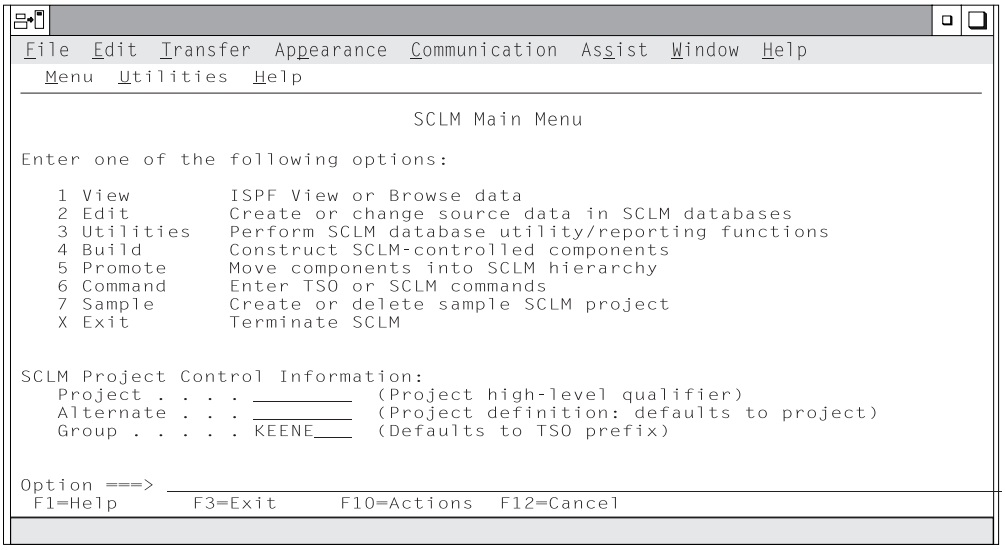


Figure 43. SCLM Main Menu Panel (FLMDMN)

SCLM Main Menu Options

When you select one of these options and press Enter, another panel appears that is determined by the option you selected.

View	See "View (Option 1)" on page 149.
Edit	See "Edit (Option 2)" on page 152.
Utilities	See "Utilities (Option 3)" on page 159.
Build	See "Build (Option 4)" on page 217.
Promote	See "Promote (Option 5)" on page 223.
Command	Enter and execute a TSO, CLIST, REXX exec, or SCLM command from within SCLM.
Sample	See "Sample Project Utility (Option 7)" on page 232
Exit	Exit from SCLM.

SCLM Main Menu Action Bar Choices:

Menu	See "Menu Action Bar Choice" on page xxix.
Utilities	See "Utilities Action Bar Choice" on page xxx.
Help	Help for general and specific topics.

SCLM Main Menu Panel Fields:

Project	A project's unique identifier. This field is required to access any SCLM function.
Alternate	The name of an alternate project definition to use. If this field is left blank, it defaults to the value specified in the **Project** field.

Group	This group defines the bottom of the hierarchical view used by the selected function, and can be any group in the hierarchy. This field defaults to your TSO PREFIX or to your user ID if no TSO PREFIX has been created. This field must be a development group if Edit (2) is chosen.

View (Option 1)

The SCLM View function uses the ISPF View service with an SCLM shell around it. The View function allows you to display data in a project hierarchy or data that is not controlled by SCLM. The SCLM View interface analyzes the hierarchy structure for the project you specify and automatically provides the appropriate concatenation sequence for the groups. It presents the four lowest key groups identified in the project definition, starting from the **Group** specified on the Main Menu.

SCLM View is functionally equivalent to ISPF View. (Refer to *ISPF User's Guide* for more information.) For example, you can specify a member name unless you want to see a member selection list. Additionally, you can modify the displayed library (or "group") concatenation sequence. You can also view a partitioned data set (PDS), a partitioned data set extended (PDSE), or a sequential data set. Figure 44 shows the panel SCLM displays when you select option 1, View, from the SCLM Main Menu.

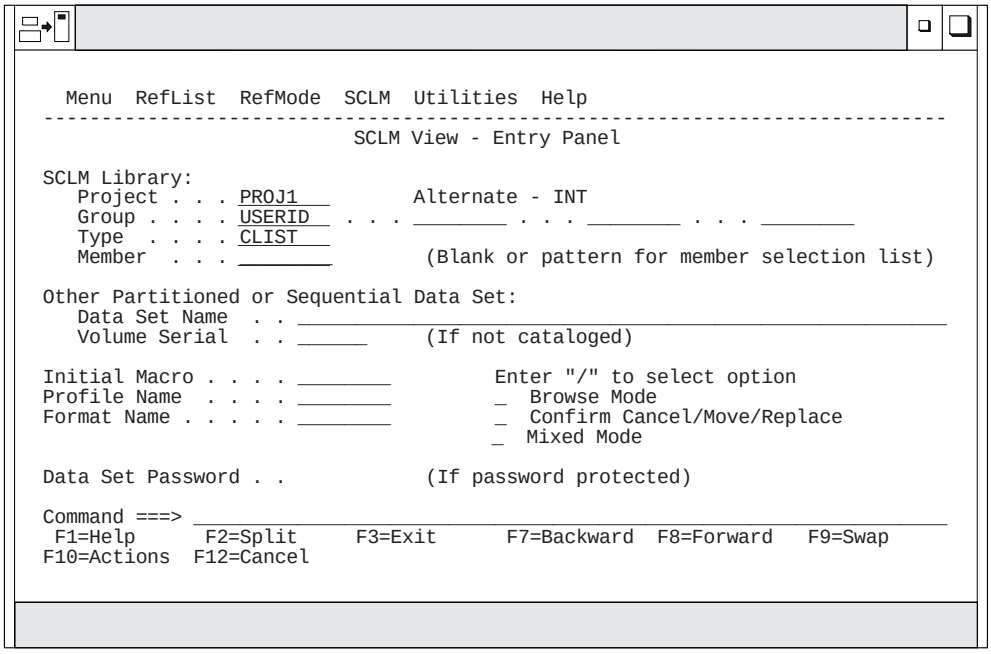


Figure 44. SCLM View - Entry Panel (FLMEB#P)

Note: The NRETREIV command key is enabled to work with this option. See "Name Retrieval with the NRETRIEV command" on page 145 for more information.

SCLM View - Entry Panel Action Bar Choices

The action bar displays the same choices as those discussed in "SCLM Main Menu Action Bar Choices:" on page 148. Additional choices are:

Reflist

The Reflist pull-down menu has the following choices:

Reference Data Set List	Displays a list of up to fifteen data set names that have been entered in the ″Other″ **Data Set Name** field and other fields in ISPF that take a data set name as input.
Reference Library List	Displays a list of the last eight ISPF libraries that you have referenced.
Personal Data Set List	Displays a list of up to thirty data set names that you have created and saved.
Personal Data Set List Open...	Displays the **Open** dialog for all Personal Data Set Lists.
Personal Library List	Displays a list of up to eight ISPF Library specifications that you maintain.
Personal Library List Open...	Displays the **Open** dialog for all Personal Library Lists.

Refmode

The Refmode pull-down menu has the following choices:

List Retrieve	Sets referral lists, personal data set lists, and personal library lists into a retrieve mode. When you select an entry from the list, the information is placed into the **Dsname Level** field, but the Enter key is *not* simulated. You can then set other options before pressing the Enter key. (If this is the current setting, this choice is unavailable.)
List Execute	Sets referral lists, personal data set lists, and personal library lists into a retrieve mode. When you select an entry from the list, the information is placed into the **Dsname Level** field, and the Enter key *is* simulated. (If this is the current setting, this choice is unavailable.)

SCLM

The SCLM pull-down menu has the following choices:

Library	Displays the SCLM library utility panel.
Sublib..	Displays the SCLM Sublibrary Management Utility panel.
Migration...	Displays the SCLM Sublibrary Management Utility panel.
DB Contents...	Displays the SCLM Database Contents panel.
Architecture...	Displays the SCLM Architecture Report panel.
Export...	Displays the SCLM Export Utility panel.
Import...	Displays the SCLM Import Utility panel.
Audit/Version...	Displays the SCLM Audit and Version Utility panel.
Delete Group...	Displays the SCLM Delete Group Utility panel.
Build...	Displays the SCLM Build panel.
Promote...	Displays the SCLM Promote panel.

SCLM View - Entry Panel Fields

Project	The project that you specified on the SCLM Main Menu. An **Alternate** field also appears if you specified an alternate project definition. If you change this field, all groups in the concatenation sequence are treated as data that SCLM does not control.
Group	SCLM uses the group specified in the **Group** field on the SCLM Main Menu to determine the four key or primary groups in the hierarchy that initially appear on the panel. You can enter both SCLM-controlled groups and non-SCLM-controlled groups in the concatenation sequence at the same time. If you specify a group that is defined in the project definition but not allocated, and you then request a member list, the library (LIB) members on the member list panel might not be what is expected. SCLM treats an unallocated group as if the group field were blank and ignores that group. When this situation exists, SCLM provides a panel that shows how the LIB numbers correspond to the existing groups.
Type	The identifier for the type of information in the group, such as SOURCE, ARCHDEF, or PANELS. If you change this field to a value that is **not** defined to the project definition, all the groups in the concatenation sequence are treated as data that SCLM does not control.
Member	The name of an SCLM or non-SCLM-controlled partitioned data set member. If you leave this field blank or type a pattern, a member list to appears.
Data Set Name	Any fully-qualified data set name, such as 'USERID.SYS1.MACLIB'. If you include your TSO user prefix (defaults to user ID), you must enclose the data set name in single quotation marks. If you omit the TSO user prefix, your TSO user prefix is added to the beginning of the data set name.
Volume Serial	A DASD volume identifier. ISPF does not allow a data set to reside on more than one volume. SCLM does not use the system catalog when you specify a volume serial.
Initial Macro	An Edit macro to be processed before you begin viewing your sequential data set or any member of a partitioned data set. This initial macro allows you to set up a particular environment for the View session you are beginning. If you leave the **Initial Macro** field blank and your Edit profile includes an initial macro specification, the initial macro from your Edit profile is processed. To suppress the processing of an initial macro in your Edit profile, enter NONE in the **Initial Macro** field.
Profile Name	A profile name to override the default Edit profile.
Format Name	The name of a format definition or blank if no format is used. A format definition can include **EBCDIC** fields, **DBCS** fields, and a **Mixed** field. If the specified format includes a **Mixed** field definition and you specify NO in the **Mixed Mode** field, SCLM ignores the operation mode.
Confirm Cancel/Move/Replace	Specifies that you want ISPF to display a confirmation panel whenever you issue a Cancel, Move, or Replace command.
Browse Mode	Specifies that you want to Browse the data set using the Browse function. This function is useful for large data sets and data sets that are formatted RECFM=U.

View (Option 1)

View on Workstation	Select this option to view the host data set member on the workstation using the workstation tool configured in the ISPF Tool Configurator. For more information, see the chapter on the ISPF Workstation Tool Integration Program in the *ISPF User's Guide*. Do not select this option if you want to view the host data set member on the host using SCLM VIEW.
Warn on First Data Change	Specifies that you want ISPF to warn you that changes cannot be saved in View. The warning is displayed when the first data change is attempted.
Mixed Mode	You can browse unformatted mixed data that contains both EBCDIC (1-byte) characters and Double Byte Character Set (DBCS or 2-byte) characters. To do this, you must select mixed mode by entering a slash (/) next to the **Mixed Mode field**. If your terminal does not support DBCS, SCLM View ignores the **Mixed Mode** field.
Data Set Password	The password for OS password-protected data sets. This is not your TSO user ID password.

Edit (Option 2)

The edit function is an interface to the ISPF editor. The SCLM editor ensures that editing occurs only in development groups. SCLM automatically locks the member when you begin the edit session.

The SCLM editor is the ISPF editor with an SCLM shell around it. Recursive editing is not supported within SCLM. If the member has changed when you end the edit session or if an explicit *SAVE* operation is performed, SCLM stores and parses the edited member and stores its accounting record. You can only edit members that reside in data sets under control of SCLM.

When you select the Edit option, the SCLM editor analyzes the hierarchy structure for the specified project and displays the sequence of the groups in your library concatenation. SCLM presents the four lowest key or primary groups for the project previously specified in the project definition. The SCLM lock feature, coupled with the ISPF "draw down" feature, ensures that the member you want to modify is the most current version of a component in the library concatenation.

SCLM copies or draws down the member or compilation unit to your development library in the development group from its first appearance in a higher key or primary group in the library concatenation. The member or compilation unit remains locked until you delete it or promote it to a higher group.

SCLM Edit also supports editing host data sets on the workstation. SCLM Edit will draw down the member if necessary, lock it, and copy it into working storage. The data set name is converted to a workstation file name and that name is appended to the workstation's current working directory. The host data set is transferred to the workstation, and the working file is then passed to the user's chosen edit program. When the user finishes the edit session, the working file is transferred back to the host and stored in the SCLM development group. Accounting information will then be saved for the member. The user will be prompted for a language if the member is new or does not have a language. For more information see the chapter on the ISPF Workstation Tool Integration Program in the *ISPF User's Guide*.

Figure 45 shows the panel SCLM displays when you select Option 2, Edit, from the SCLM Main Menu.

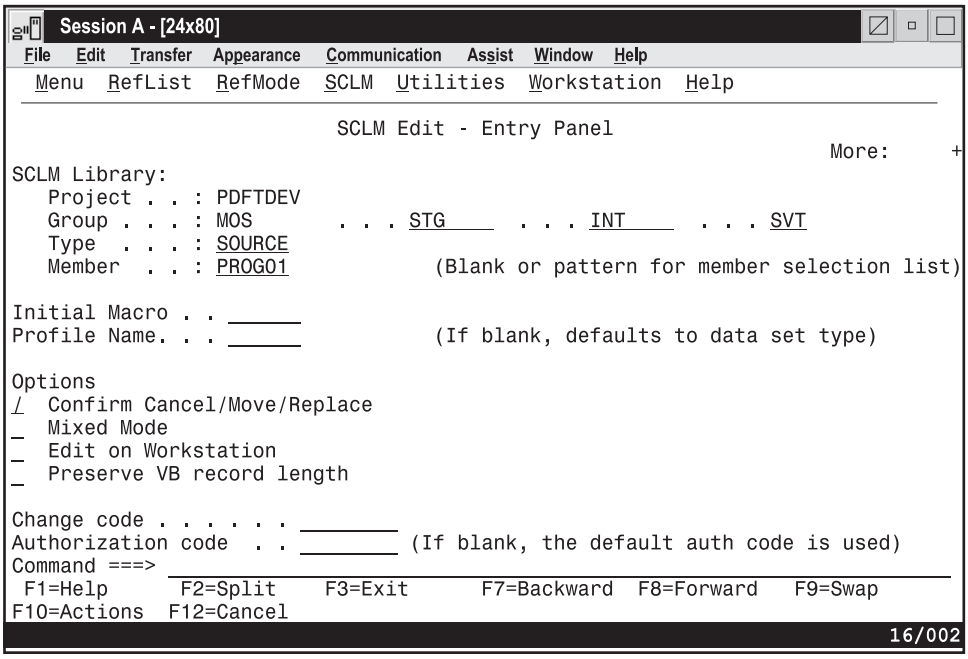


Figure 45. SCLM Edit - Entry Panel (FLMED#P)

Note: The NRETREIV command key is enabled to work with this option. See "Name Retrieval with the NRETRIEV command" on page 145 for more information.

SCLM Edit - Entry Panel Fields

Project	The project that you specified on the SCLM Main Menu. An **Alternate** field also appears if you specified an alternate project.

Group	The development group that you specified in the **Group** field on the SCLM Main Menu. This group is followed by the next key group in the hierarchy up to four groups. The SCLM editor ensures that editing occurs only in development groups by not allowing you to change the value of the first group field. SCLM guarantees that the group is a valid development library by verifying it against the specified project definition. (All other displayed groups are in unprotected fields and you can alter them.) If the order of the groups is specified so that it does not match the hierarchical view for the development group, SCLM does not allow the edit session and displays the message "Invalid library order". If F1 is pressed twice, SCLM displays a panel showing all groups that comprise the hierarchical view of the development group. If you specify a group that is defined in the project definition but not allocated, and then request a member list, the library (LIB) numbers on the member list panel might not be what is expected. SCLM treats an unallocated group as if the group field were blank and ignores that group. When this situation exists, SCLM provides a panel that shows how the LIB numbers correspond to the existing groups.
Type	The identifier for the type of information in the SCLM group, such as SOURCE, ARCHDEF, or PANELS.
Member	The name of an SCLM or non-SCLM controlled partitioned data set member. Leaving this field blank or typing a pattern as a member name causes SCLM to display a member list.
Initial Macro	An edit macro to be processed before you begin editing. This initial macro overrides any IMACRO value in your profile. If you leave the **Initial Macro** field blank and your edit profile includes an IMACRO specification, the initial macro from your edit profile is processed. If you want to suppress the processing of an initial macro in your edit profile, enter `NONE` in the **Initial Macro** field. Refer to *ISPF Edit and Edit Macros* for more information.
Profile Name	The name of an edit profile that you can use to override the default edit profile. Refer to *ISPF Edit and Edit Macros* for more information.
Confirm Cancel/Move/ Replace	Allows you to specify whether a confirmation panel will appear for these options.
Mixed Mode	You can edit unformatted mixed data that contains both EBCDIC (1-byte) characters and Double Byte Character Set (DBCS or 2-byte) characters. To do this, you must specify **Mixed Mode**. When you select **Mixed Mode**, the editor looks for shift-out and shift-in delimiters surrounding DBCS data. If you do not select it, the editor does not accept mixed data. If your terminal does not support DBCS, SCLM Edit ignores the operation mode.
Edit on Workstation	Select this option to edit the host data set member on the workstation using the workstation editor configured in the ISPF Tool Configurator. For more information see the chapter on the ISPF Workstation Tool Integration Program in the *ISPF User's Guide* Do not select this option if you want to edit the host data set member on the host using SCLM EDIT.

Preserve VB record length	When you select this field with a "/", it specifies that the editor store the original length of each record in variable length data sets and when a record is saved, the original record length is used as the minimum length for the record. The minimum length can be changed using the SAVE_LENGTH edit macro command. The editor always includes a blank at the end of a line if the length of the record is zero or eight.
Change Code	Optionally, you can specify a change code to indicate why you updated the member. Change codes cannot contain commas.
Authorization Code	Optionally, you can specify a current authorization code for the member. If you do not specify an authorization code, the default authorization code is used for the member. Authorization codes cannot contain commas.
Parser Volume	The specific volume ID in which SCLM stores output from the SCLM parser. This field is not required.

Comparison of SCLM and ISPF Editors

The SCLM edit function provides an interface to the ISPF editor. For example, you can specify a profile name and an initial macro before editing a member. With the SCLM editor, you can lock or parse a member, create or update an accounting record, and specify change or authorization codes. Recursive editing is only allowed within the data set concatenation currently being edited. Therefore, the member name to edit must be supplied as part of the edit command (see "EDIT Command" on page 156).

The parser supplied with SCLM does not recognize ISPF packed data. If the ISPF pack mode is on, the parser supplied with SCLM returns statistical values reflecting packed data. You must unpack the data before it is parsed by SCLM to obtain correct statistical values.

When editing parts controlled by SCLM, it is important to use the SCLM editor. The ISPF editor has a configuration table that supports three levels of awareness of SCLM-controlled parts if trying to edit SCLM-controlled parts with the ISPF editor (outside of SCLM):

No awareness — ISPF edit allows SCLM members to be edited, with no warning or message.

Warning Mode — ISPF edit displays an SCLM WARNING message when editing an SCLM-controlled member. However, the ISPF edit will continue.

Fail Mode — ISPF edit does not allow the edit to start on an SCLM-controlled member.

If the ISPF editor is operating in Fail Mode, edit recovery operates in Warning Mode for purposes of the recovery; you will be able to recover the member, and the SCLM WARNING message appears.

ISPF uses two checks to determine if a member is SCLM-controlled:

- The SCLM flag for the member is on (this is set by SCLM SAVE)
- A `project.PROJDEFS.LOAD` data set exists, where the high-level qualifier of the data set being edited is equal to `project`.

When the configuration table has Fail Mode set, both conditions must be true for the ISPF editor to operate in Fail Mode. If only the second condition is true, the ISPF editor operates in Warning Mode.

SCLM Command Macros

The following sections describe the command macros available for use with the SCLM editor.

EDIT Command

The SCLM EDIT command allows a user to recursively edit a member within the same hierarchy concatenation of a SCLM supported type. That is, as long as the member exists within the groups and type specified in the **Group** and **Type** fields on the SCLM Edit - Entry panel, recursive editing is allowed.

Command Format:

```
Edit Member-name
```

Save Command

The SCLM SAVE command is similar to the ISPF Save command except that the member is automatically parsed and the accounting record of the member is created or updated.

The first time you save a member that has not been created using the SCLM editor (or migrated into SCLM), SCLM displays the SCLM Edit Profile panel (see Figure 46 on page 158) for you to specify a change code and the language of the member. The profile appears if SCLM has not been informed of the language of the member. The member is saved regardless of the parser return code on the first save.

Command Format:

```
SAVE
```

SCREATE Command

The SCLM SCREATE command is similar to the ISPF Edit CREATE command except that the SCLM editor automatically creates an accounting record for the created member, locks it out, and parses it.

If you do not enter a change code on the SCLM Edit - Entry panel (when one is required), SCLM displays the SCLM Edit Profile panel shown in Figure 46 on page 158. Also, if the language of the member you want to create differs from the language of the member you are editing, enter the SPROF command on the Edit - Entry panel. The SCLM Edit Profile panel appears so that you can specify another language. Otherwise, the newly-created member has the same member attributes as the current member.

Note: If the member to be created already exists in your group, SCLM returns a message indicating that the member already exists. Thus you can avoid inadvertently overwriting members.

The SCLM SCREATE command does not offer an extended panel for creating a member outside the hierarchy.

Command Format:

```
SCREATE   member-name    [label1 | label2]

SCRE
```

The label parameters indicate the lines from which the new member is created. For example, assume that member OLD has been previously defined to SCLM. The COBOL programming language is associated with member OLD. If you are editing member OLD, place copy block ('cc') commands in the **Line Command** field (usually represented by a six-digit number on the far left side of your edit screen) of lines two and five of member OLD, and then issue the command

```
SCREATE NEW
```

from the command line. Member NEW will be added to the data set containing member OLD. Furthermore, member NEW will contain lines two through five of member OLD and will also inherit member OLD's association with COBOL. In this case, the block copy commands are the first and second labels passed with the SCREATE command.

SMOVE Command

The SCLM SMOVE command is similar to the ISPF MOVE command except that the SCLM editor deletes the accounting and build map information of the member being moved if it exists in the development group from which the SMOVE was issued.

The SCLM SMOVE command does not offer an extended panel for moving a member from outside the hierarchy.

Note: Once a member is successfully moved, the source member of the move is deleted. If you CANCEL out of the edit session where the SMOVE command was initiated, the data is lost.

Command Format:

```
SMOVE     member-name  [AFTER label]
                       [BEFORE label]
```

The AFTER label parameter indicates the line after which to place the member that is being moved. To create an AFTER label, enter an "A" or "a" in the **Line Command** field (usually represented by a column of six-digit numbers on the far left side of your display) for the line you want.

The BEFORE label parameter indicates the line before which to place the member that is being moved. To create a BEFORE label, enter a "B" or "b." in the **Line Command** field for the line you want.

SPROF Command

The SPROF command allows you to specify parameters that SCLM requires to track a member through the hierarchy. SCLM displays the SCLM Edit Profile panel, shown in Figure 46 on page 158, to specify a language for a new member. This panel also displays when you end the edit session if you did not enter a change code on the SCLM Edit - Entry panel when it is required, or if the language of the member has not yet been specified.

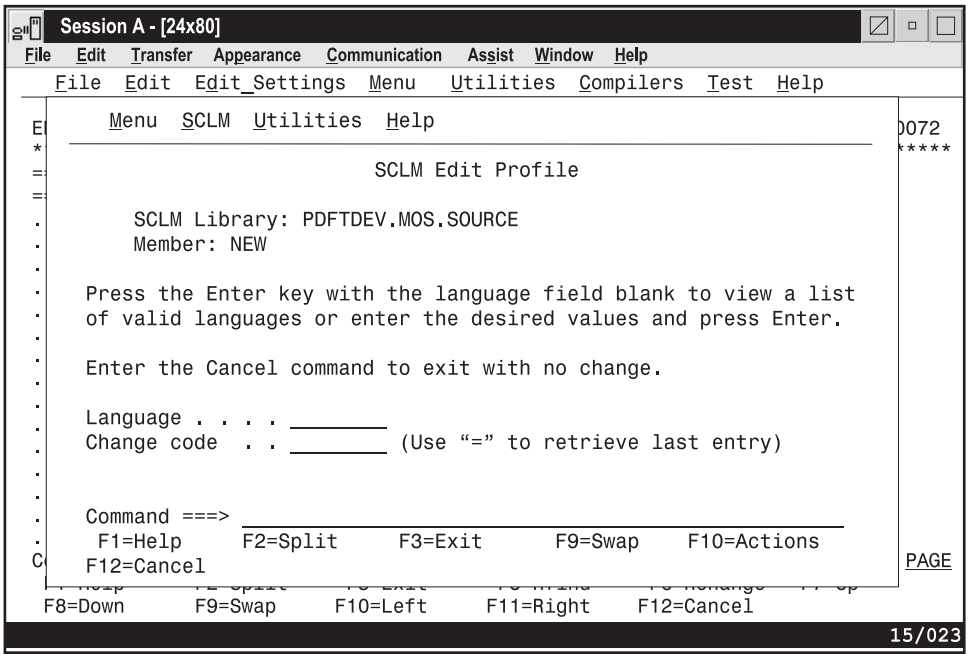


Figure 46. SCLM Edit Profile (FLMEINFO)

SCLM Edit Profile Panel Fields

Language	The language name to be used to process the member. This field is required and must be the same as the LANG keyword specified on the FLMLANGL macro.
Change code	Specify a change code to indicate why you updated the member. This field is optional unless a change code verification routine is defined for the hierarchy. Change codes cannot contain commas.

You can change the information on this panel at any time during the edit session by invoking SPROF. If you alter the **Language** field or modify the member, or both, SCLM parses and creates or updates the accounting record of the member when the member is saved. If you leave the language field blank or enter an invalid language, SCLM displays a selectable list of valid languages defined to the project.

SCLM processes the member and saves it in your development group if you alter the language or change code and if the member does not exist in your development library. If you alter the language or change code but do not modify the member and it exists in the development group, SCLM regenerates only the accounting information.

Enter END from the SCLM Edit Profile panel to end SCLM edit profile specifications and return to the SCLM edit session. Enter CANCEL to cancel any changes you have made on the panel, end SCLM edit profile specifications, and return to the SCLM edit session.

SREPLACE Command

The SCLM SREPLACE command is similar to the ISPF Edit REPLACE command except that the SCLM editor automatically parses, locks out, and creates an accounting record for the replaced member. Use this command, not SCREATE, when the member exists in the group.

If you do not enter a change code on the SCLM Edit Entry panel (when it is required), SCLM displays the SCLM Edit Profile panel shown in Figure 46 on page 158. Also, the replaced member has the same member attributes as the current member.

If you use SREPLACE and specify a member that does not exist, SCLM calls SCREATE by default so that you can create the member.

The SCLM SREPLACE command does not offer an extended panel for replacing a member outside the hierarchy.

The label parameters indicate the lines from which the current member is replaced by the replaced member. The label parameters are optional.

Command Format:

```
SREPLACE  member-name  [label1 | label2]

SREPL
```

To see an example of using commands with labels, see "SCREATE Command" on page 156.

Overriding SCLM Command Macros

Because the SCLM editor uses ISPF edit macros to perform its functions, you should not override SCLM command macro definitions, especially the END, SAVE, CANCEL, and RETURN macros. If you need a user-defined end macro, define an alternate command name such as QUIT. At the end of this alternate end macro, you must enter the END, RETURN, SAVE, or CANCEL command to start the SCLM end routines.

If you override an SCLM macro by using DEFINE, the macro is not redefined until you begin a new edit session.

You can also override SCLM edit macros by entering the ISPF BUILTIN command (for example, BUILTIN SAVE).

Utilities (Option 3)

Figure 47 on page 160 shows the panel SCLM displays when you select option 3, Utilities, from the SCLM Main Menu.

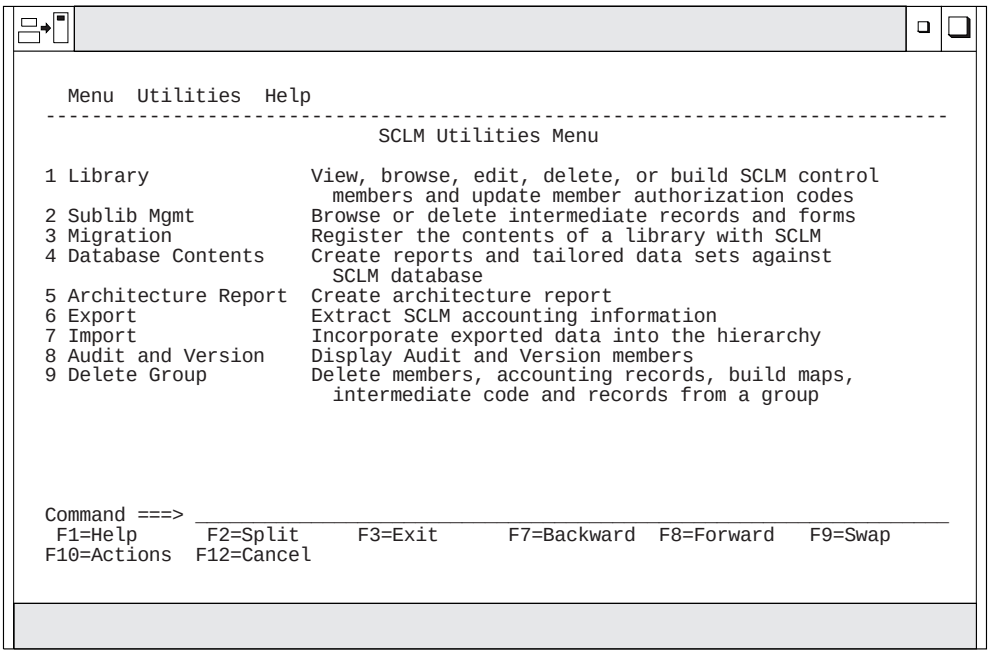


Figure 47. SCLM Utilities (FLMUDU#P)

When you select one of these options and press Enter, another panel appears, determined by the option you selected. Figure 47 shows the available options:

Library	See “Library Utility”.
Migration	See “Migration Utility” on page 176.
Database Contents	See “Database Contents Utility” on page 178.
Architecture Report	See “Architecture Report Utility” on page 189.
Export	See “Export Utility” on page 196.
Import	See “Import Utility” on page 200.
Audit and Version	See “Audit and Version Utility” on page 205.
Delete Group	See “Delete Group Utility” on page 213.

Library Utility

The library utility allows you to browse accounting records, members, and build map records. In addition, you can use this utility to delete members and their accounting and build map data, edit and build members, and update authorization codes.

The library utility is completely interactive and parallels the ISPF library utility.

Figure 48 on page 161 shows the SCLM panel that appears when you select Option 1, Library, from the SCLM Utilities panel.

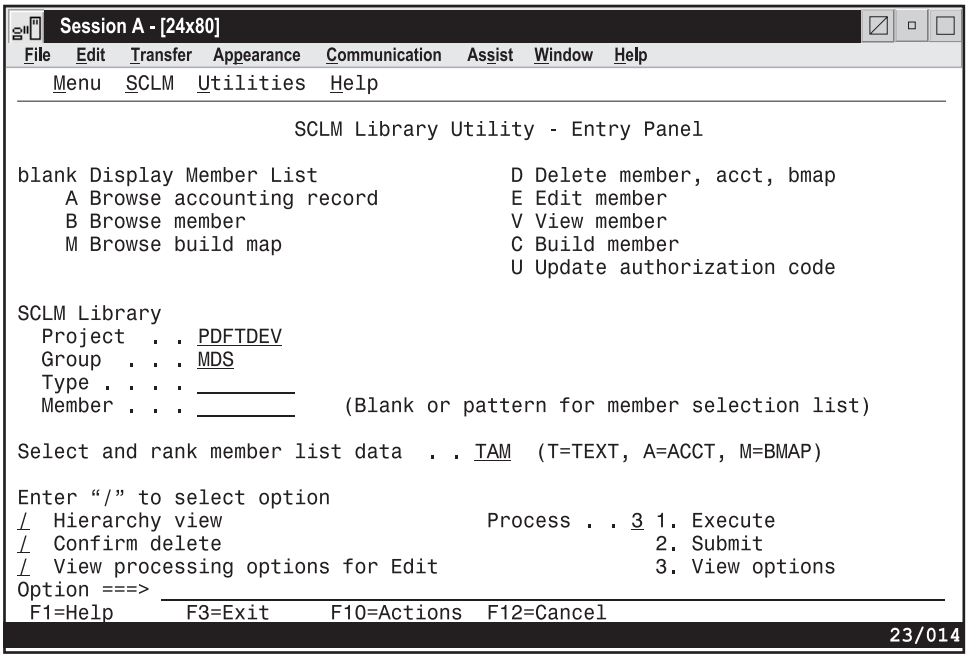


Figure 48. SCLM Library Utility (FLMUS#P)

The fields on the SCLM Library Utility panel are:

Project	The project that you specified on the SCLM Main Menu. An **Alternate** field also appears if you specified an alternate project definition. You cannot change the **Project** or the **Alternate** fields on this panel.
Group	The group that you specified in the **Group** field on the SCLM Main Menu. The group field can be modified to specify other groups defined to the project.
Type	The identifier for the type of information in the ISPF library.
Member	The name of an SCLM library member. You can display a member list by leaving the Command field blank and the **Member** field blank or by leaving the Command field blank and entering a pattern as the member name. See "Specifying Selection Criteria" on page 180 for details. Valid pattern characters are the asterisk (*) and the logical NOT symbol (¬).
Select and rank member list data	A one, two, or three character string that indicates the kind of information that appears on the member list panel. You can specify strings composed of the following characters: T, to display text data; A, to display accounting data; and M, to display build map data. Each character can only be used once. The order of the characters determines the order of the data on the member list. This option only limits the type of data that appears each member on the list. All types of data that exist for a member at a particular level are subject to processing by library utility commands.
Hierarchy view	Selects as input the library entered on the panel, as well as all the libraries in its hierarchy view. The hierarchy is searched from the bottom up for the first occurence of the specified member. If you do not select **Hierarchy view**, only the library entered on the panel is used as input. This option is valid with all Library Utility - Entry panel or member list commands except delete, which defaults to a NO value.

Confirm delete	Allows you to specify whether you want a confirmation panel to appear when attempting to delete objects (text, accounting information, or build map information) with the SCLM library utility. If you select this field, the Confirm Delete panel appears every time you request a delete. If you do not select this field, the Confirm Delete panel does not appear for deletions and data is deleted without any additional user interaction.
View processing options for Edit	Allows you to indicate whether you want to verify or update edit processing options or allow them to default to the values that last appeared on the Edit Data Entry panel. When you select this option, the SCLM Edit Data Entry panel displays so that you can verify or update edit processing options. If you do not select it, Edit options default to those values that last appeared on the Edit Data Entry panel. The panel does not appear.
Process	The **Process** field allows you to specify the processing mode for Build command. The value of the **Process** field is unique to the library utility. You will not be carried to or from the **Process** field on any other SCLM panel. **Execute** Invokes SCLM Build in the foreground. Build options default to those values that last appeared on the Build Data Entry panel. The panel does not appear. **Submit** Invokes SCLM build in the background. Build options default to those values that last appeared on the Build Data Entry panel. The panel does not appear. **View options** Displays the SCLM Build Data Entry panel so that you may verify or update build processing options prior to execution.

Note: The value for **Confirm delete** is reset each time the library utility is entered. The values for **Select and rank member list data**, **Process**, **Hierarchy view**, and **View processing options for Edit**, are kept from session to session until you change them.

Library Utility Commands

Type your selection in the **Command** field.

`A`, `B`, or `M`	SCLM displays the specified member or record if it is present. While in Browse, all Browse commands are supported. Note that although a hierarchy view may be specified, the Library Utility only allocates the data set containing the existing version of the requested member. The Browse command executed from within View can only operate on members within the allocated data set.
`V`	SCLM displays the specified member if it is present.
`D`	SCLM deletes all portions of the member such as text, accounting, and build map records. Delete is only allowed at the group specified on the Library Utility panel. If you delete a member from a key group that also exists in a non-key group in a higher layer of the hierarchy, you must delete the member from the non-key group manually.

E	The SCLM Editor is invoked for the member specified in the Member field. A development group must be specified in the Group field. Once in the SCLM Editor, all Edit commands are supported. The library utility allocates the first four key groups for a project. If the member exists at a higher group, the group containing the member will be allocated, replacing the original fourth allocated group. The COPY, MOVE, and EDIT commands can only operate on members within the allocated data sets. The use of COPY or MOVE from within an Edit session invoked from the utility is not recommended.
C	SCLM Build is performed on the specified member.
U	SCLM displays an input panel and updates the authorization code according to your input. Update is only allowed at the group specified on the Library Utility panel. (To delete or update any data, you must have at least UPDATE authority to the specified data set.) Any value entered in the **New authorization code** field on the input panel remains there until it is changed by the user or the library utility is exited and re-entered. There is a brief period during which changes made to a member's authorization code by another session or user will not be recognized. If you receive an unexpected error message while updating a member's authorization code, use the browse accounting record command to check the member's current authorization code. If the authorization code needs to be updated, try the update authorization code command again.

To browse, edit, delete, build, or update several members, use the member selection list.

Member Selection List

You can delete, browse, or update members by making selections from a member selection list. To display a member selection list, do the following:

1. Leave the **Command** field blank.
2. Type the group and type information in the appropriate fields. The **Project** field contains the project you specified on the SCLM Main Menu. You cannot change this field here.
3. Leave the **Member** field blank or enter a pattern.
4. Choose the data to appear and the order to display it on the member list panel by entering a string in the **Select and rank member list data** field.
5. Indicate whether you want a hierarchy view by entering a slash (/) in the **Hierarchy view** field.
6. Press Enter.

Note: Any changes that are made outside of the member selection list are not reflected in the list until it is exited and re-entered. For example, if you rename or delete a member, or add a member using ISPF, you will not see these changes in the list until you exit the list and display it again.

Note: The NRETREIV command key is enabled to work with this option. See "Name Retrieval with the NRETRIEV command" on page 145 for more information.

Figure 49 on page 164 shows the panel SCLM displays when you complete the instructions for displaying a member list. This display contains text, accounting, and build map data, indicating that the string "TAM" was entered for the **Select and rank member list data** field. In this example, the A line command is invoked

for member FLM01MD5. Use the scroll commands or the LOCATE command to scroll the list.

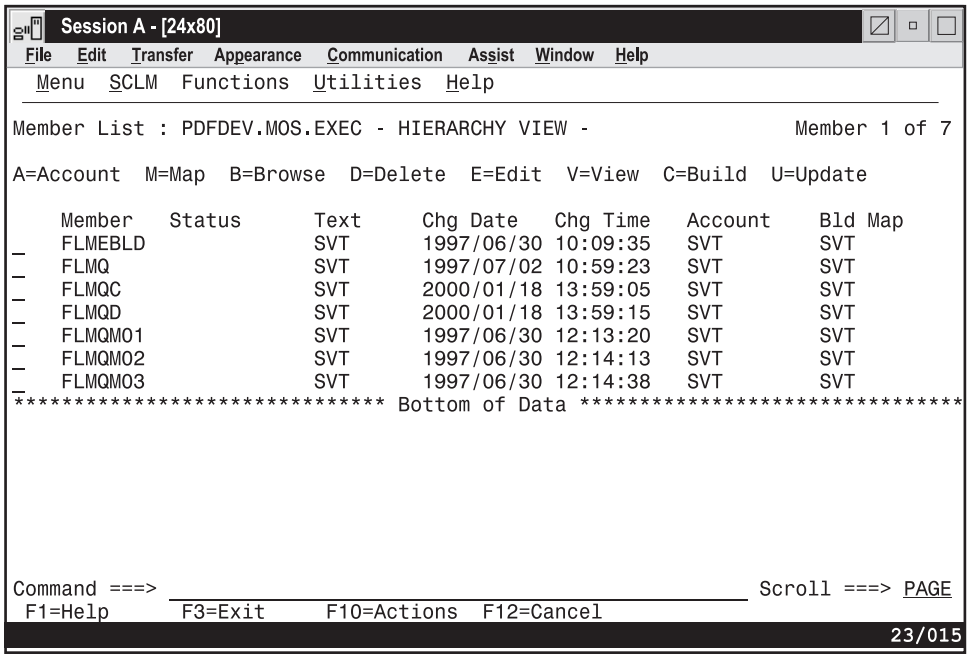


Figure 49. Member Selection List (FLMUSL#P)

Another way to view a member list is shown in Figure 50. In this example, the string "AT" was specified for the **Select and rank member list data** field, causing accounting and text data, in that order, to appear on the member list panel. Also note that a hierarchy view was requested for this member list.

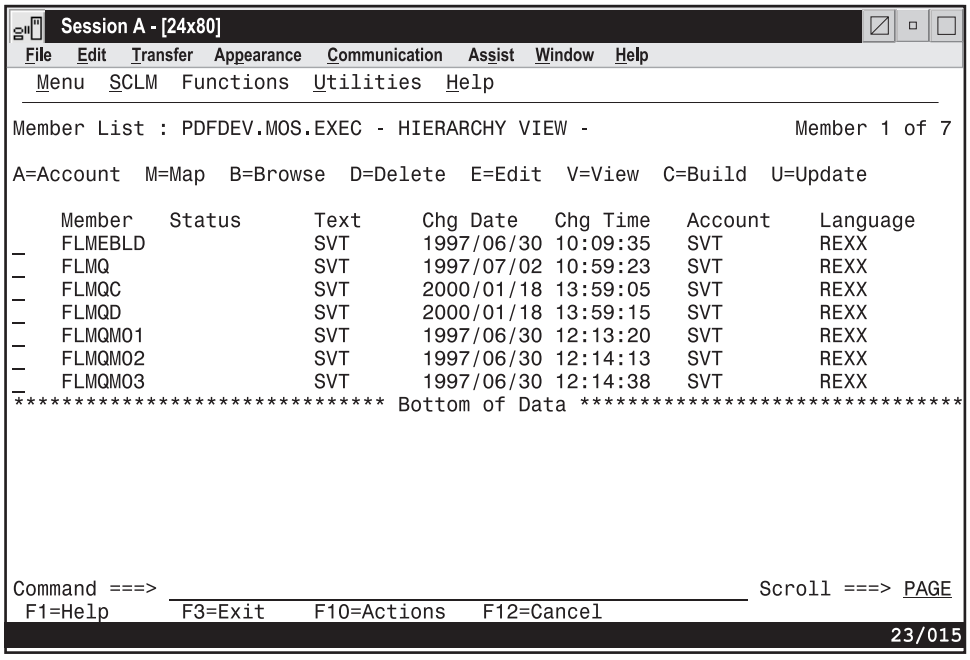


Figure 50. Member Selection List with Hierarchy View (FLMUSL#P)

The fields that appear on the SCLM Member Selection List panel are:

Member	The names of the members fitting the criteria you specified on the SCLM Library Utility - Entry panel.

Status	SCLM displays the status of the member according to the line command you select. The status field indicates the action that was taken for the selected member. For example, a status of *EDITED will appear next to any member for which the 'E' command is selected, even if the member is not saved. The status for delete indicates the group at which the delete occurred. The status displayed for each command is shown in the following example:
	`A  Display an accounting record  *BRACCT`
	`B  Browse a member  *BRTEXT`
	`C  Build a Member  *BUILT`
	`D  Delete a member  *D-GROUP1`
	`E  Edit a member  *EDITED`
	`M  Display a build map record  *BRBMAP`
	`U  Update an authorization code  *UPDATED`
	`V  View a member  *VIEWED`
	When an error occurs or the member name is changed on the edit or Build Data Entry panel, the status for the member will be blank.

Account	A group name in this field indicates that the accounting information for the associated member exists.
Language	The language of the member appears in this column when accounting data is requested and when space permits.
Text	A group name in this field indicates that the member exists.
Chg Date	The value of this field depends on the type of data requested for display. When text data is requested, this field contains the last change date for the member from the PDS directory. If accounting data is requested but text is not, this field contains the change date from the accounting record. If only build map data is requested, the change date from the build map appears.
Chg Time	The value of this field depends on the type of data requested for display. When text data is requested, this field contains the last change time for the member from the PDS directory. If accounting data is requested but text is not, this field contains the change time from the accounting record. If only build map data is requested, the change time from the build map appears.
Bld Map	A group name in this field indicates that the build map record for the associated member exists.
Authcode	The current authorization code for the member appears in this column when accounting data is requested and when space permits.

Accounting Record

If you enter the A line command to display an accounting record, SCLM displays a panel showing the information recorded for the member as shown in Figure 51 on page 166.

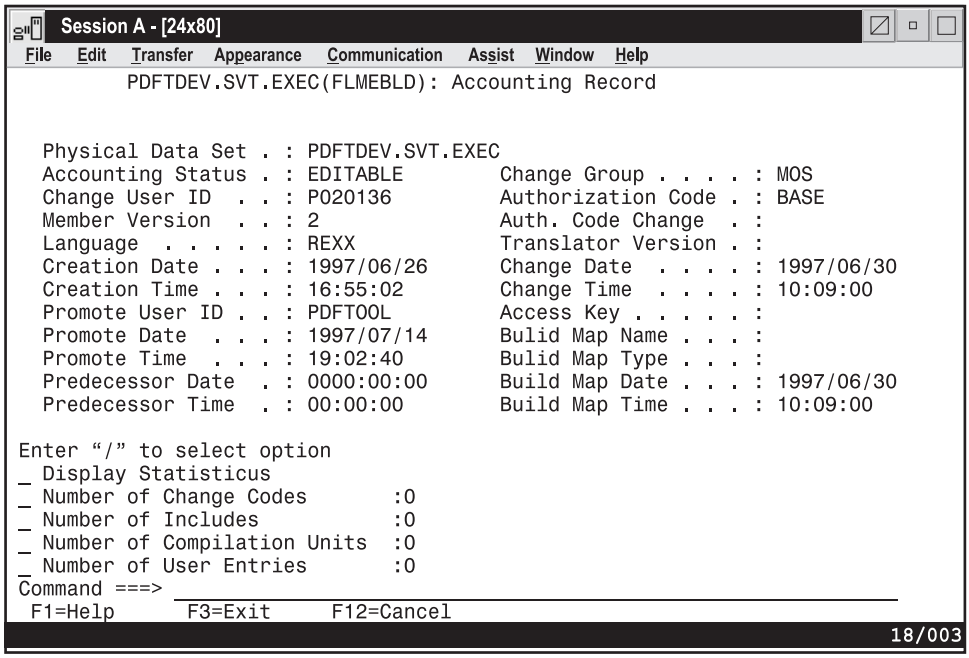


Figure 51. Accounting Record (FLMUSA#P)

The display fields on the Accounting Record panel cannot be modified.

Use a slash (/) to select an option and press Enter to display additional panels. You can browse the statistics or lists of change codes, includes, compilation units, or user entries referenced by a member. You can also scroll the lists.

Physical Data Set	The physical data set in which the SCLM-controlled member actually resides. SCLM allows you to define project data sets that don't have conventional SCLM data set names by providing SCLM aliases for them. When this is the case, the name appearing on the panel title is the SCLM alias for the actual data set in the **Physical Data Set** field.	
Accounting Status	The status of the member.	
	EDITABLE	Members that you can edit
	NON-EDIT	Members that SCLM creates as a result of build processing
	LOCKOUT	Members that are locked at the development group in which they exist but have not been parsed. You can use the SCLM Editor or Migration Utility to change the status of these members to EDITABLE before attempting to build or promote them.
	INITIAL	Members for which a lock has been requested. This status generally appears while a member is being edited. When the edit is complete, the status changes to EDITABLE.
Change User ID	The user ID of the person who made the last update to the member.	

Member Version	The number of times that an EDITABLE member was drawn down. The member version is also updated whenever the language of the member is changed. For a NON-EDIT member, such as OBJ, it is the number of times that the member was generated by SCLM. New members use a version of 1.
Language	The language of the member.
Creation Date	The date the member was first registered with SCLM.
Creation Time	The time the member was first registered with SCLM.
Promote User ID	The user ID of the person who last promoted the member.
Promote Date	The date the member was last promoted.
Promote Time	The time the member was last promoted.
Predecessor Date	The change date of the member that this member overlays when it is promoted up the hierarchy.
Predecessor Time	The change time of the member that this member overlays when it is promoted up the hierarchy.
Change Group	The name of the group in which the member was last updated.
Authorization Code	The current authorization code for the member.
Auth. Code Change	A non-blank value indicates that SCLM is attempting to update the Authorization Code for this member. If the update completes successfully, the value of this field becomes the new authorization code of the member.
Translator Version	The version of the translator used during build processing.
Change Date	The last date a developer modified the member.
Change Time	The last time a developer modified the member.
Access Key	An identifier used to restrict access to a member.
Build Map Name	For NON-EDIT members, this field specifies the name of the build map that was created when the NON-EDIT member was created. For EDITABLE members, this field is blank.
Build Map Type	For NON-EDIT members, this field specifies the type of the build map that was created when the NON-EDIT member was created. For EDITABLE members, this field is blank.
Build Map Date	The date used by SCLM to determine if the member has changed since the last build. For EDITABLE members, this field is usually the same as the **Change Date** field. When the **Change Date** field is updated, the **Build Map Date** field is updated. For NON-EDIT members, this field is the date of the last build of the member.
Build Map Time	The time used by SCLM to determine if the member has changed since the last build. For EDITABLE members, this field is usually the same as the **Change Time** field. When the **Change Time** field is updated, the **Build Map Time** field is updated. For NON-EDIT members, this field is the time of the last build of the member.
Display Statistics	SCLM displays the Accounting Record Statistics panel, shown in Figure 52 on page 168.
Number of Change Codes	The number of change codes entered against the member. See Figure 53 on page 169
Number of Includes	The number of include references in the source member. See Figure 55 on page 171.
Number of User Entries	The number of user data entry records associated with the member.

Statistics

SCLM displays statistical information, as shown in Figure 52, when you enter a "/" in the **Display Statistics** field on the Accounting Record panel. These statistics are parser-dependent.

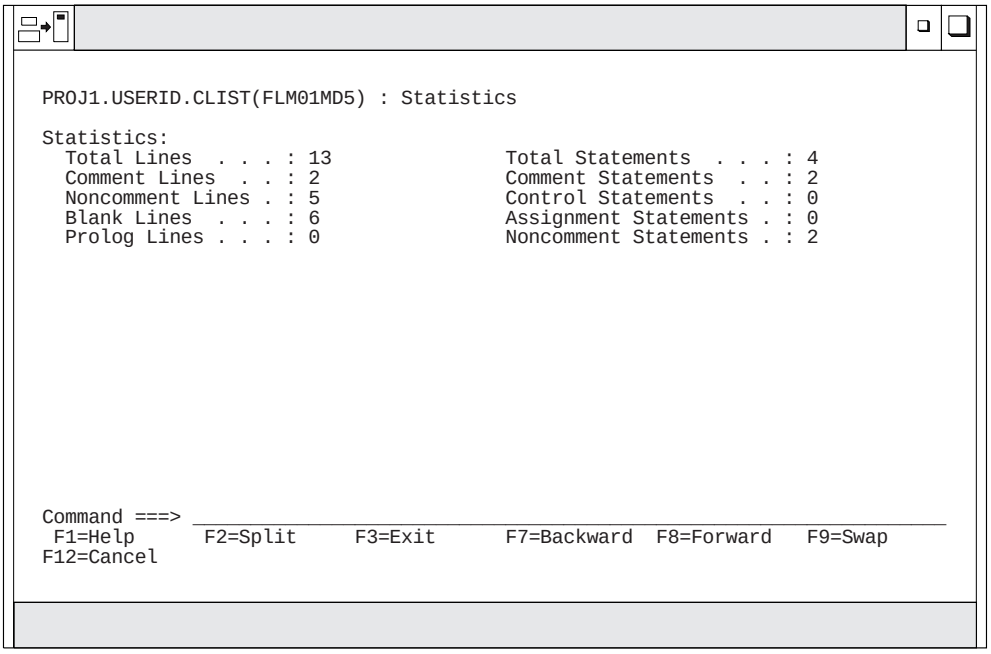


Figure 52. Accounting Record Statistics (FLMUSS#P)

The fields on the Accounting Record Statistics panel are:

Total Lines	The total number of lines in the member, which is equal to the sum of comment lines, noncomment lines, and blank lines.
Comment Lines	The number of comment lines. A comment line is any line that has comment information only. If a line has both a statement and a comment, SCLM considers it a noncomment line.
Noncomment Lines	The number of source lines. A noncomment line is a source line that contains at least part of a noncomment statement. If a line has both a statement and a comment, SCLM considers it a noncomment line.
Blank Lines	The number of blank lines in the member. A blank line is language-independent; no nonblank characters can be on it. These statistics are parser-dependent.
Prolog Lines	The number of prolog lines in the member.
Total Statements	The sum of the comment statements and the noncomment statements in the member.
Comment Statements	The number of comment statements. A comment statement is denoted by a set of beginning and ending comment delimiters for the particular language being parsed. If an ending delimiter is not defined for a language, the end of the line is used. A comment statement can span several lines, or several comment statements can exist on a single line.
Control Statements	The number of logical control statements.
Assignment Statements	The number of assignment statements.

Noncomment Statements	The number of complete statements that SCLM can process. Noncomment statements are language-dependent, follow language syntax rules, and are separated by the language delimiter. A noncomment statement can span several lines, or several noncomment statements can exist on a single line.

Note: The parser that is invoked for the member determines the field values. The definitions apply for ISPF-supplied parsers.

Change Code List: Figure 53 and Figure 54 on page 170 are examples of the information SCLM displays when you enter a "/" in the **Number of Change Codes** field on the Accounting Record panel. If your are allowed to delete the records you specify, Figure 53 is displayed. If not, you will see Figure 54 on page 170.

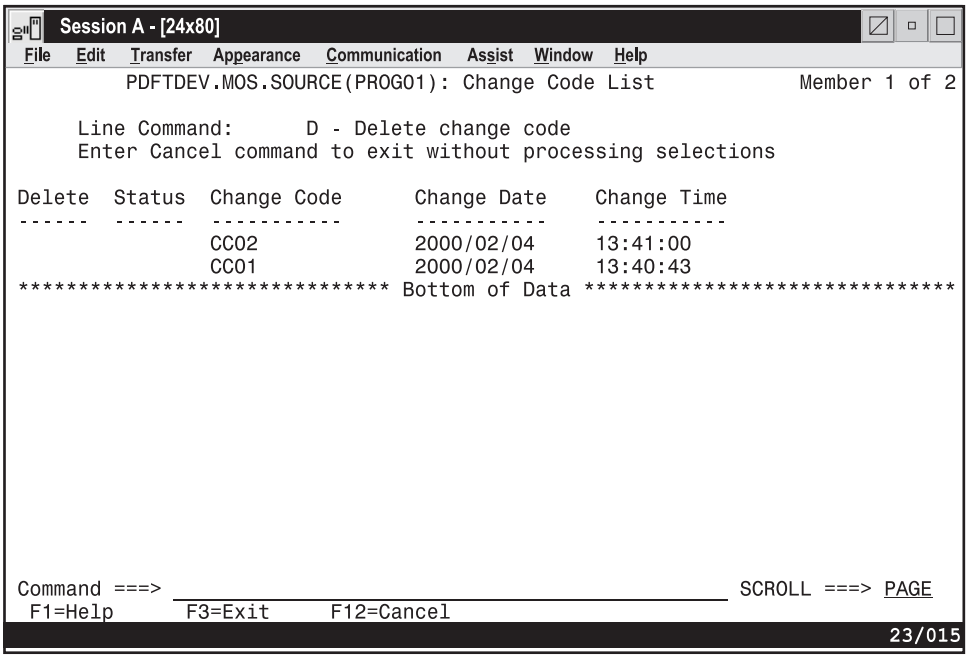


Figure 53. Change Code List - Deletable Records(FLMUSC#P)

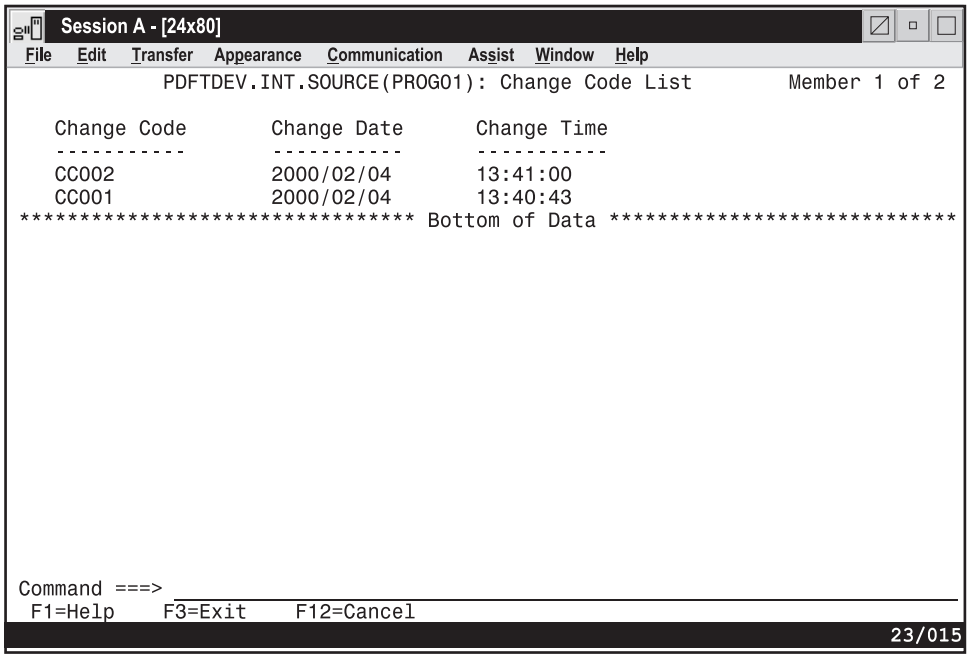


Figure 54. Change Code List - Nondeletable Records (FLMUSC2P)

The fields on the Change Code List panel are:

Delete	You specify that you want to delete the change code when you enter `D` in this field. SCLM selects the change code for deletion.
Status	SCLM displays `*SELECT` to indicate the change code you selected. Enter the END command to confirm the delete request.
Change Code	A value assigned to indicate why a member was updated.
Change Date	The last date a developer modified the member for the associated change code. The Change Date on the top of the list is the most recent.
Change Time	The last time a developer modified the member; it is associated with the Change Date.

Include List: Figure 55 on page 171 is an example of the information SCLM displays when you enter a "/" in the **Number of Includes** field on the Accounting Record panel.

Figure 55. Include List (FLMUSI#P)

The fields on the Include List panel are:

Include	The name of an include reference in the source member. An include reference is a generic term for code that SCLM inserts when it compiles the source member. The syntax of an include statement in a program is language-dependent and is defined by language syntax rules.
Include set	The include-set name is used to associate an include with the types in the hierarchy where that include can be found. The include-set name is returned by the parser. A blank name indicates that the include is associated with the default include set.

User Data Entries: Figure 56 on page 172 is an example of the information SCLM displays when you enter a / in the **Number of User Entries** field on the Accounting Record panel.

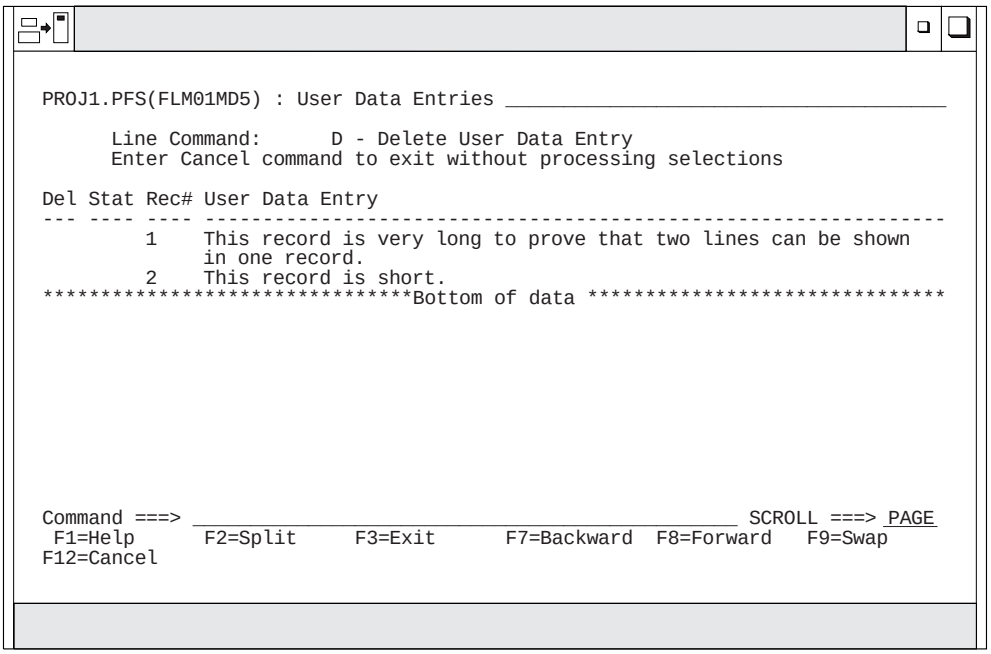


Figure 56. User Data Entries (FLMUSE#P)

The fields on the User Data Entries panel are:

Del	You specify that you want to delete the user data entry record when you select `D` in this field.
Stat	SCLM displays `*SEL` to indicate the user data entry record you selected. Enter the END command to confirm the delete request.
Rec#	SCLM displays a record number with the first line of each user data entry record.
User Data Entry	Project-specific information entered into the accounting record by the SAVE service. The user data entry record can span two lines for a maximum of 128 characters.

Build Map Record

Enter the M line command on the SCLM Library Utility panel or on the member selection list to display a build map record. The Build Map Record panel, shown in Figure 57 on page 173, displays the fixed build map information SCLM records for a member.

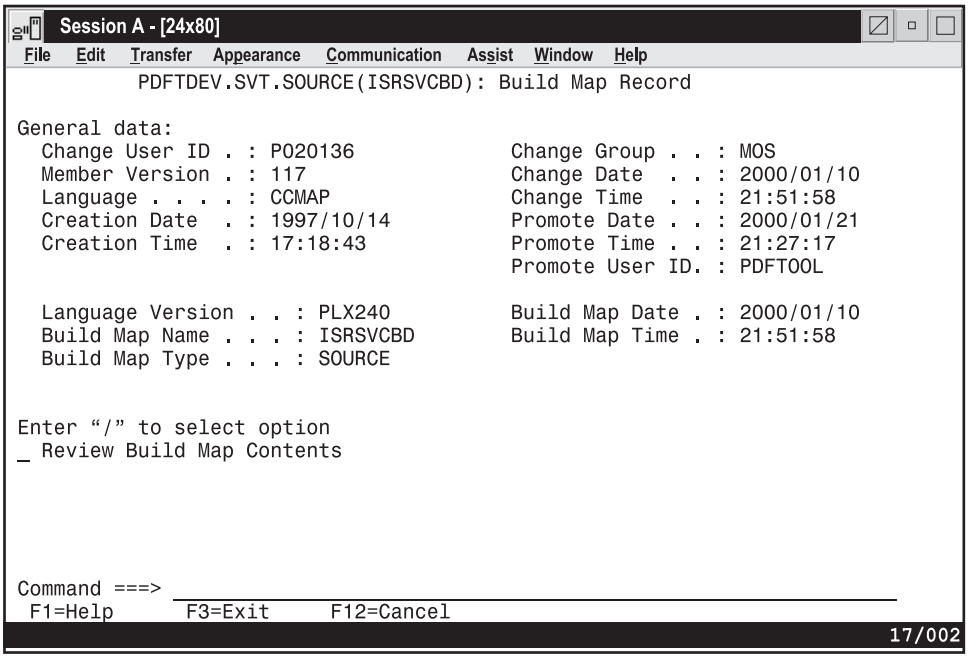


Figure 57. Build Map Record (FLMUSB#P)

The fields on the Build Map Record panel are:

Change User ID	The user ID of the person who made the last update to the member.
Member Version	The number of times that the build map has been generated by SCLM. The first time a build map is generated a version of 1 is used.
Language	The language of the build member. This language is determined by SCLM Build; it is not specified by the user or the project manager.
Creation Date	The date the build map was first created.
Creation Time	The time the build map was first created.
Change Group	The name of the group in which the member was last updated.
Change Date	The last date the member was modified.
Change Time	The last time the member was modified.
Promote Date	The date the member was last promoted.
Promote Time	The time the member was last promoted.
Promote User ID	The user ID of the person who last promoted the member.
Translator Version	The version of the translator used during build processing.
Language Version	The version of the language that SCLM uses in language-based builds.
Build Map Name	The name of the member with which the build map is associated.
Build Map Type	The type of the member with which the build map is associated.
Build Map Date	The date of the build that created the build map.
Build Map Time	The time of the build that created the build map.
Review Build Map Contents	SCLM displays the Build Map Contents panel, shown in Figure 58 on page 174, when you select this field.

Build Map Contents

When you enter a / in the **Review Build Map Contents** field, SCLM displays the build map contents in a browse data set, as shown in Figure 58. The data set shows the contents of a build map record for an architecture defined in a CC architecture member.

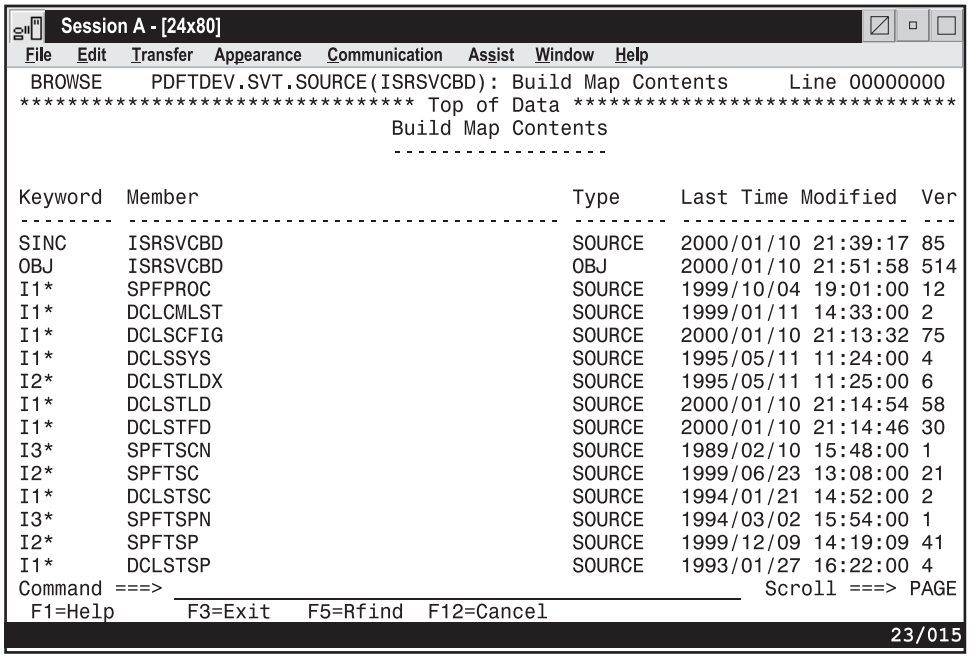


Figure 58. Build Map Contents (FLMUSBRP)

The fields on the Build Map Contents panel are:

Keyword	You can use certain keywords to identify architecture information. See "Architecture Statements" on page 254 for more details. The internal build map keywords, denoted with an asterisk, are described as follows. The architecture member example contains two keywords: OBJ, and LIST. If a keyword is denoted with an asterisk (*), it includes references found in source member FLM01MD5.
Member	The name of the member referenced in the architecture member.
Type	The name of the type containing the member.
Last Time Modified	For an EDITABLE member, this field is the last time SCLM parsed and stored the specified member. For SCLM-generated (NON-EDIT) members, such as OBJ and LIST, this field is the last time SCLM generated the member.

Internal Keywords

Keywords that SCLM uses to track references. The internal keyword `I#` indicates the group in which the members were first referenced. The following internal keywords are produced by SCLM internal processing and supported by SCLM. They cannot be used in the actual architecture definitions.

Keyword	Description
PINCL*	An architecture definition that generates the output shown on the previous build map entry. The output represents an input to the translate process.
INT*	An intermediate that the build of the member being viewed generated. This keyword represents the output of a translate process.
INTDEP*	Intermediate member on which the member being viewed is dependent. This keyword represents the input of a translate process.
WITH*	Indicates an upward dependency.
DYNI*	Indicates a dynamic include.
Ix*	Includes as determined by the accounting record for the main source member, where *x* is in the range (1–99).
EXTDPEND*	Indicates an external dependency.

Authorization Code Update

Type `U` on the Library Utility panel or the member selection list to display the Authorization Code Update panel. Figure 59 shows the panel SCLM displays for you to update the authorization code for a member.

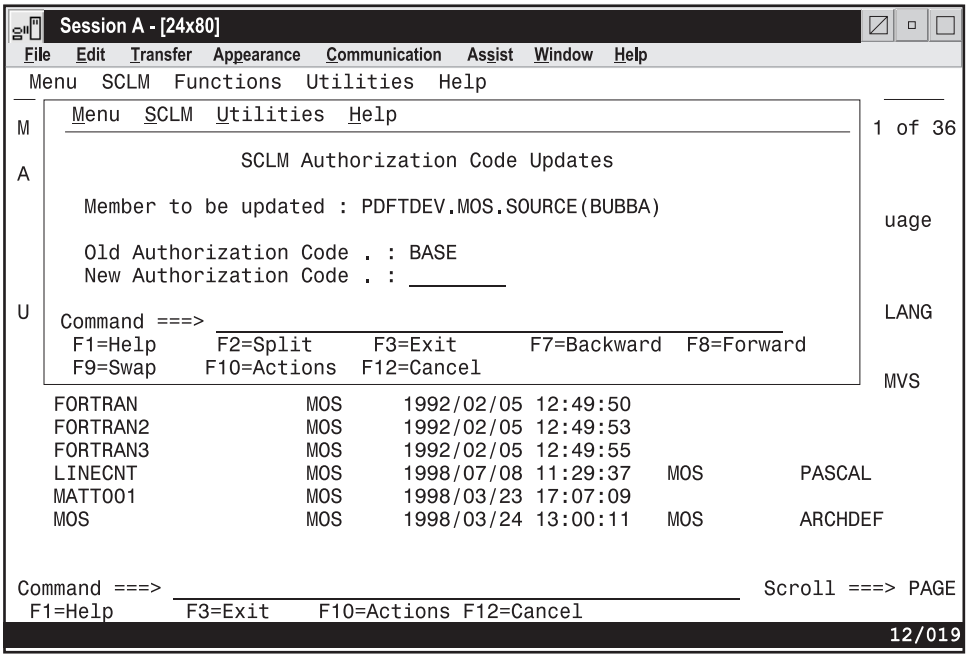


Figure 59. Authorization Code Update (FLMUSU#P)

The fields on the Authorization Code Update panel are:

Member to be updated	The member name you entered in the **Member** field on the SCLM Library Utility panel.
Old Authorization Code	The current authorization code for the member.
New Authorization Code	The new authorization code for the member. Enter the new authorization code in this field. Then press Enter to confirm the update request and update the authorization code, or enter END to cancel the update request. Authorization codes cannot contain commas.

Migration Utility

Using the migration utility, you can introduce members or groups of members to an SCLM project and place them under SCLM control in a development group. The migration utility also lets you verify authorization codes, prohibit simultaneous updates of members, and collect statistical, dependency, and historical information for each member processed without using the SCLM edit function. SCLM collects *dependency* information, which identifies software components that need another software component to complete successfully.

Before you start MIGRATE, the members must exist in the development library you specify. Upon successful completion of MIGRATE, each member selected will have valid SCLM accounting information. A typical scenario used to migrate existing project data follows:

1. Copy all of the members that have the same language into a development library.
2. Start MIGRATE using * for the member pattern and the appropriate language to parse all members and store their statistical, dependency, and historical information.
3. Copy all of the members that have a different language into the development library.
4. Start MIGRATE again using * for the member pattern and the new language.
5. Continue until all of the members have been migrated.

If some of the members have SCLM accounting information, the MIGRATE service verifies that the accounting information matches the member in the development library. MIGRATE takes no action for members that already have valid SCLM accounting information, unless executed in forced mode.

Use this utility when you have a large number of members that have not been entered in your project database, such as members that you did not create with the SCLM edit function.

In addition to the SCLM editor, the Migration Utility lets you indicate the members you want tracked. Use this utility to enter one or more members into a database of a project (for example, during a conversion to SCLM). In development groups, you can also use it to lock, parse, and create accounting records for members that have not been registered to SCLM.

Like the SCLM editor, the migration utility verifies authorization codes, prohibits simultaneous updates of members, and collects statistical, dependency, and

historical information for every member processed. SCLM stores this information in the database of a project. For a complete description of the lock, parse, and store process, refer to the *SCLM Reference*.

Figure 60 shows the panel that appears when you select Option 3, Migration, from the Utilities Panel.

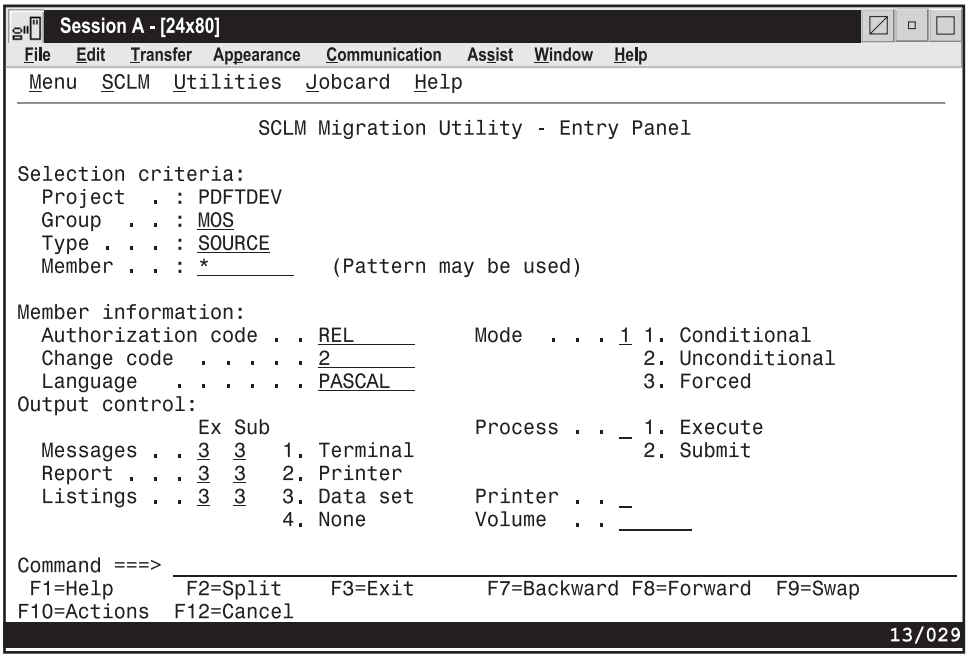


Figure 60. SCLM Migration Utility (FLMUM#P)

Note: The NRETREIV command key is enabled to work with this option. See "Name Retrieval with the NRETRIEV command" on page 145 for more information.

The action bar displays the same choices as those discussed in "SCLM Main Menu Action Bar Choices:" on page 148. An additional choice is Jobcard.

The fields for the Migration Utility - Entry panel are:

Project	The project that you specified on the SCLM Main Menu. You cannot change this field. An **Alternate** field also appears if you specified an alternate project.
Group	The group in which the members to be migrated are located. This group must be defined in the project definition and must be a development group.
Type	The type in which the members to be migrated are located. This type must be defined in the project definition.
Member	The name of the member you want processed. You can use patterns for the member name. See "Specifying Selection Criteria" on page 180 for details.
Authorization code	The authorization code for a member. SCLM cannot process a member if the authorization code assigned to a member is not in the group being accessed. Authorization codes cannot contain commas.

Change code	The ç for the member. To enter a different ç for the member, type over the displayed ç. A change code verification routine can verify the code you entered before it processes the member. Change codes cannot contain commas.
Language	The language of the member. Refer to the *SCLM Reference* for a list of languages for which SCLM supplies parsers.
Mode	Select one of the following: **Conditional** To stop processing members if migrate discovers an error that is greater than the GOODRC parameter specified for a language parser in the project definition. If you have a list of members that you want to place under SCLM control, and migrate fails for one of those members, processing stops after the first error. Migrate does not process any other members that match the specified criteria. **Unconditional** To continue processing regardless of errors discovered during parsing of each member. If you have a list of members that you want to place under SCLM control, migrate attempts to process all the members matching the selection criteria, regardless of any errors encountered. **Forced** Forces SCLM to create a new accounting record for the members specified regardless of previous status. Processing will stop after the first error is encountered. If you have a list of members that need to be changed, migrate will create new accounting records for any members specified. This can be used to update language, authorization code or change code information for the specified members.
Output control	Specify the destination for messages, report, and listings when they are executed (Ex) or submitted (Sub), by entering the corresponding destination number: 1 for Terminal, 2 for Printer, 3 for Dataset, or 4 for None.
Process	You can call the processing part of the migration utility from the interactive or batch environment by selecting Execute or Submit, respectively. If you request batch processing by selecting Submit, you must specify the job statement information that is used in the JCL generated for batch processing. For information on using a unique jobname on the jobcard in batch processing, see “Batch Processing” on page 230.
Printer	Specify the printer output class.
Volume	Specify the volume on which SCLM should save data sets.

Database Contents Utility

You can use the SCLM database contents utility to retrieve information about the project hierarchy from the project database and produce a report. You control the order and format of the data in the report. The utility generates a report that lists the members that match your selection criteria.

This accounting data can then be extracted for members in the database that meet the selection criteria you specify.

The output from the database contents utility can be used as input to other project-defined tools or as input to the SCLM services using the FILE format of FLMCMD.

Figure 61 shows the panel that appears when you select Option 4, Database Contents, from the Utilities panel.

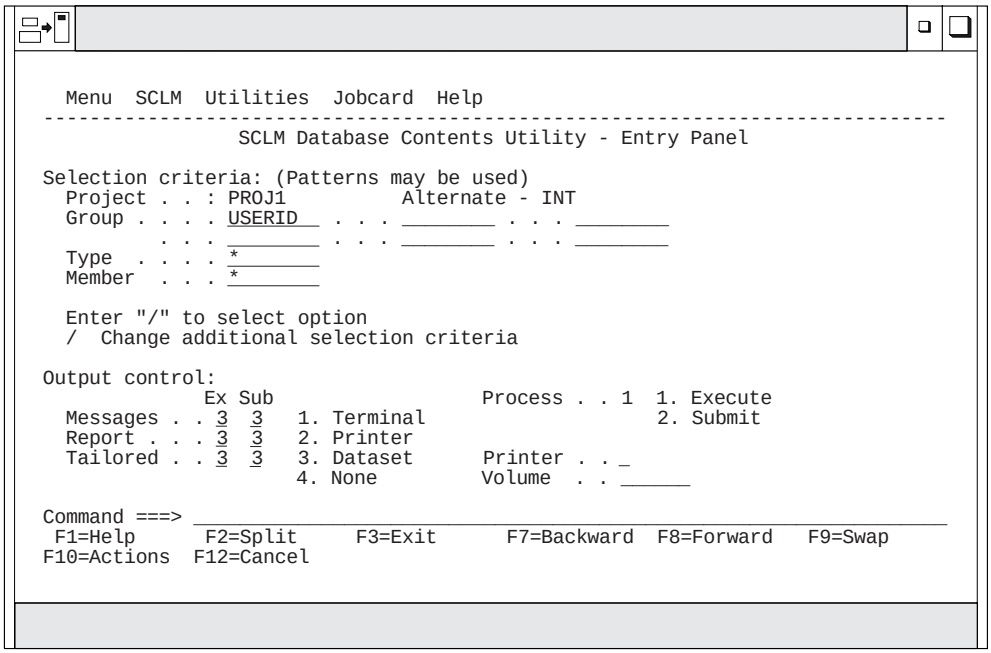


Figure 61. SCLM Database Contents Utility (FLMRC#P)

You can use patterns for all of the selection criteria fields (except **Project** and **Alternate**), as described in "Specifying Selection Criteria" on page 180.

The fields on the Database Contents Utility panel are:

Project	The project that you specified on the SCLM Main Menu. An **Alternate** field also appears if you specified an alternate project.
Group	The groups that are to be reported. Only groups defined to the project definition are allowed.
Type	The name of the type you want processed. Only types defined to the project definition are allowed.
Member	The name of the member you want processed.
Change additional selection criteria	Select this field if you want to change the additional selection criteria. The panel shown in Figure 62 on page 181 appears when you select this. If you change additional selection criteria, the changes are carried over from one execution to another. If you do not select this field, and thus do not change the additional criteria, the criteria from the last report are used.

Output control	Specify the destination for messages, reports, and tailored output when they are executed (Ex) or submitted (Sub), by entering the corresponding destination number: 1 for Terminal, 2 for Printer, 3 for Data set, or 4 for None. You cannot select Terminal for both **Report** and **Tailored Output**. Similarly, you cannot select None for both **Report** and **Tailored Output**. If the tailored output is to be used as input to a tool or to the SCLM services, Data set should be specified for **Tailored Output**. If you enter Terminal, Printer, or Data set in the **Tailored Output** field, the panel shown in Figure 64 on page 185 appears.
Process	You can call the processing part of the database contents utility from the interactive or batch environment by selecting Execute or Submit, respectively. If you request batch processing by selecting Submit, you must specify the job statement information that is used in the JCL generated for batch processing.
Printer	Specify the printer output class.
Volume	Specify the volume on which SCLM should save data sets. For information on using a unique jobname on the jobcard in batch processing, see "Batch Processing" on page 230.

Specifying Selection Criteria

You can use patterns to specify a variety of acceptable values for the accounting information fields. A pattern consists of alphanumeric characters and three special characters: an asterisk (*), a logical NOT symbol (¬), and an equal sign (=).

Use an asterisk to match any string of characters including the null string. You can use it more than once.

Use the logical NOT symbol (¬) to negate the result of a match with the pattern. You can specify it only once. The logical NOT symbol is removed from the pattern before a match is attempted. Therefore, the position of the logical NOT symbol within the pattern is not significant.

Use an equal sign (=) to indicate all groups that are at the same layer in the hierarchy as the group you specify. An equal sign can only be specified once in the pattern.

You should use the equal sign only in the group field, and you should not use the equal sign in conjunction with other wildcard characters. If you use the equal sign, you must specify a valid group name. The name specified is taken literally.

Note: Do not use an equal sign (=) as the first character in a pattern because it is a special character in ISPF.

Use the patterns shown in Table 16 to select accounting information.

Table 16. Pattern Examples

Pattern	Match
AB*Z	ABZ,ABCZ,ABCZYZ,ABCABZ
¬AB*Z	ABC,XABZ,ABZX
*AB*Z	ABZ,XABZ,ABCABZ,ABCZ,ABCZYZ
DEV1=	DEV1,DEV2

Table 16. Pattern Examples (continued)

Pattern	Match
STAGE1=	STAGE1,STAGE2

Note: See Figure 42 on page 143 for an illustration of the hierarchy represented in the last two rows.

The portion of the project database that SCLM displays is determined by the parameters you specify.

The panel in Figure 62 appears if you select **Change additional selection criteria** field on the Database Contents Utility panel.

If you do not select this, the panel does not appear and the reports are generated with the values that already exist on the Additional Selection Criteria panel.

```
  Menu
-------------------------------------------------------------------------------
            SCLM Database Contents - Additional Selection Criteria

Selection criteria: (Patterns may be used)
  Authorization code  . . REL      _         Data type . . 1  1. Account
  Change code . . . . . . *        _                          2. Build map
  Change group  . . . . . USERID   _                          3. Both
  Change user id  . . . . *        _
  Language  . . . . . . . *        _         Enter "/" to select option
                                             /  First occurrence only
Hierarchy search information:
  Architecture Control . . 3  1. In          Scope . . 1  1. Normal
                              2. Out                      2. Subunit
                              3. Not used                 3. Extended

  Architecture Group . . . USERID
  Architecture Type  . . . ARCHDEF
  Architecture Member  . . _________


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

Figure 62. SCLM Database Contents - Additional Selection Criteria (FLMRCA)

The fields on the Additional Selection Criteria panel allow you to specify accounting and architecture information that the utility uses to identify the members to be processed.

Accounting Information Fields

When you specify values or patterns for the accounting information fields, the utility selects any member that has accounting information matching all of the patterns or values for all fields you specify.

Use the following accounting information fields to select members:

Authorization code	Members that are assigned an authorization code matching the authorization code. Authorization codes cannot contain commas. The logical NOT symbol (¬) in the pattern specifies only the members that are not assigned an authorization code matching the pattern.
Change code	Members that can be edited that were assigned a change code matching the change code pattern. Change codes cannot contain commas. Only one of the change codes assigned to the member must match the pattern. The logical NOT symbol (¬) in the pattern specifies only the members that are not assigned a change code matching the pattern.
Change group	Members that were last changed in a group matching the change group pattern.
Change user id	Members that were last changed by the user ID matching the change user ID pattern.
Language	Members whose language matches the language pattern.
Data type	Specify the following: **Account** To report exclusively on accounting information. **Build Map** To report exclusively on build map information. **Both** To report on build map and accounting information. Data type defaults to Account if nothing is specified.
First occurrence only	If you select this and use more than one group pattern, a precedence system determines which members are selected. The group1 pattern takes precedence over the group2 pattern, which takes precedence over the group3 pattern, and so on. If SCLM finds versions of a member in groups matching more than one pattern, it selects only the version at the group with the most precedence. If more than one version of the member matches the pattern with the most precedence, it selects all of those versions. If you do not select this field, SCLM selects all versions of all members.

Hierarchy search information

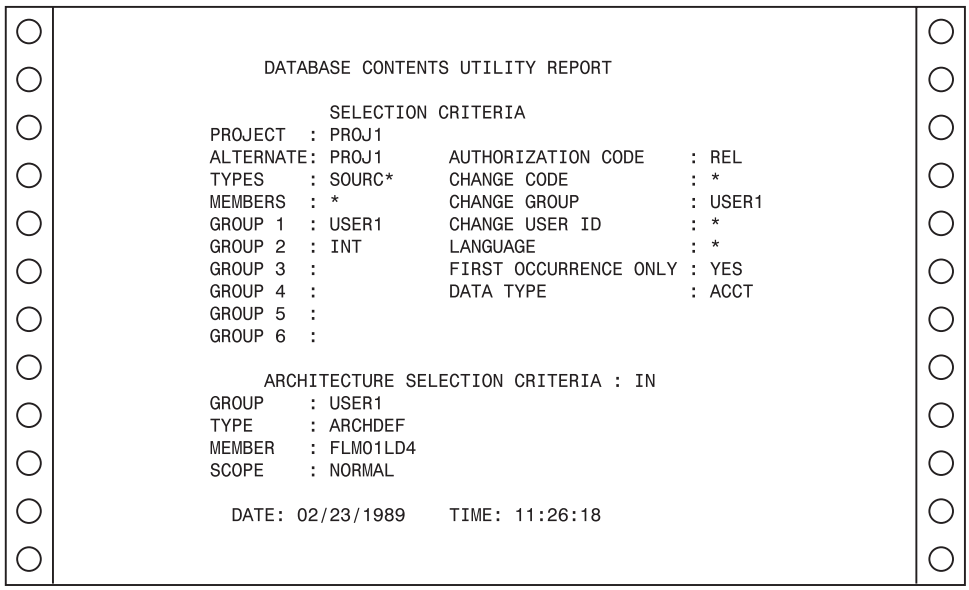
These fields allow you to use architecture definition criteria to select members. The architecture definition fields identify subapplications or software components.

To guarantee correct data, use the build function to update the architecture in the **Architecture Control** field. If you specify an architecture that has never been built, none of the members is selected. If you specify an architecture that has been built but is out of date, the resulting data is inaccurate. Promote the architecture in report-only mode to see which components are out of date. Patterns are not valid for architecture definition fields.

Architecture Control	Specify the following: **In** To select members controlled by the architecture definition. **Out** To select members not controlled by the architecture definition. **Not used** To indicate that an architecture definition is not used to identify selected members.
Architecture Group	The group identifying the lowest group in the hierarchy where SCLM should find the architecture definition.
Architecture Type	The type containing the architecture definition that controls the selected members.
Architecture Member	The member containing the architecture definition that controls the selected members.
Scope	Specify the following architecture scope: **Normal** To select members that do or do not have compilation unit dependencies. **Subunit** To select members that do have compilation unit dependencies. **Extended** To select members that do have compilation unit dependencies.

The database contents report contains a list of all members that you select from the selection criteria. If you request tailored output, SCLM generates the data set from this list of accounting and build map information.

Figure 63 on page 184 shows an example of a database contents utility report that SCLM generates when you enter `NONE` in the **Tailored Output** field on the SCLM Database Contents Utility panel.



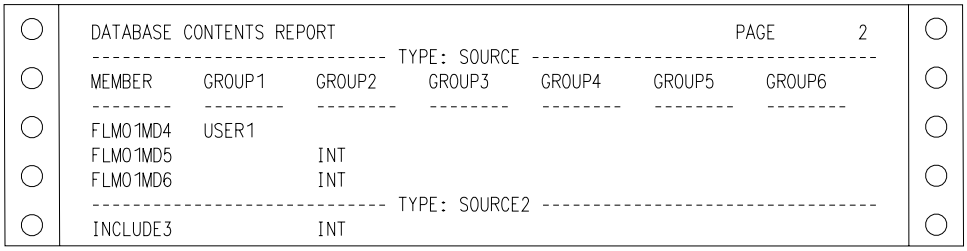
```
                    DATABASE CONTENTS UTILITY REPORT

                              SELECTION CRITERIA
          PROJECT  : PROJ1
          ALTERNATE: PROJ1       AUTHORIZATION CODE     : REL
          TYPES    : SOURC*      CHANGE CODE            : *
          MEMBERS  : *           CHANGE GROUP           : USER1
          GROUP 1  : USER1       CHANGE USER ID         : *
          GROUP 2  : INT         LANGUAGE               : *
          GROUP 3  :             FIRST OCCURRENCE ONLY  : YES
          GROUP 4  :             DATA TYPE              : ACCT
          GROUP 5  :
          GROUP 6  :

               ARCHITECTURE SELECTION CRITERIA : IN
          GROUP    : USER1
          TYPE     : ARCHDEF
          MEMBER   : FLMO1LD4
          SCOPE    : NORMAL

            DATE: 02/23/1989    TIME: 11:26:18
```

Figure 63. Database Contents Utility Report (Part 1 of 2)



```
DATABASE CONTENTS REPORT                                          PAGE       2
------------------------------ TYPE: SOURCE ----------------------------------
MEMBER     GROUP1     GROUP2     GROUP3     GROUP4     GROUP5     GROUP6
--------   --------   --------   --------   --------   --------   --------
FLMO1MD4   USER1
FLMO1MD5              INT
FLMO1MD6              INT
------------------------------ TYPE: SOURCE2 ---------------------------------
INCLUDE3              INT
```

Figure 63. Database Contents Utility Report (Part 2 of 2)

Note: An asterisk (*) next to the group name on a report indicates that the member represents build map information.

Tailored Output

If you want to tailor the database contents output, enter `Terminal`, `Printer`, or `Dataset` in the **Tailored Output** field on the Database Contents Utility panel. The Customization Parameters panel appears, shown in Figure 64 on page 185, which you use to generate the tailored output.

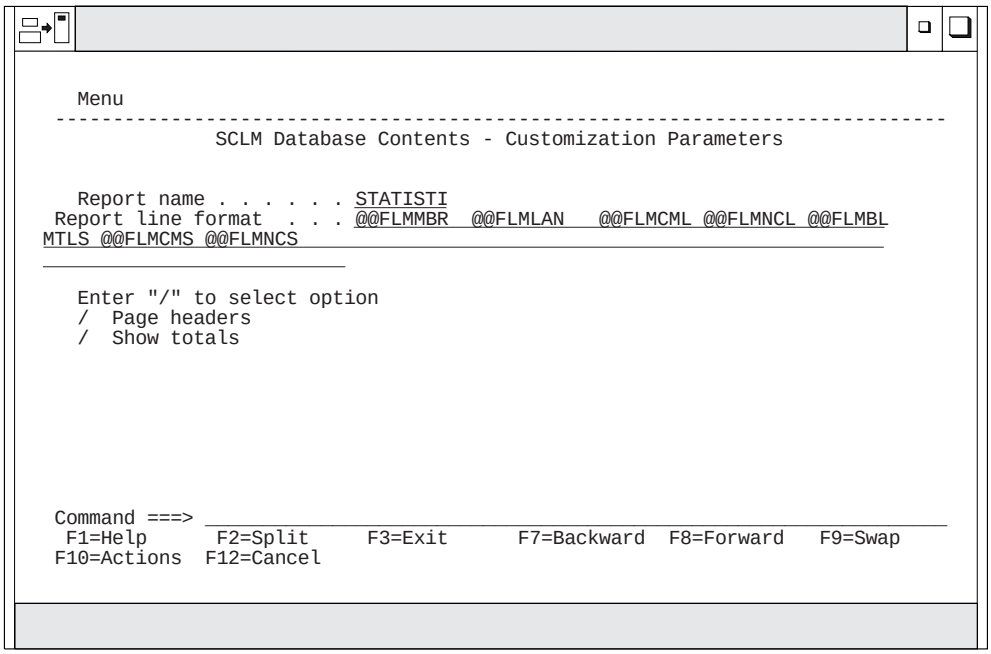


Figure 64. SCLM Database Contents - Customization Parameters (FLMRCT)

The fields on the Customization Parameters panel are:

Report name	The title of the report in the tailored output. The maximum length is 35 characters. Do not use commas in this field. The default value for Report name is STATISTICS REPORT.
Report line format	The format of a line of data in the tailored output. The line format can be up to 160 characters long. Report line format has a default value, which is used when no values are specified: `@@FLMMBR @@FLMLAN @@FLMCML @@FLMNCL @@FLMBLL @@FLMTLS @@FLMCMS @@FLMNCS` If you use SCLM variables with data lengths greater than 8 characters, place these variables at the end of the report line to ensure that the columns in the report line up evenly. You can use any string or character as a literal. When you use literals, the string prints once on each output line. The report line has a maximum size of 2048 characters. The tailored output prints 80 characters per line. This can produce multiple 80-character lines for one report line. Press Enter to confirm these requests or enter END to cancel them.

Page headers	Select **Page headers** to include page and column header information in the tailored output. If you want to output a page header, input parameter information appears in the tailored output. You can also specify a title. Data will be positioned in column 2 of the tailored output. Column 1 is used for carriage returns. If you do not select **Page headers**, page headers and carriage returns are suppressed. The data will be positioned in column 1 of the tailored output. The default value for **Page headers** is that they are selected.
Show totals	Select this to total the numeric data fields and show the totals in the tailored output. SCLM outputs a summary line at the end of the output that totals the values of the numeric fields in the output. The output also includes a count of the number of members reported. The default value for **Show totals** is that they are selected.

Figure 65 shows an example of a tailored output. The title of the report is `Sample Report`. The report line format, specified as `@@FLMPRJ @@FLMGRP @@FLMTYP @@FLMMBR`, causes the utility to generate output consisting of the members reported in the database contents report and their associated included members.

Tailored Output Examples

The tailored output that appears in Figure 65 on page 187 is a formatted representation of the accounting and build map information of the members that matched the selection criteria. The tailored output format specification consists of SCLM variables and constant values. The tailored output displays the SCLM variables as headers over the lines of variable values.

The *ISPF Software Configuration and Library Manager (SCLM) Reference* provides a list of SCLM variables that can be used in the database contents utility.

```
                DATABASE CONTENTS UTILITY REPORT

                        SELECTION CRITERIA
PROJECT  : PROJ1
ALTERNATE: PROJ1      AUTHORIZATION CODE    : REL
TYPES    : SOURC*     CHANGE CODE           : *
MEMBERS  : *          CHANGE GROUP          : USER1
GROUP 1  : USER1      CHANGE USER ID        : *
GROUP 2  : INT        LANGUAGE              : *
GROUP 3  :            FIRST OCCURRENCE ONLY : YES
GROUP 4  :            DATA TYPE             : ACCT
GROUP 5  :
GROUP 6  :

     ARCHITECTURE SELECTION CRITERIA : IN
GROUP    : USER1
TYPE     : ARCHDEF
MEMBER   : FLMO1LD4
SCOPE    : NORMAL

     CUSTOMIZATION PARAMETERS
PAGE HEADERS : YES
SHOW TOTALS  : YES
REPORT NAME  : SAMPLE REPORT

  DATE: 2000/01/06    TIME: 09:52:17
```

```
                                                                    PAGE        2
SAMPLE REPORT

@@FLMALT @@FLMGRP @@FLMTYP @@FLMMBR
--------------------------------------------------------------------------------
PROJ1    USER1    SOURCE   FLMO1MD4
PROJ1    INT      SOURCE   FLMO1MD5
PROJ1    INT      SOURCE   FLMO1MD6
PROJ1    INT      SOURCE2  INCLUDE3
--------------------------------------------------------------------------------
PROJ1                              4
```

Figure 65. Database Contents Utility Tailored Output

The tailored output examples in figures 66 through 69 show examples of change code, accounting statistics, source listing, and cleanup reports.

Change Code Report: The report name is CHANGE CODE REPORT.

The report line format input for this example is: `@@FLMGRP @@FLMTYP @@FLMMBR @@FLM$CD @@FLM$CC`. The page headers appear on all pages of the report. Totals do not appear. Figure 66 on page 188 shows the tailored output.

```
                                                                    PAGE        2
CHANGE CODE REPORT

@@FLMGRP @@FLMTYP @@FLMMBR  @@FLM$CD @@FLM$CC
--------------------------------------------------------------------------------
USER1    SOURCE   FLMO1MD4
INT      SOURCE   FLMO1MD5  02/14/89 2
                            02/01/89 PR3573

                            02/01/89 CR3582

                            02/01/89 PR3456

INT      SOURCE   FLMO1MD6  02/14/89 2
                            02/01/89 PR3573

INT      SOURCE2  INCLUDE3  02/14/89 2
```

Figure 66. Change Code Report, Page 2

Accounting Statistics Report: The report name is ACCOUNTING STATISTICS REPORT.

The report line format input for this example is: `@@FLMMBR @@FLMLAN @@FLMTLL @@FLMCML @@FLMNCL @@FLMBLL @@FLMTLS @@FLMCMS`.

The page headers appear on all pages of the report. Totals appear for all numeric data. Figure 67 shows the tailored output.

```
                                                                    PAGE        2
ACCOUNTING STATISTICS REPORT

@@FLMMBR @@FLMLAN @@FLMTLL @@FLMCML @@FLMNCL @@FLMBLL @@FLMTLS @@FLMCMS
--------------------------------------------------------------------------------
FLMO1MD4 PASCAL          8         0         4         4         2         0
FLMO1MD5 PASCAL         13         2         5         6         4         2
FLMO1MD6 PASCAL          8         0         4         4         2         0
INCLUDE3 PASCAL          5         5         0         0         5         5
--------------------------------------------------------------------------------
       4                34         7        13        14        13         7
```

Figure 67. Accounting Statistics Report, Page 2

Source Listing Report: This example shows a generated script data set that the SCRIPT/VS processor can process.

The report line format input for this example is: `.IM @@FLMMBR`.

The report does not have page headers, totals, or a name. Figure 68 shows the tailored output.

```
.IM FLMO1MD4
.IM FLMO1MD5
.IM FLMO1MD6
.IM INCLUDE3
```

Figure 68. Source Listing Report

Cleanup Report: The cleanup data set is a command data set that can be passed as input to the SCLM command processor. See *ISPF Software Configuration and Library Manager (SCLM) Reference* for more information on the SCLM command processor.

The report line format input for this example is:
`DELETE,@@FLMPRJ,@@FLMALT,@@FLMGRP,@@FLMTYP,@@FLMMBR.`

The report does not have page headers, totals, or a name. Figure 69 shows the sample tailored output.

```
DELETE,PROJ1   ,PROJ1   ,USER1   ,SOURCE  ,FLM01MD4
DELETE,PROJ1   ,PROJ1   ,INT     ,SOURCE  ,FLM01MD5
DELETE,PROJ1   ,PROJ1   ,INT     ,SOURCE  ,FLM01MD6
DELETE,PROJ1   ,PROJ1   ,INT     ,SOURCE2 ,INCLUDE3
```

Figure 69. Cleanup Report

Architecture Report Utility

The architecture report provides listings of all the components in a given application. The report generator examines the requested architecture and all of its references, and then constructs a formatted report. The report lists software components in each type referenced by the architecture. One advantage of the report is that it helps you to eliminate unnecessary code. The title page of the report identifies the date and time SCLM generated the report, names the architecture member you requested, and is based on the report cutoff you select. It also identifies any alternate project definition used.

The report is divided into two sections:

- Architecture

  Lists all architecture and source members subordinate to a given architecture to the report cutoff you specify. The architecture information is particularly useful during the development stages of a project to identify the current status of the application architecture. It is also useful at any time to determine a list of the software components of an application.

  The report uses an indentation format to present a visual concept of the structure of the application. It also lists the number architecture types processed.
- Cross-reference

  Lists all the members, by type, that are referenced by members in the first part of the report. Use this information to determine the origin of a member.

Figure 71 on page 192 shows an example of an architecture report.

SCLM displays the panel in Figure 70 on page 190 when you select Option 5, Architecture Report, on the Utilities panel.

Note: Compilation unit dependencies are not used to generate the architecture report.

The architecture report is divided into three parts: a header, architecture information, and cross-reference information. The architecture report header lists the accounting and architecture selection criteria plus the customization parameters you specify. The architecture information lists all of the software components, by

type, in a specified application. This part of the report can help you eliminate unnecessary code. The cross-reference information indicates where a given software component is imbedded in the architecture of the application.

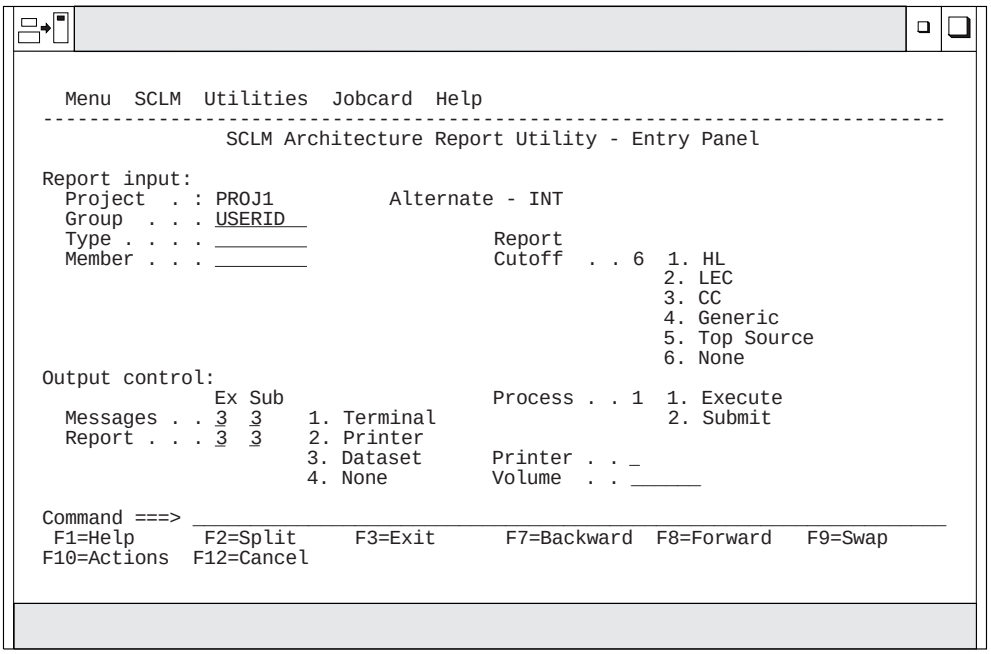


Figure 70. SCLM Architecture Report (FLMRA#P)

The fields on the SCLM architecture report Utility - Entry panel are:

Project	The project that you specified on the SCLM Main Menu. An **Alternate** field also appears if you specified an alternate project definition.
Group	The group used to identify the lowest group in the hierarchy where the architecture begins.
Type	The type containing the architecture definition that controls the selected member.
Member	The member containing the architecture definition.

Report Cutoff	You must specify one of the following report cutoff values (which determine the depth of the report): **HL (High-level)** To list only the HL architecture members in the application represented by the architecture member you specified in the **Member** field. **LEC (Linkedit control)** To list all of the HL and LEC architecture members in the application represented by the architecture member you specified in the **Member** field. **CC (Compilation control)** To list all of the HL, LEC, CC, Generic, and INCLD'ed members in the application represented by the architecture member you specified in the **Member** field. **GEN (Generic)** To list all of the HL and generic architecture members in the application represented by the architecture member you specified in the **Member** field. **Top Source** To list all of the HL, LEC, CC, Generic, and INCL'ed members and the top source members in the application represented by the member you specified in the **Member** field. **None** To list all HL, LEC, CC, and generic architecture members in each of the types and all source member names down to the lowest include group in the application represented by the architecture member you specified in the **Member** field. The default value for **Report Cutoff** is None.
Output control	Specify the destination for messages and report when they are executed (Ex) or submitted (Sub), by entering the corresponding destination number: 1 for Terminal, 2 for Printer, 3 for Dataset, or 4 for None.
Process	You can call the processing part of the architecture report utility from the interactive or batch environment by selecting Execute or Submit, respectively. If you request batch processing by selecting Submit, you must specify the job statement information that is used in the JCL generated for batch processing. For information on using a unique jobname on the jobcard in batch processing, see "Batch Processing" on page 230.
Printer	Specify the printer output class.
Volume	Specify the volume on which SCLM should save data sets.

Architecture Report Example

Figure 71 on page 192 shows an example of the architecture report with a report cutoff of NONE. Figure 72 on page 195 shows an example of the architecture report with a report cutoff of LEC.

The architecture report provides lists of all the components in an application. The title page identifies the date and time the report was generated, the architecture member requested, and the report cutoff. It also identifies the alternate project definition, if specified.

Architecture Report Utility

```
*******************************************************************************
*******************************************************************************
**                                                                           **
**            SOFTWARE CONFIGURATION AND LIBRARY MANAGER (SCLM)               **
**                                                                           **
**                             ARCHITECTURE REPORT                           **
**                                                                           **
**                            2000/01/06   00:01:30                          **
**                                                                           **
**                                                                           **
**                                                                           **
**                              PROJECT:   PROJ1                             **
**                              GROUP:     DEV1                              **
**                              TYPE:      ARCHDEF                           **
**                              MEMBER:    FLM01SB2                          **
**                              CUTOFF:    NONE                              **
**                                                                           **
**                                                                           **
*******************************************************************************
*******************************************************************************
===============================================================================
*                                                                             *
*                              ARCHITECTURE REPORT                            *
*                                                                             *
*  H = HIGH LEVEL        C = COMPILATION CONTROL  T = TOP SOURCE E = ERROR     *
*  L = LINKEDIT CONTROL  G = GENERIC              I = INCLUDED   D = DEFAULT   *
*                                                                             *
===============================================================================

CODE:  H   MEMBER:  FLM01SB2

----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+---

H FLM01SB2 ARCHDEF
L  FLM01LD4 ARCHDEF
D   FLM01MD4 SOURCE
T    FLM01MD4 SOURCE
I     FLM01EQU SOURCE
D   FLM01MD6 SOURCE
T    FLM01MD6 SOURCE
I     FLM01EQU SOURCE
D   FLM01MD5 SOURCE
T    FLM01MD5 SOURCE
I     FLM01EQU SOURCE
L  FLM01LD3 ARCHDEF
D   FLM01MD3 SOURCE
T    FLM01MD3 SOURCE
I     FLM01EQU SOURCE
D   FLM01MD6 SOURCE
T    FLM01MD6 SOURCE
I     FLM01EQU SOURCE
D   FLM01MD5 SOURCE
T    FLM01MD5 SOURCE
I     FLM01EQU SOURCE

NUMBER OF HIGH LEVEL MEMBERS PROCESSED          =    1
NUMBER OF LINK EDIT CONTROL MEMBERS PROCESSED   =    2
NUMBER OF GENERIC MEMBERS PROCESSED             =    0
```

Figure 71. Architecture report with cutoff of NONE (Part 1 of 3)

```
NUMBER OF DEFAULT MEMBERS PROCESSED             =    4
NUMBER OF COMPILATION CONTROL MEMBERS PROCESSED  =    0
NUMBER OF TOP MEMBERS PROCESSED                 =    4
NUMBER OF INCLUDED MEMBERS PROCESSED            =    1
NUMBER OF ERROR MEMBERS FOUND                   =    0
================================================================================
*                                                                              *
*                  CROSS REFERENCE FOR TYPE:     SOURCLST                      *
*                                                                              *
================================================================================

MEMBER    REF. ARCH. MEM.  TYPE       KEYWORD   INCLUDE-SET
------    ---------------  ----       -------   -----------

FLM01MD3      FLM01MD3     SOURCE     LIST
FLM01MD4      FLM01MD4     SOURCE     LIST
FLM01MD5      FLM01MD5     SOURCE     LIST
FLM01MD6      FLM01MD6     SOURCE     LIST

TOTAL MEMBERS PROCESSED FOR TYPE  =  4

================================================================================
*                                                                              *
*                  CROSS REFERENCE FOR TYPE:     OBJ                           *
*                                                                              *
================================================================================

MEMBER    REF. ARCH. MEM.  TYPE       KEYWORD   INCLUDE-SET
------    ---------------  ----       -------   -----------

FLM01MD3      FLM01MD3     SOURCE     OBJ
FLM01MD4      FLM01MD4     SOURCE     OBJ
FLM01MD5      FLM01MD5     SOURCE     OBJ
FLM01MD6      FLM01MD6     SOURCE     OBJ

TOTAL MEMBERS PROCESSED FOR TYPE  =  4

================================================================================
*                                                                              *
*                  CROSS REFERENCE FOR TYPE:     SOURCE                        *
*                                                                              *
================================================================================

MEMBER    REF. ARCH. MEM.  TYPE     KEYWORD   INCLUDE-SET
------    ---------------  ----     -------   -----------

FLM01EQU      FLM01MD4     SOURCE     I1
              FLM01MD4     SOURCE
              FLM01MD3     SOURCE     I1
              FLM01MD3     SOURCE
              FLM01MD6     SOURCE     I1
              FLM01MD6     SOURCE
              FLM01MD5     SOURCE     I1
              FLM01MD5     SOURCE
FLM01MD3      FLM01MD3     SOURCE     SINC
              FLM01MD3     SOURCE     PROM
              FLM01LD3     ARCHDEF    INCLD
FLM01MD4      FLM01MD4     SOURCE     SINC
```

Figure 71. Architecture report with cutoff of NONE (Part 2 of 3)

```
              FLM01MD4     SOURCE     PROM
              FLM01LD4     ARCHDEF    INCLD
FLM01MD5      FLM01MD5     SOURCE     SINC
              FLM01MD5     SOURCE     PROM
              FLM01LD4     ARCHDEF    INCLD
              FLM01LD3     ARCHDEF    INCLD
FLM01MD6      FLM01MD6     SOURCE     SINC
              FLM01MD6     SOURCE     PROM
              FLM01LD4     ARCHDEF    INCLD
              FLM01LD3     ARCHDEF    INCLD

TOTAL MEMBERS PROCESSED FOR TYPE  =  22
================================================================================
*                                                                              *
*                  CROSS REFERENCE FOR TYPE:     LMAP                        *
*                                                                              *
================================================================================

MEMBER    REF. ARCH. MEM.  TYPE       KEYWORD   INCLUDE-SET
------    ---------------  ----       -------   -----------

FLM01LD3      FLM01LD3     ARCHDEF    LMAP
FLM01LD4      FLM01LD4     ARCHDEF    LMAP

TOTAL MEMBERS PROCESSED FOR TYPE  =  2

================================================================================
*                                                                              *
*                  CROSS REFERENCE FOR TYPE:     LOAD                        *
*                                                                              *
================================================================================

MEMBER    REF. ARCH. MEM.  TYPE       KEYWORD   INCLUDE-SET
--------  ---------------  --------   --------  -----------

FLM01LD3      FLM01LD3     ARCHDEF    LOAD
FLM01LD4      FLM01LD4     ARCHDEF    LOAD

TOTAL MEMBERS PROCESSED FOR TYPE  =  2

================================================================================
*                                                                              *
*                  CROSS REFERENCE FOR TYPE:     ARCHDEF                     *
*                                                                              *
================================================================================

MEMBER    REF. ARCH. MEM.  TYPE       KEYWORD   INCLUDE-SET
------    ---------------  ----       -------   -----------

FLM01ARH      FLM01LD4     ARCHDEF    COPY
              FLM01LD3     ARCHDEF    COPY
FLM01LD3      FLM01LD3     ARCHDEF    PROM
              FLM01SB2     ARCHDEF    INCL
FLM01LD4      FLM01LD4     ARCHDEF    PROM
              FLM01SB2     ARCHDEF    INCL
FLM01SB2      FLM01SB2     ARCHDEF    PROM

TOTAL MEMBERS PROCESSED FOR TYPE  =  7
```

Figure 71. Architecture report with cutoff of NONE (Part 3 of 3)

```
********************************************************************************
********************************************************************************
**                                                                            **
**            SOFTWARE CONFIGURATION AND LIBRARY MANAGER (SCLM)                **
**                                                                            **
**                             ARCHITECTURE REPORT                            **
**                                                                            **
**                            2000/01/06   00:02:30                           **
**                                                                            **
**                                                                            **
**                                                                            **
**                              PROJECT:   PROJ1                              **
**                              GROUP:     DEV1                               **
**                              TYPE:      ARCHDEF                            **
**                              MEMBER:    FLM01SB2                           **
**                              CUTOFF:    LINK EDIT CONTROL                  **
**                                                                            **
**                                                                            **
********************************************************************************
********************************************************************************

================================================================================
*                                                                              *
*                              ARCHITECTURE REPORT                             *
*                                                                              *
*  H = HIGH LEVEL        C = COMPILATION CONTROL  T = TOP SOURCE E = ERROR      *
*  L = LINKEDIT CONTROL  G = GENERIC              I = INCLUDED   D = DEFAULT    *
*                                                                              *
================================================================================

CODE:  H   MEMBER:  FLM01SB2

----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+---

H FLM01SB2 ARCHDEF
L  FLM01LD4 ARCHDEF
L  FLM01LD3 ARCHDEF

NUMBER OF HIGH LEVEL MEMBERS PROCESSED           =    1
NUMBER OF LINK EDIT CONTROL MEMBERS PROCESSED    =    2
NUMBER OF ERROR MEMBERS FOUND                    =    0

================================================================================
*                                                                              *
*                  CROSS REFERENCE FOR TYPE:     SOURCE                        *
*                                                                              *
================================================================================

MEMBER    REF. ARCH. MEM.  TYPE       KEYWORD   INCLUDE-SET
------    ---------------  ----       -------   -----------

FLM01MD3      FLM01LD3     ARCHDEF    INCLD
FLM01MD4      FLM01LD4     ARCHDEF    INCLD
FLM01MD5      FLM01LD4     ARCHDEF    INCLD
              FLM01LD3     ARCHDEF    INCLD
FLM01MD6      FLM01LD4     ARCHDEF    INCLD
              FLM01LD3     ARCHDEF    INCLD

TOTAL MEMBERS PROCESSED FOR TYPE  =  6
```

Figure 72. Architecture report with cutoff of LEC (Part 1 of 2)

```
================================================================================
*                                                                              *
*                  CROSS REFERENCE FOR TYPE:     LMAP                        *
*                                                                              *
================================================================================

MEMBER    REF. ARCH. MEM.  TYPE       KEYWORD   INCLUDE-SET
------    ---------------  ----       -------   -----------

FLM01LD3      FLM01LD3     ARCHDEF    LMAP
FLM01LD4      FLM01LD4     ARCHDEF    LMAP

TOTAL MEMBERS PROCESSED FOR TYPE  =  2

================================================================================
*                                                                              *
*                  CROSS REFERENCE FOR TYPE:     LOAD                        *
*                                                                              *
================================================================================

MEMBER    REF. ARCH. MEM.  TYPE       KEYWORD   INCLUDE-SET
------    ---------------  ----       -------   -----------

FLM01LD3      FLM01LD3     ARCHDEF    LOAD
FLM01LD4      FLM01LD4     ARCHDEF    LOAD

TOTAL MEMBERS PROCESSED FOR TYPE  =  2

================================================================================
*                                                                              *
*                  CROSS REFERENCE FOR TYPE:     ARCHDEF                     *
*                                                                              *
================================================================================

MEMBER    REF. ARCH. MEM.  TYPE       KEYWORD   INCLUDE-SET
------    ---------------  ----       -------   -----------

FLM01ARH      FLM01LD4     ARCHDEF    COPY
              FLM01LD3     ARCHDEF    COPY
FLM01LD3      FLM01LD3     ARCHDEF    PROM
              FLM01SB2     ARCHDEF    INCL
FLM01LD4      FLM01LD4     ARCHDEF    PROM
              FLM01SB2     ARCHDEF    INCL
FLM01SB2      FLM01SB2     ARCHDEF    PROM

TOTAL MEMBERS PROCESSED FOR TYPE  =  7
```

Figure 72. Architecture report with cutoff of LEC (Part 2 of 2)

Export Utility

The export utility writes accounting and cross-reference data to stand-alone and portable accounting and cross-reference databases that contain only those records associated with a specified group. The export utility does not change any data currently residing in the specified group. The output from the export utility is used as input to the import utility.

With the export utility, you can capture SCLM accounting information associated with a specified group. Use the export utility when you want to create a consistent set of data to archive or transport. You can specify that the exported accounting information be purged from an existing export VSAM data set.

Export *only* works on accounting information. Data in project partitioned data sets is *not* exported.

Before using the export utility, verify that the project manager has completed all the steps required to perform the export setup task. Specifically, export data sets must be defined and allocated for the group in the project from which the data is exported.

Figure 73 shows the panel that appears when you select Option 6, Export, from the Utilities panel.

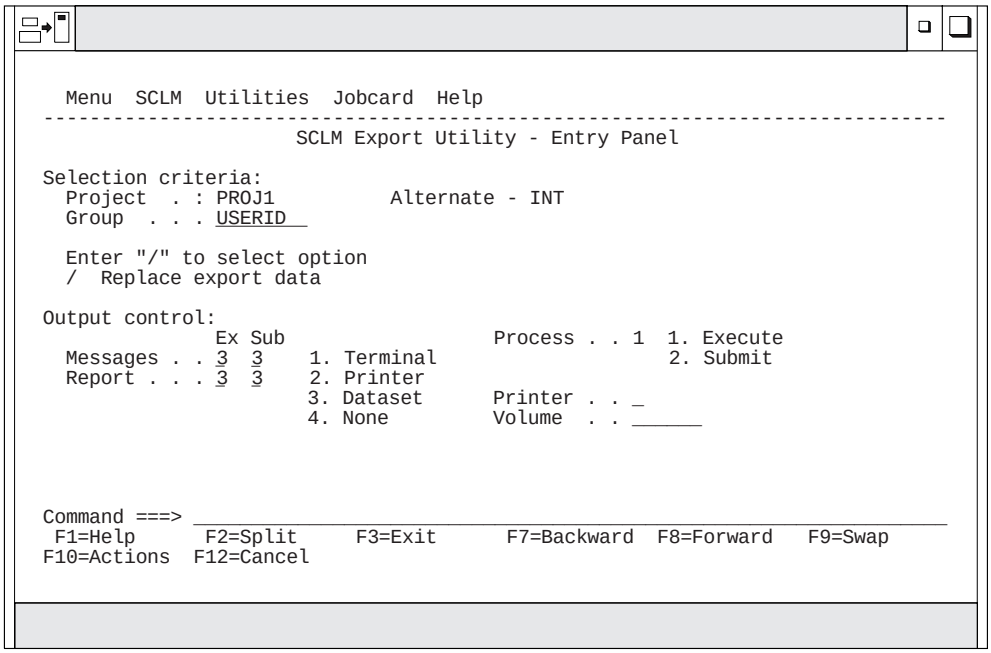


Figure 73. SCLM Export Utility (FLMDXE#P)

To export an SCLM group, enter information for each field. The fields for the Export Utility - Entry panel are:

Project	The project that you specified on the SCLM Main Menu. An **Alternate** field also appears if you specified an alternate project definition.
Group	The group from which you are exporting data.
Replace export data	Specify whether to replace the export accounting and cross-reference data in the export data sets with data from this export. If you do not select this field and the export data sets contain data, the data is not replaced, the export is not performed, and an error message is issued. Export does *not* purge data from the project hierarchy primary accounting and cross-reference data sets.
Output control	Specify the destination for messages and reports when they are executed (Ex) or submitted (Sub) by entering the corresponding destination number.

Process	You can call the processing part of the export utility from the interactive or batch environment by selecting Execute or Submit, respectively. If you request batch processing by selecting Submit, you must specify the job statement information that is used in the JCL generated for batch processing. For information on using a unique jobname on the jobcard in batch processing, see "Batch Processing" on page 230.
Printer	Specify the printer output class
Volume	Specify the volume on which SCLM should save data sets

Export Report Example

Figure 74 on page 199 shows a sample export report.

The report contains a header indicating that it is an Export Report, which project definition and group are being exported, and the data set names of the VSAM files that contain the exported information. The header is followed by three sections: accounting records, build map records, and intermediate records. The report always contains a section for each type even if no records of that type were processed.

The **Verify Status** field contains the value PASSED unless one of the following is true:

- The authorization code change field is non-blank for the record
- The accounting type is INITIAL
- The record could not be read

The **Completion Status** field contains the value PASSED if the record was exported; otherwise, it contains the value FAILED, which means there was some error writing the record to the export database. Completion Status should always contain the value NOT ATTEMPTED if the **Verify Status** field contains the value FAILED, because SCLM does not attempt to export a record if the record did not pass verification.

If the export cross-reference data set is defined for the project definition, the cross-reference records are also exported; but the export report does not include them. If the export cross-reference data set is not defined for the project definition, but the group being exported contains cross-reference records, the Verify Status is set to FAILED and the Completion Status is set to NOT ATTEMPTED. No intermediate records are processed.

```
**************************************************************************
**************************************************************************
**
**
**             SOFTWARE CONFIGURATION AND LIBRARY MANAGER (SCLM)
**
**                                EXPORT  REPORT
**
**                           2000/10/18   10:58:27
**
**                           PROJECT:      PROJ1
**                           ALTERNATE:    PROJ1
**                           GROUP:        USER
**
**
**    EXPORT ACCOUNTING FILE: PROJ1.EXPORT.ACCOUNT.DATABASE
**    EXPORT CROSSREF FILE:   PROJ1.EXPORT.CROSSREF.DATABASE
**************************************************************************
**************************************************************************

  ACCOUNTING RECORDS:                                                PAGE: 1
                            VERIFY     COMPLETION
    TYPE        MEMBER      STATUS       STATUS
  --------     --------     ------    -------------
  SOURCE      A            PASSED    PASSED
  SOURCE      ADABL1       PASSED    PASSED
  SOURCE      ADABL2       PASSED    PASSED
  SOURCE      ADABL3       PASSED    PASSED
  SOURCE      ADABL4       PASSED    PASSED
  SOURCE      ADAMAIN      PASSED    PASSED
  SOURCE      ADASPEC      PASSED    PASSED
  SOURCE      ASPEC        PASSED    PASSED
  SOURCE      B            PASSED    PASSED
  SOURCE      BPRIME       PASSED    PASSED
  SOURCE      BSPEC1       PASSED    PASSED
  SOURCE      BSPEC2       PASSED    PASSED
  SOURCE      BSPEC3       PASSED    PASSED
  SOURCE      BSPEC4       PASSED    PASSED
  SOURCE      BSPEC5       PASSED    PASSED
  SOURCE      BSPEC6       PASSED    PASSED
  SOURCE      B1           PASSED    PASSED
  SOURCE      B2           PASSED    PASSED
  SOURCE      C            PASSED    PASSED
  SOURCE      CSPEC        PASSED    PASSED
  SOURCE      D            PASSED    PASSED
  SOURCE      DSPEC        PASSED    PASSED
  SOURCE      ESPEC        PASSED    PASSED
  SOURCE      LONGCU       PASSED    PASSED
  SOURCE      MAINBIND     PASSED    PASSED
  SOURCE      MH1          PASSED    PASSED
  SOURCE      XSPEC        FAILED    NOT ATTEMPTED
```

Figure 74. Export Report (Part 1 of 2)

```
BUILD MAP RECORDS:                                                PAGE: 2

                          VERIFY     COMPLETION
  TYPE        MEMBER      STATUS       STATUS
--------    --------     ------    -------------
SOURCE      ADAMAIN      PASSED    PASSED
SOURCE      ADASPEC      PASSED    PASSED
SOURCE      ASPEC        PASSED    PASSED
SOURCE      BSPEC1       PASSED    PASSED
SOURCE      BSPEC2       PASSED    PASSED
SOURCE      BSPEC3       PASSED    PASSED
SOURCE      BSPEC4       PASSED    PASSED
SOURCE      BSPEC5       PASSED    PASSED
SOURCE      BSPEC6       PASSED    PASSED
SOURCE      CSPEC        PASSED    PASSED
SOURCE      DSPEC        PASSED    PASSED
SOURCE      ESPEC        PASSED    PASSED
```

Figure 74. Export Report (Part 2 of 2)

Import Utility

The import utility reintroduces the exported SCLM accounting information into the current project after verifying that this data corresponds to the current contents of the SCLM-controlled data sets.

Before using the import utility, verify that the project manager has completed all the steps required to perform the import setup task. Specifically, a copy of the project database from which the items were exported must exist. This means that the PDS members must have been copied. Export VSAM data sets must be defined and allocated for the group in the project into which the data will be imported.

Like the SCLM editor, the import utility verifies authorization codes and prohibits simultaneous updates of members. The group specified to receive the import must be a development group. The import utility also ensures that all the software components to be imported are available and have accounting information. Finally, the import utility verifies that each software component is either new or directly based on the version that exists in the higher group.

The export database is purged after the import is successfully completed.

Figure 75 on page 201 shows the panel that appears when you select Option 7, Import, from the Utilities panel:

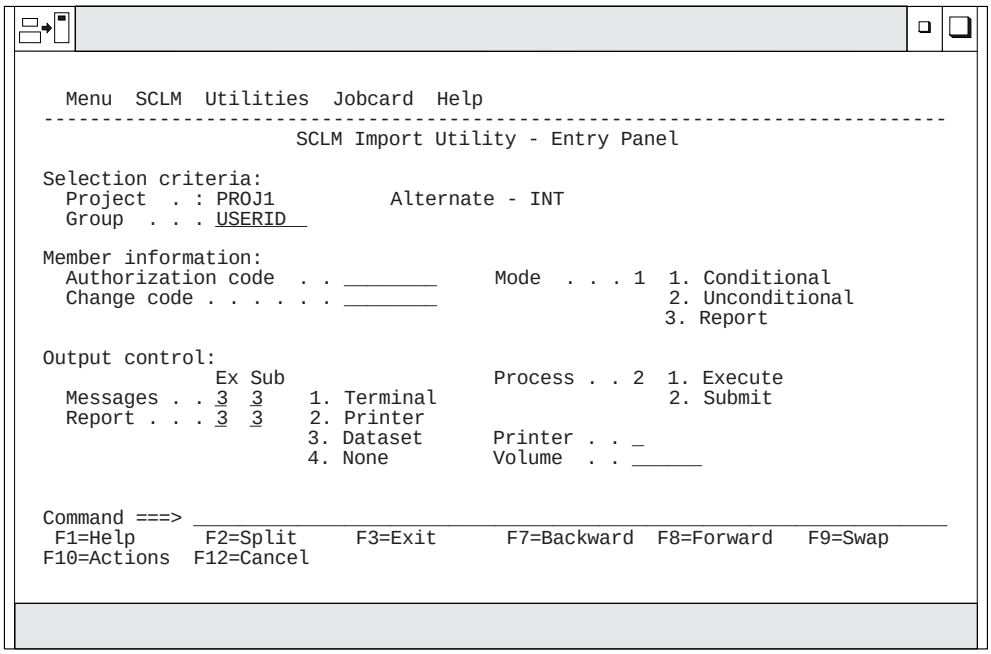


Figure 75. SCLM Import Utility (FLMDXI#P)

To import an SCLM group, enter information in each field. The fields for the Import Utility - Entry panel are:

Project	The project that you specified on the SCLM Main Menu. An **Alternate** field also appears if you specified an alternate project definition.
Group	The development group into which the import is to occur. This group can be any development group defined in the project definition.
Authorization code	The authorization code to be used for all the suitable members to be imported. This field defaults to the authorization code of each member at the time the member is exported. If the authorization code assigned to a member is not in the group being accessed, SCLM does not process the member. Authorization codes cannot contain commas.
Change code	Optionally specify a change code to be added to the change code list of each imported member. Change codes cannot contain commas. If you do not specify a change code, SCLM uses the change code at the time the member is exported.
Mode	Select one of the following: **Conditional** To stop the import process if there is a verification failure. **Unconditional** To bypass importation of only those elements that would introduce problems with project integrity. **Report** To perform verification and report generation processing only.

Output control	Specify the destination for messages and report when they are executed (Ex) or submitted (Sub), by entering the corresponding destination number: 1 for Terminal, 2 for Printer, 3 for Dataset, or 4 for None.
Process	You can call the processing part of the Import Utility from the interactive or batch environment by selecting Execute or Submit, respectively. If you request batch processing by selecting Submit, you must specify the job statement information which is used in the JCL generated for batch processing. For information on using a unique jobname on the jobcard in batch processing, see "Batch Processing" on page 230.
Printer	Specify the printer output class.
Volume	Specify the volume on which SCLM should save data sets.

Import Report Example

Figure 76 on page 203 is a sample import report.

The report contains a header indicating that it is an Import Report, which project definition and group are being imported into, and the data set neams of the VSAM files containing the information that is being imported. The header is followed by three sections: accounting records, build map records, and intermediate records. The report always contains a section for each type even if no records of that type were processed.

The **Verify Status** field contains the value FAILED if any of the verification steps failed for the member; otherwise, it contains the value PASSED.

The **Completion Status** field contains the value PASSED if the record was actually imported; it contains the value FAILED if the import was attempted for a member, but failed; it contains the value NOT ATTEMPTED if the **Verify Status** field contains the value FAILED because no import of a record is attempted if the record did not pass verification. Certain verification steps will pass only for an Unconditional import; these cases result in a **Verify Status** of WARNING and the **Completion Status** for such a member depends on the mode of the import.

If an accounting record has cross-reference records and the accounting record imports successfully, its cross-reference records are also imported. The import report does not include cross-reference records.

```
************************************************************************
************************************************************************
**                                                                    **
**                                                                    **
**             SOFTWARE CONFIGURATION AND LIBRARY MANAGER (SCLM)       **
**                                                                    **
**                                IMPORT  REPORT                      **
**                                                                    **
**                           2000/10/23   06:12:47                    **
**                                                                    **
**                           PROJECT:      PROJ1                      **
**                           ALTERNATE:    PROJ1                      **
**                           GROUP:        USER1                      **
**                           AUTH CODE:                               **
**                           CHANGE CODE:                             **
**                           MODE:         UNCONDITIONAL              **
**                                                                    **
**    EXPORT ACCOUNTING FILE: PROJ1.USER.EXPORT.ACCOUNT               **
**    EXPORT CROSSREF FILE:   PROJ1.USER.EXPORT.CROSSREF              **
************************************************************************

 ACCOUNTING RECORDS:                                          PAGE: 1
                              VERIFY     COMPLETION
     TYPE        MEMBER       STATUS       STATUS
    --------     --------     ------    -------------
    ARCHDEF      ADAASM       PASSED    PASSED
    ARCHDEF      ADALEC2      PASSED    PASSED
    ARCHDEF      ADAPL1       PASSED    PASSED
    ARCHDEF      ADAPSC       PASSED    PASSED
    ARCHDEF      ADASCR       PASSED    PASSED
    ARCHDEF      ALL          PASSED    PASSED
    ARCHDEF      ALLADA       PASSED    PASSED
    ARCHDEF      ALLBAD       PASSED    PASSED
    ARCHDEF      ASMTEXT      PASSED    PASSED
    ARCHDEF      B3CC1        PASSED    PASSED
    ARCHDEF      CIRCULAR     PASSED    PASSED
    ARCHDEF      HADACC       PASSED    PASSED
    ARCHDEF      HADALEC      PASSED    PASSED
    ARCHDEF      HARDADA      PASSED    PASSED
    ARCHDEF      IVADACC      PASSED    PASSED
    ARCHDEF      IVADALEC     PASSED    PASSED
    ARCHDEF      NONADA       PASSED    PASSED
    ARCHDEF      PASCC        PASSED    PASSED
    ARCHDEF      PASCC1       PASSED    PASSED
    ARCHDEF      PASCC2       PASSED    PASSED
    ARCHDEF      PASCERR      PASSED    PASSED
    ARCHDEF      PAS1LEC      PASSED    PASSED
    ARCHDEF      PAS2LEC      PASSED    PASSED
    ARCHDEF      PAS3LEC      PASSED    PASSED
    ARCHDEF      PL1CC1       PASSED    PASSED
    ARCHDEF      PL1CC2       PASSED    PASSED
    ARCHDEF      PL1LEC       PASSED    PASSED
    ARCHDEF      PL1LEC2      PASSED    PASSED
    ARCHDEF      P1           PASSED    PASSED
    ARCHDEF      P2           PASSED    PASSED
    ARCHDEF      P3           PASSED    PASSED
    ARCHDEF      P4           PASSED    PASSED
```

Figure 76. Import Report (Part 1 of 3)

```
ARCHDEF      SAMEHL       PASSED    PASSED
ARCHDEF      SCRIPTHL     PASSED    PASSED
ARCHDEF      SEGLIMIT     PASSED    PASSED
ARCHDEF      SINCALOT     PASSED    PASSED
ARCHDEF      TDSGL1       PASSED    PASSED
ARCHDEF      TDSGL2       WARNING   PASSED
ARCHDEF      TDSHL        PASSED    PASSED
LINKLIST     ADAMAIN      PASSED    PASSED
LISTING      ASM1         PASSED    PASSED
LISTING      ASM2         PASSED    PASSED
LISTING      SCRIPTHL     PASSED    PASSED
LMAP         ADAMAIN      PASSED    PASSED
LMAP         PASLIST1     PASSED    PASSED
LOAD         ADAMAIN      PASSED    PASSED
LOAD         PASLIST1     PASSED    PASSED
SOURCE       A            PASSED    PASSED
SOURCE       ADABL1       PASSED    PASSED
SOURCE       ADABL2       PASSED    PASSED
SOURCE       ADABL3       PASSED    PASSED
SOURCE       ADABL4       PASSED    PASSED
SOURCE       B            PASSED    PASSED
SOURCE       BPRIME       PASSED    PASSED
SOURCE       BSPEC6       PASSED    PASSED
SOURCE       B1           PASSED    PASSED
SOURCE       B2           PASSED    PASSED
SOURCE       C            PASSED    PASSED
SOURCE       D            PASSED    PASSED
SOURCE       LONGCU       PASSED    PASSED
SOURCE       MH1          PASSED    PASSED
MVS1LIST     ADAMAIN      FAILED    NOT ATTEMPTED
MVS1LIST     ASPEC        FAILED    NOT ATTEMPTED
MVS1LIST     BSPEC6       PASSED    PASSED
MVS24DA      BB           PASSED    PASSED
MVS24DA      MHBODY       PASSED    PASSED
MVS24DA      MHS1         PASSED    PASSED
MVS2LIST     ABODY        PASSED    PASSED
MVS2LIST     ASUB1        PASSED    PASSED
MVS2LIST     ASUB2        PASSED    PASSED

ACCOUNTING RECORDS:                                          PAGE: 2
                                  VERIFY     COMPLETION
  TYPE         MEMBER             STATUS       STATUS
--------     --------           ------     -------------
MVS2LIST     ASUB3        PASSED    PASSED
OBJ          ADAMAIN      PASSED    PASSED
OBJ          ASM1         PASSED    PASSED
OBJ          ASM2         PASSED    PASSED
OBJ          PASCOBJ      PASSED    PASSED
SOURCE       ASM1         PASSED    PASSED
SOURCE       ASM2         PASSED    PASSED
SOURCE       COBOL        PASSED    PASSED
SOURCE       COMPA        PASSED    PASSED
SOURCE       COMPB        PASSED    PASSED
SOURCE       COMPC        PASSED    PASSED
SOURCE       FIBCOP       PASSED    PASSED
SOURCE       FIBO         PASSED    PASSED
SOURCE       FORTRAN      PASSED    PASSED
SOURCE       MSG1         PASSED    PASSED
SOURCE       MSG2         PASSED    PASSED
SOURCE       PANEL1       PASSED    PASSED
SOURCE       PANEL2       PASSED    PASSED
SOURCE       PASCPGM      PASSED    PASSED
```

Figure 76. Import Report (Part 2 of 3)

```
   SOURCE       PL1INCL1      PASSED    PASSED
   SOURCE       PL1INCL2      PASSED    PASSED
   SOURCE       PL1INCL3      PASSED    PASSED
   SOURCE       PL1MAIN       FAILED    NOT ATTEMPTED
   SOURCE       PL2MAIN       PASSED    PASSED
   SOURCE       PL3MAIN       PASSED    PASSED
   SOURCE       SCRIPTHL      PASSED    PASSED
   SOURCE       SCRIPT1       PASSED    PASSED
   SOURCE       SCRIPT1A      PASSED    PASSED
   SOURCE       SCRIPT1B      PASSED    PASSED
   SOURCE       SCRIPT2       PASSED    PASSED
   SOURCE       SCRIPT2A      PASSED    PASSED

BUILD MAP RECORDS:                                                PAGE: 3
                                VERIFY     COMPLETION
     TYPE        MEMBER         STATUS       STATUS
   --------     --------       ------    -------------
   ARCHDEF      ADACC          PASSED    PASSED
   ARCHDEF      ADALEC         PASSED    PASSED
   ARCHDEF      ASMTEXT        PASSED    PASSED
   ARCHDEF      PASCC          PASSED    PASSED
   ARCHDEF      PAS1LEC        PASSED    PASSED
   ARCHDEF      SCRIPTHL       PASSED    PASSED
   SOURCE       ADAMAIN        PASSED    PASSED
   SOURCE       ASPEC          PASSED    PASSED
   SOURCE       BSPEC6         PASSED    PASSED
   MVS24DA      ABODY          PASSED    PASSED
   MVS24DA      ASUB1          PASSED    PASSED
   MVS24DA      ASUB2          PASSED    PASSED
   MVS24DA      ASUB3          PASSED    PASSED
   SOURCE       ASM1           PASSED    PASSED
   SOURCE       ASM2           PASSED    PASSED
   SOURCE       SCRIPTHL       PASSED    PASSED
```

Figure 76. Import Report (Part 3 of 3)

Audit and Version Utility

The audit and version utility allows you to audit SCLM operations on SCLM controlled members and create versions of editable members. Using the audit and version utility, you can view the audit information for a member, retrieve a version to a sequential data set not controlled by SCLM, to a partitioned data set not controlled by SCLM, or to a SCLM controlled development group. This utility also enables you to delete audit and version information from the database.

The project manager controls the audit and version capabilities through the use of macros within the project definition. Audit information is stored in a VSAM data set, and versions of the SCLM members are stored in one or more partitioned data sets allocated for this use.

Note: The data kept in audit VSAM data sets and the versioning partitioned data sets is for the exclusive use of the audit and version utility. Do *not* edit or alter these data sets without using the audit and version utility or the data may be lost.

Figure 77 on page 206 shows the panel that appears when you select Option 8, Audit and Version, from the SCLM Utilities Panel.

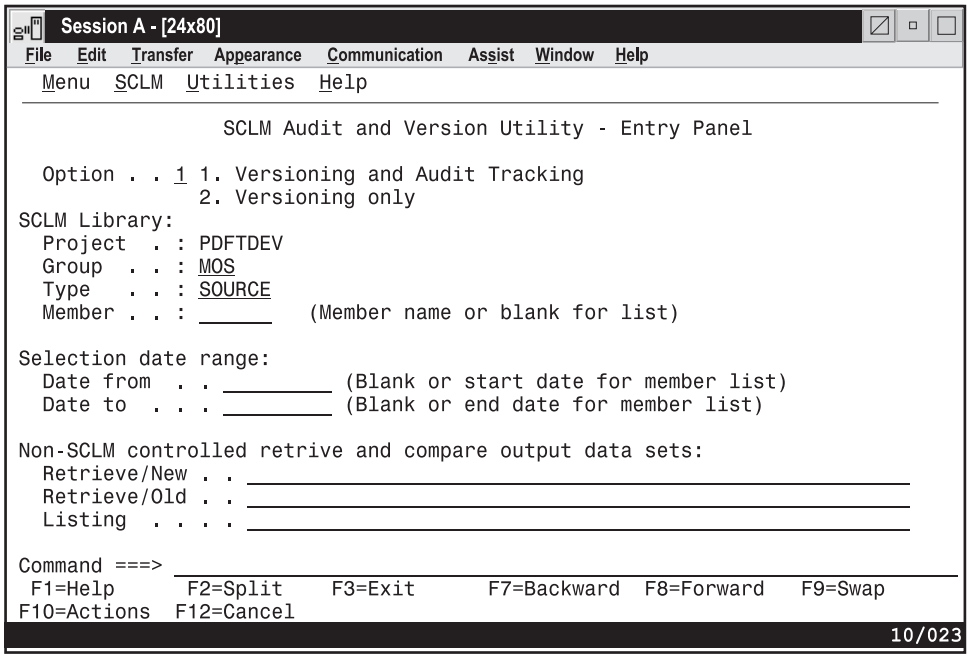


Figure 77. SCLM Audit and Version Utility (FLMVUS#P)

The fields on the SCLM Audit and Version Utility - Entry Panel are:

Project	The project that you specified on the SCLM Main Menu. An **Alternate** field also appears if you specified an alternate project definition.
Group	The group for which you want audit and versioning information. The specified group must have an audit VSAM data set defined in the project definition. It must also be defined on an FLMATVER macro in the project definition.
Type	The type of the member for which you want the version and audit information displayed or retrieved. The type must be defined on an FLMATVER macro in the project definition. A wildcard or '*' is not permitted in this field. This is a required field.
Member	The member for which you are requesting information. If you leave this field and the **Command** field blank, SCLM displays the SCLM Version Selection panel. The **Member** field is optional.
Date from	The starting date of the range of dates to search for the specified member. The date must be in the form YY/MM/DD. If you specify a member and leave this field blank, SCLM searches from the beginning of the file to the TO date. If you specify a member and leave the **Date from** and **Date to** fields blank, all versions of the member appear. SCLM verifies that the date you enter is valid and not greater than today's date. The **Date from** field is optional.

Date to	The ending date of the range of dates to search for the specified member. The date must be in the form YY/MM/DD. If you specify a member and leave this field blank, SCLM uses the current date as the end date for the search. If you leave the **Date from** and **Date to** fields blank, all versions of the member appear. SCLM verifies that the date you enter is valid and greater than or equal to the **Date from** value. If you specify a future date, this field defaults to the current date. The **Date to** field is optional.
Data set names	**Retrieve/New** The name of the data set in which the version of the member selected by the **R** command is stored. A value entered in this field takes precedence over the *To Group* and *To Type* fields. This data set can be sequential or partitioned. If you specify a partitioned data set, you cannot specify a member name. This data set is also treated by SCLM as being outside of SCLM control. **Retrieve/Old** The name of the data set in which the version of the member selected by the **C** command is stored. On exit from this panel, the members in both retrieval data sets are compared by SuperC. **Listing** The name of the data set to which the compare output is written. SuperC creates this data set for you, if necessary. A member name is required for a partitioned data set.
To Group	The SCLM development group that you want to retrieve the versioned member to. The group must be a development group that is defined in the project definition.
To Type	The type of the member that you want to retrieve the versioned member into. The type must be defined in the project definition. A wildcard or asterisk (*) is not permitted in this field.
Authorization Code	A valid authorization code for the retrieve group that you want to associate with the member.

SCLM Version Selection

You can retrieve a version of a member, view the accounting information for that version of the member, compare versions of retrieved members, or delete that version and its associated accounting information by entering selections on the SCLM Version Selection panel. To display the SCLM Version Selection panel, do the following on the SCLM Audit and Version Utility Entry panel:

1. Leave the **Command** field blank or enter `V` in the **Command** field.
2. Enter the group and type information in the appropriate fields.
3. Press Enter.

Use the SCROLL commands or the LOCATE command to scroll the list.

To restrict the member list displayed, you can enter values in any or all of the following fields:

- Member (enter full member name or leave blank)
- Date From (enter valid date)
- Date To (enter valid date)

Figure 78 shows the SCLM - Version Selection panel appearing when you follow the preceding steps:

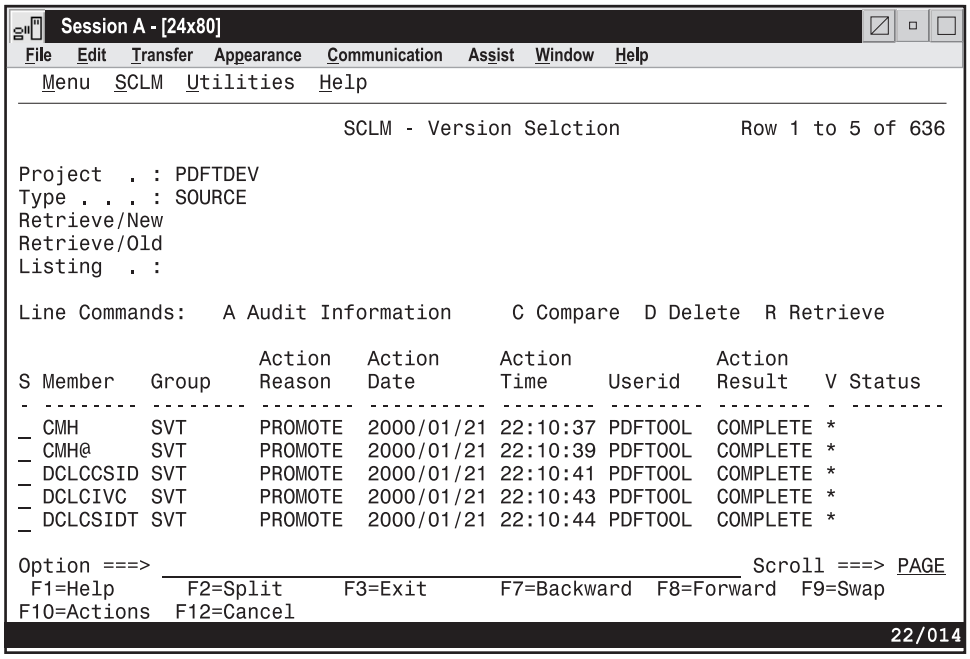


Figure 78. SCLM Version Selection Panel (FLMVSL#P)

The fields for the Version Selection panel shown in Figure 78 are:

Member	The names of the members matching the selection criteria on the SCLM Audit and Version Utility - Entry panel that have audit and version information.
Group	The name of the group you specified on the SCLM Audit and Version Utility - Entry panel.
Action Reason	The action that was performed against the specified member. Valid values include: • BUILD • BLDDEL • DELETE • EXT LIB • FREE • IMPORT • LOCK • PROMOTE • STORE • UNLOCK • UPTATHCD (update authorization code) • UPTCHGCD (update change code) • UPTUENTY (update user entry)
Action Date	The date the action listed in the **Action Reason** field occurred.
Action Time	The time the action listed in the **Action Reason** field occurred.
Userid	The user ID of the person who performed the action.

Action Result	The result of the action performed on the member. **ATTEMPT** The action is in progress or was not completed (ABEND, for example). **COMPLETE** The action completed successfully. **FAILED** The version action failed. You can get a message ID from the audit record display that can be used to determine the cause for failure. You must look at the Audit Record Display Panel to know which field to reference.
Ver	Indicates (using an asterisk) whether a version of the member exists.
Status	Indicates the status of the line command. Possible values are: • *SELECT • *DELETED • *FAILED • *ERROR • RETRVOLD • RETRVNEW

To the left of each member listed is a space for entering a line command to do the following:

A Display the audit information for the member.

When you enter the A line command beside a member name, the Audit / Version Record panel appears, as shown in Figure 79 on page 211, giving you the information recorded for that member.

C Retrieve the member and store it in the Retrieve/Old data set you specified on the *SCLM Audit and Version Utility — Entry* panel.

When you enter the **C** command beside a member, SCLM retrieves the member in preparation for SuperC comparing the contents of the Retrieve/New and Retrieve/Old data sets. The comparison begins when you exit the *SCLM Version* selection panel, at which time you are prompted for compare options.

For implications of multiple member retrieval and data set characteristics, see the discussion of the **R** command below.

D Delete the audit record in the VSAM audit data set and delete the versioned member in the partitioned data set.

When you enter the D line command beside a member name, SCLM deletes the audit record and the corresponding versioned member, if one exists. A message appears, indicating that the operation completed successfully.

R Retrieve the member and store it in the Retrieve/New data set you specified on the SCLM Audit and Version Utility - Entry panel.

When you enter the R line command beside a member, SCLM retrieves the member. When you retrieve more than one member into a sequential data set, each member after the first is copied over the previous member. To retrieve

more than one member to a sequential data set, copy the first member to another data set before retrieving a second member. We recommend that you use a partitioned data set if you intend to copy more than one member.

SCLM will not allow you to retrieve a second version of the same member but you may retrieve a version of a different member. To retrieve a second version of the same member you must first return to the SCLM Audit and Version Utility Entry panel and then come back to the SCLM Version Selection panel.

Note: When you retrieve a member into either an SCLM-controlled or non-SCLM-controlled partitioned data set, SCLM does not issue a warning if another member with the same name is already in the data set.

You can enter multiple commands on the panel as long as the commands do not conflict. All requests are handled in succession unless an error occurs. If an error occurs, the selection list indicating the error reappears. You must correct the error before further processing can occur.

Note: When you retrieve the most recent version of a source member into a development group of the hierarchy, the accounting data and ISPF statistics match those of the member that is already in the hierarchy. Therefore, outputs are not produced when the member is built because the outputs that are already in the library are current.

In addition, when the recovered member is promoted to the level where the member resides, the existing member is not overwritten. If the content of the existing member has been corrupted and it is important to replace that member, you must save the member in the hierarchy after it is recovered. You can save the member using SCLM edit, migrate in forced mode, or the SAVE service.

SCLM Audit and Version Record

If you enter 'A' to display the SCLM Audit and Version record, the Audit / Version panel shown in Figure 79 on page 211 appears.

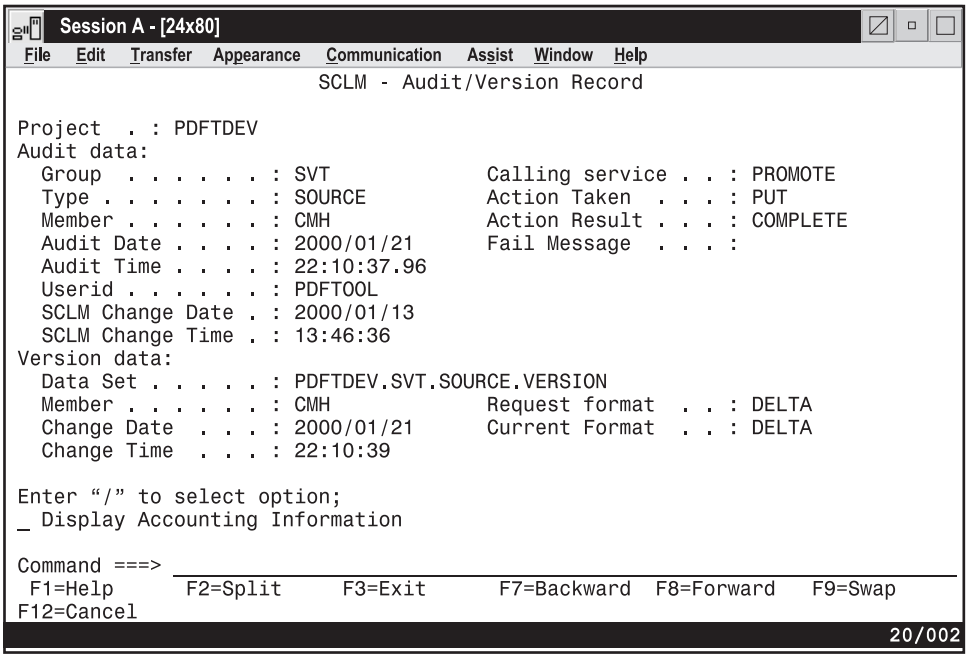


Figure 79. SCLM Audit / Version Record (FLMVBA#P)

The fields for the panel shown in Figure 79 are:

Project	The project that you specified on the SCLM Main Menu. An **Alternate** field also appears if you specified an alternate project definition.
Group	The group for which the accounting information appears.
Type	The type for which the accounting information appears.
Member	The member for which the accounting information appears.
Audit Date	The date the audit was performed.
Audit Time	The time the audit was performed.
Userid	The userid of the person who caused the audit record to be created.
SCLM Change Date	The date the member was last edited.
SCLM Change Time	The time the member was last edited.
Data Set	The name of the PDS where the version data, if any, for this record is stored. This name is always present, whether or not version data exists.
Member	The name of the member in which version data is stored, if this record has version data. This field is blank if there is no version data.
Change Date	The date the versioned member was written.
Change Time	The time the versioned member was written.
Calling Service	The service that SCLM is running at the time; for example, BUILD, PROMOTE, STORE, LOCK, or DELETE.

Action Taken	The function that causes the audit / version to be taken. For example, EDIT causes a SAVE. EDIT is the calling service and SAVE is the action taken. The action could be LOCK, DELETE, MIGRATE, and so on. The calling service and the action taken could be the same. For example, the BUILD service could cause the BUILD action to take a version.
Action Result	Indicates the status of the action taken.
Fail Message	Indicates a failure. This field contains the message number of the failing message.

If the action result is COMPLETED, you can display the related accounting information. Enter S to select this option located at the bottom of the SCLM Audit / Version Record panel. See Figure 51 on page 166 for an example of the Accounting Record panel.

Version Compare

If you enter 'C' to retrieve a version to place in the Retrieve/Old data set, your subsequent exit from the *SCLM Version Selection* panel causes the *Version Compare Options* panel, shown in Figure 80, to appear. Enter the options you want to use to compare the contents of the Retrieve/New and Retrieve/Old data sets. The output of the comparison goes to the **Listing** data set.

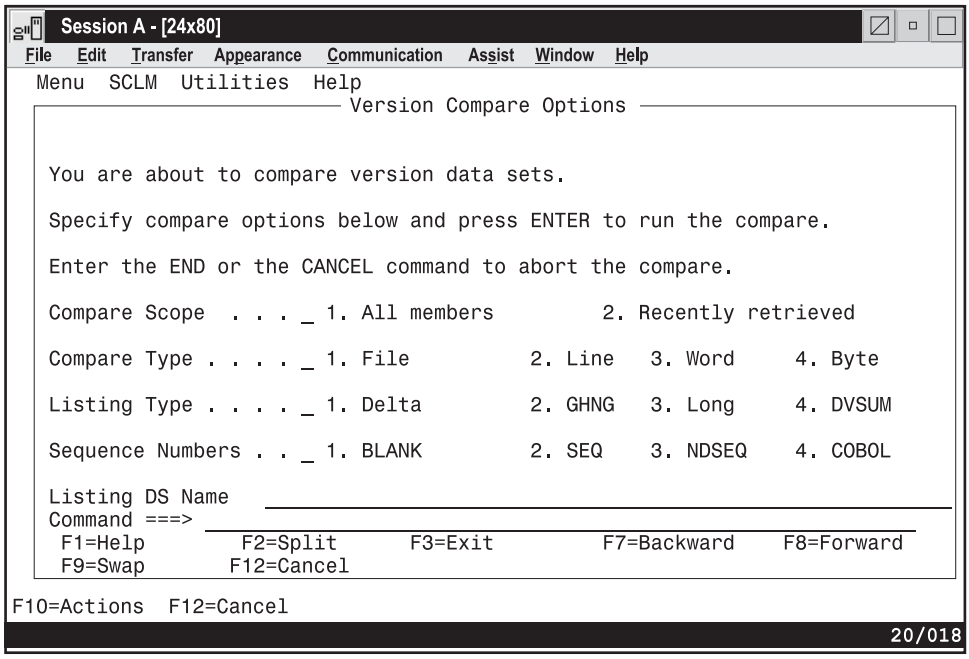


Figure 80. SCLM Version Compare Options Panel (FLMVSC#P)

The fields for the panel shown in Figure 80 are:

Compare Scope	Controls whether **all members** in the Retrieve/New and Retrieve/Old data sets are compared, or only the **recently retrieved** member or members you just now selected through the C command on the *Version Selection* panel you just exited.

Compare Type	Specifies the granularity of the comparison, ranging from entire member to member (**file**) comparison down to single **byte** differences. **Line** compare is useful for source data. **Word** compare is most useful for text data.
Listing Type	Specifies the context scope of the listing report. You can get a listing with summary information only (**OVSUM**), single line differences between files (**Delta**), differences plus or minus the five unchanged lines before and after changed lines (**CHNG**), or a listing that includes ***all*** of the lines in both files (**Long**).
Sequence numbers	Specifies whether sequence numbers in the compared files are to be ignored or treated as data. Choosing **SEQ** means to ignore differences in standard sequence number columns 72 through 80 for FB LRECL 80 members. Choosing **NOSEQ** means to treat ***all*** columns in the files as data. The **COBOL** selection means to ignore differences in columns 1 through 8 of the data. Choosing **Blank** causes SuperC to ignore standard sequence number columns if the data set is FB 80 or VB 255. Otherwise, the comparison processes those columns as data.
Listing DS Name	The data set into which the compare listing is written. You can pre-allocate this data set, or let ISPF create one for you. If this data set is partitioned, you must specify a member name.

Note: More sophisticated comparisons can be done using the ISPF Option 3.13, **SuperC Compare Utility**.

Delete Group Utility

The Delete Group utility lets you delete database components associated with a specified group. You can delete a member or members and all associated SCLM accounting information, including accounting records, build map records, cross-reference records, and intermediate records. You can further specify whether you want everything deleted, only build outputs, only accounting information and build map records, or only build map records. You may also specify that nothing actually be deleted but a deletion report be generated.

The delete group utility does not delete members that have no accounting information.

Figure 81 on page 214 shows the panel that is displayed when you select Option 9, Delete Group, from the Utilities panel.

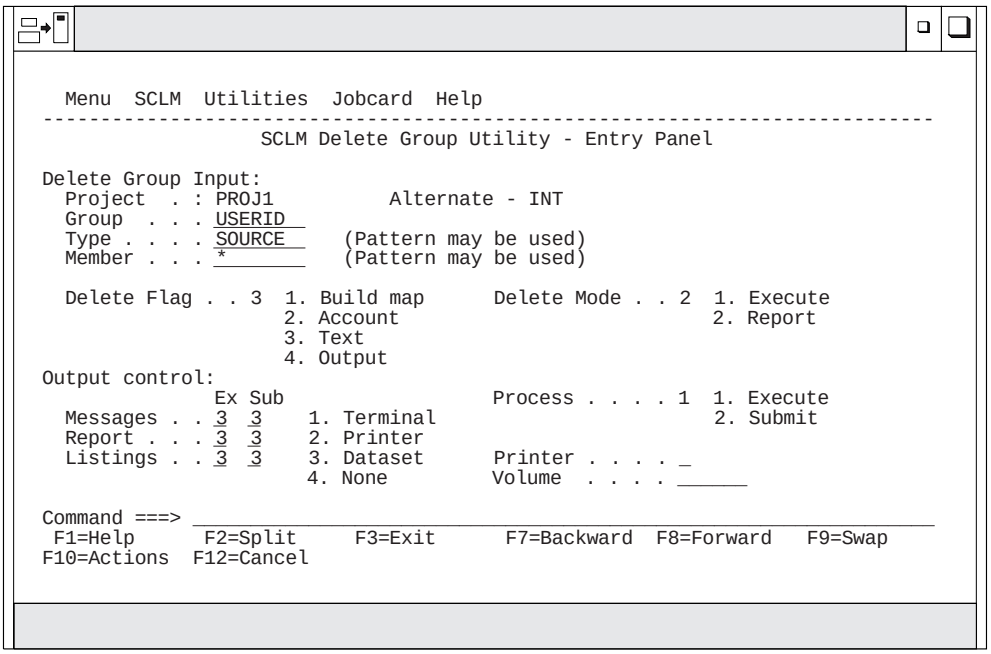


Figure 81. SCLM Delete Group Utility (FLMDDG#P)

To delete information from an SCLM group, you must enter information for each field. The fields for the Delete Group Utility - Entry panel are:

Project	The project specified on the SCLM Main Menu. This field is display only. An **Alternate** field also appears if you specified an alternate project definition.
Group	The group for which information is to be deleted. Delete Group only works on groups defined to the project. This field is required. There are no default values.
Type	The type from which information is to be deleted. You can use patterns for the type you want processed. See "Specifying Selection Criteria" on page 180 for details. Delete Group only works on types defined to the project.
Member	The name or pattern of the members and SCLM information to be deleted. Only members that have accounting information are deleted. You can use patterns for the member name. See "Specifying Selection Criteria" on page 180 for details.

Delete Flag	The indicator of the type of data to be deleted. **Build map** All build map records that match the pattern are deleted. **Account** All accounting records, cross-reference records, intermediate records, and build map records that match the pattern are deleted. The accounting type will not be checked. **Text** All accounting records, cross-reference records, intermediate records, build map records, intermediate code, and text members that match the pattern are deleted. The accounting type will not be checked. **Output** All build map records, intermediate records and code, and all non-editable accounting records, their cross-reference records and associated text members that match the pattern are deleted. Editable accounting records, their cross-reference records or associated text members are not deleted.
Delete Mode	The indicator for the action performed by the delete group. Select one of the following: **Execute** All members that match the selection criteria for the specified Delete Flag are deleted. **Report** No deletion will occur; contents of what would, upon execution, be deleted for the specified selection criteria and Delete Flag are reported. Report is always be the default whenever this panel appears. Even after you execute a delete group, the mode is changed to Report. To delete members, update authority to the hierarchy data sets containing the members is required, even if the Delete Group utility is run in REPORT mode.
Output control	Specify the destination for messages and the report when they are executed (Ex) or submitted (Sub), by entering the corresponding destination number: 1 for Terminal 2 for Printer, 3 for Dataset, or 4 for None. A listing data set will not be allocated when the Delete Mode is Report, even though Dataset is specified for the Listings field.
Process	You can call the processing part of the delete group utility from the interactive or batch environment by selecting Execute or Submit, respectively. If you request batch processing by selecting Submit, you must specify the job statement information which is used in the JCL generated for batch processing. For information on using a unique jobname on the jobcard in batch processing, see “Batch Processing” on page 230.
Printer	Specify the printer output class.
Volume	Specify the volume on which SCLM should save data sets.

Delete Group Report Example

Figure 82 on page 217 shows a sample Delete Group report.

The report contains a header indicating that it is a Delete Report, which project definition and group are specified, the type and member selection criteria, and the delete flag and mode. The header is followed by three sections: members, build maps, and Ada intermediate code. The report always contains all of these sections even if there is no activity to report for a section. Output members are denoted by an asterisk (*) at the beginning of the report line.

The **VERIFY STATUS** field contains the value PASSED unless the delete routine was unable to verify the record for one of the following reasons:
- User has no update authority
- Member has non-blank access key
- Error reading the record

The **COMPLETION STATUS** field contains the value PASSED if the member was actually deleted. The field contains NOT ATTEMPTED if the verification failed or the delete group was run in REPORT MODE only. The field contains FAILED if an error occurred during the execution of the deletion. The field contains WARNING if the text member or intermediate code did not exist. The accounting record is still deleted.

Although cross-reference records are deleted, there is no section explicitly for them in the delete group report. If the accounting record is successfully deleted, its cross-reference records, if any, are also deleted.

The report contains a header indicating that it is a Delete Group report. The header also shows which project definition and group are specified, the type and member selection criteria, and the delete flag and mode.

```
****************************************************************
****************************************************************
**                                                            **
**                                                            **
**       SOFTWARE CONFIGURATION AND LIBRARY MANAGER (SCLM)     **
**                                                            **
**                    DELETE GROUP REPORT                     **
**                                                            **
**                    2000/03/26   13:30:39                   **
**                                                            **
**                    PROJECT:      PROJ1                     **
**                    ALTERNATE:    PROJ1                     **
**                    GROUP:        USER1                     **
**                    TYPE:         *                         **
**                    MEMBER:       *                         **
**                    FLAG:         TEXT                      **
**                    MODE:         REPORT                    **
****************************************************************
****************************************************************

 MEMBERS:                                              PAGE 1
                                  VERIFY   COMPLETION
      GROUP    TYPE     MEMBER    STATUS   STATUS
    --------   ------   -------  ------    ----------
    USER1      SOURCE   ASM1     PASSED    NOT ATTEMPTED
    USER1      SOURCE   ASM2     PASSED    NOT ATTEMPTED
    USER1      SOURCE   PASMAIN  PASSED    NOT ATTEMPTED
   *USER1      LISTING  PASMAIN  PASSED    NOT ATTEMPTED
   *USER1      LMAP     PASMAIN  PASSED    NOT ATTEMPTED
   *USER1      LOAD     PASMAIN  PASSED    NOT ATTEMPTED
   *USER1      OBJ      PASMAIN  PASSED    NOT ATTEMPTED
    USER1      SOURCE   PASCPGM  PASSED    NOT ATTEMPTED
    USER1      SOURCE   PSCINCL1 PASSED    NOT ATTEMPTED
    USER1      SOURCE   PSCINCL2 PASSED    NOT ATTEMPTED
    USER1      SOURCE   PSCINCL3 PASSED    NOT ATTEMPTED
    USER1      SOURCE   SCRIPTHL PASSED    NOT ATTEMPTED
    USER1      SOURCE   SCRIPT1  PASSED    NOT ATTEMPTED

 BUILD MAPS:                                           PAGE: 2
                                  VERIFY   COMPLETION
      GROUP    TYPE     MEMBER    STATUS   STATUS
    --------   ------   -------  ------    ----------
    USER1      SOURCE   PASCMAIN PASSED    NOT ATTEMPTED

 ADA INTERMEDIATE CODE:                                 PAGE:  3
                                             VERIFY   COMPLETION
 GROUP    CU QUAL  CU NAME        CU TYPE   STATUS     STATUS
--------  -------- -------------  -------   ------   -----------
******************** NO RECORDS PROCESSED ********************
```

Figure 82. Delete Group Report

Build (Option 4)

The build processor automatically compiles, links, or deletes output to make build outputs match build inputs. The build function:

- Ensures total project integrity by verifying that all components defined to the architecture being built are present and complete
- Performs necessary translations such as compiles and links
- Conditionally saves translator output in the database
- Generates a build report

Build (Option 4)

Build compiles, links, and integrates software components according to the architecture. For any group in the hierarchy, the build function uses the software components within the hierarchy of that group to update the out-of-date members. Use build to compile and link individual components as well as to integrate the smaller components into larger components.

For each component that it processes, the build function takes one of the following actions:

- Does nothing if the component has not changed since the previous build
- Deletes out-of-date outputs if that will leave the component in an up-to-date state
- Compiles or links changed components.

At the completion of the build, SCLM, when requested, produces a report identifying the members that were generated or deleted by the build function.

You also can specify that a Build Report be generated without actually invoking any translators. The Build Report identifies those components in the hierarchy that would change if translators were to be invoked.

Before build begins processing the member, it tries to open the VSAM accounting and cross-reference data sets for the group where the build is taking place. If you do not have UPDATE authority to the data sets or if there is an error opening one of the data sets, the build will fail. See *ISPF Software Configuration and Library Manager (SCLM) Reference* for more information on the processing done by the build processor.

The panel shown in Figure 83 appears when you select Option 4, Build, from the SCLM Main Menu.

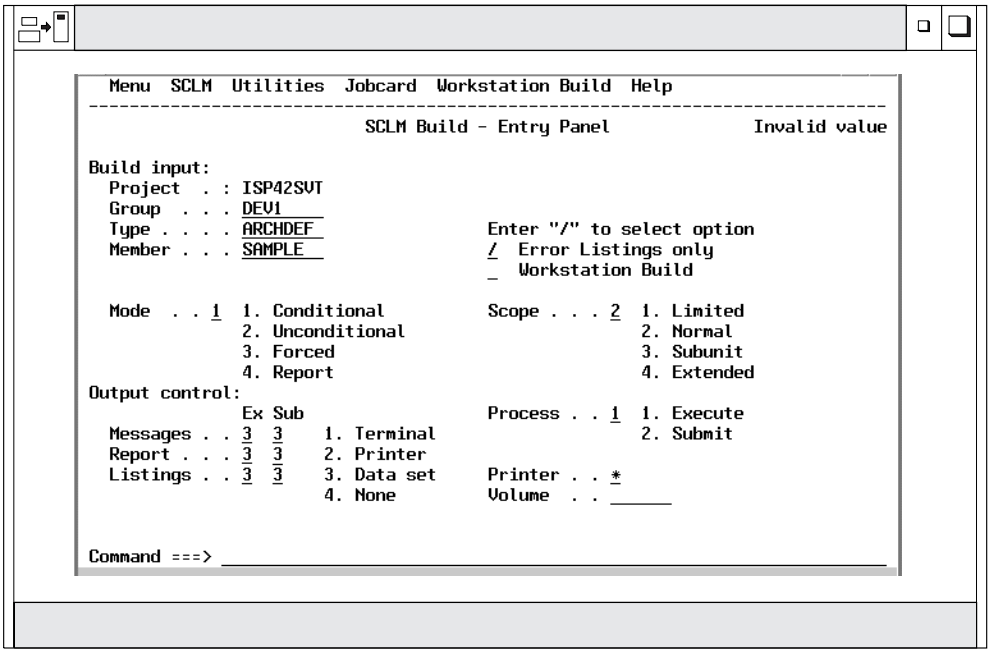


Figure 83. SCLM Build (FLMB#P)

Note: The NRETREIV command key is enabled to work with this option. See "Name Retrieval with the NRETRIEV command" on page 145 for more information.

The fields for the SCLM Build - Entry panel are:

Project	The project that you specified on the SCLM Main Menu. An **Alternate** field also appears if you specified an alternate project.
Group	The group in which the build is to occur.
Type	The type of the member to build.
Member	The name of the member to build.
Scope	You must specify a scope equal to or greater than the scope specified with the SCOPE keyword in the FLMLANGL macro. **Limited** To process those components that the architecture members directly reference. If you use a source member, the build function processes only that member. **Normal** To process the components and members referenced by the specified architecture member. In addition, this scope processes upward dependencies for all Ada-type source members referenced directly by the architecture member and all source members referenced as upward dependencies. **Subunit** To process the components and members processed in normal scope as well as downward dependencies for all Ada-type source members referenced directly by the architecture members. **Extended** To process the components and members processed in normal scope as well as downward dependencies for all source members within the normal scope and the source to all outputs referenced. In addition, extended scope processes any outputs referenced via LINK architecture definition statements or parsed includes. Extended scope also includes anything that Promote verifies that is related to the member built. For example if the architecture definition statement LINK is used to reference a load module, the architecture definition that created the referenced load module is included in the extended scope. Because SCLM uses information from the most recent build map to determine what should be included in extended scope, extended scope may include members that are no longer relevant to the architecture. If you receive error messages about members that are no longer relevant to the architecture definition, try building in normal scope before using extended scope.

Mode

Conditional
To check for unacceptable translator return codes (for example, compiler or linker return codes). Processing stops immediately if build detects any translation errors.

SCLM saves build maps and translator output only for translations that complete successfully. However, the translator listings (if desired) for all components processed, and the build report, are saved and reflect the final results of the build.

Unconditional
To continue processing of all members despite translation errors of other members.

Use this mode when you need to update complete applications or large subapplications. You can also use this mode initially to detect translation errors in several components.

As with the conditional mode, BUILD will stop when verification errors occur and not continue on to execute the BUILD translators. After a successful verification of the members, SCLM will pass control to the BUILD translators, regardless of the return code value from each translator. This will provide information as to the extent of any errors that may have been introduced by changing the members. A conditional BUILD would stop after the first translator return code that exceeds the GOODRC value for the related FLMTRNSL macro.

Build does not attempt a translation unless all of its dependencies that were in scope were completed successfully. For example, a linkedit is not attempted if the compilation of a source member failed.

Forced
To force all requested components to be translated again regardless of the previous status of the modules.

Report
To generate a complete build report without performing an actual build. The report reflects the potential results of an unconditional build.

Output control

Specify the destination for messages, report, and listings when they are executed (Ex) or submitted (Sub), by entering the corresponding destination number: 1 for Terminal, 2 for Printer, 3 for Dataset, or 4 for None.

When executing a build in the foreground, the build listing is browsed if a translation error occurs; otherwise, the build report is browsed. The translator is responsible for providing the build listing.
Note: If no output is specified for **Report**, no build user exit information is produced. That is because SCLM provides the build user exit with information from the build report.

The data sets that are created are not deleted. Specifying a volume that already contains a report, message or listing data set could result in JCL errors when the job is submitted.

Error listings only	The build service allows you to generate a temporary listings file. If you do not select Error listings only, all translator listings are copied to the temporary listings file. If you select it, only those members receiving a translator error are copied to the temporary listings file. An empty file indicates that no errors occurred. The file is temporary in the sense that the contents are not under SCLM control and may be purged by the user.
Workstation Build	Specify whether or not the build will invoke any workstation translators. For a foreground build which invokes a workstation translator, SCLM will verify that an ISPF workstation connection exists before executing the build. For a batch build which invokes a workstation translator, SCLM will verify that the information required to initiate an ISPF workstation connection has been set by a previous build or the workstation build pull-down. If not, SCLM will prompt the user to enter this information before the build job is submitted. If the build does not invoke a workstation translator, do not specify this field.
Process	You can call the processing part of the build utility from the interactive or batch environment by selecting Execute or Submit, respectively. If you request batch processing by selecting Submit, you must specify the job statement information that is used in the JCL generated for batch processing. For information on using a unique jobname on the jobcard in batch processing, see "Batch Processing" on page 230.
Printer	Specify the printer output class.
Volume	Specify the volume on which SCLM should save data sets.

Build Report Example

The build report provides a synopsis of the build. It includes:
- The date and time of the build
- The mode used
- The name of the component that was requested to be built
- The last change date and time of the component
- The project definition used
- The software components that were successfully translated
- The build maps that required regeneration
- The out-of-date software components that caused the regeneration
- The software components and build maps that were deleted from the build group.

This report provides a synopsis of the Build. The title page identifies the date and time of the build, as well as the scope and mode used. It also lists the member you specified on the Build panel and the project definition specified on the SCLM Main Menu.

The report lists the components that were built and saved in the database; that is, those components that passed the compilation or linkage edit phase. It also shows the build maps that required (re)generation, along with a list of software components that build used to determine that (re)generation of the build map was necessary. After the section for items generated, the report contains a section for items deleted. It lists the build outputs that were deleted from the build group. Finally, it lists the build maps that were deleted.

Note: Intermediate information is in the report if it is valid and useful. The following example is an Ada build report, so the sections on Intermediate

Code Generated and Intermediate Code Deleted have been included. These two sections are omitted from the report for builds that do not affect intermediate code.

If you enter REPORT in the **Mode** field, the report indicates what would be rebuilt or deleted if you requested an unconditional build.

Figure 84 shows an example of a build report.

```
********************************************************************************
**                                                                            **
**                                                                            **
**            SOFTWARE CONFIGURATION AND LIBRARY MANAGER (SCLM)                **
**                                                                            **
**                        B U I L D    R E P O R T                            **
**                                                                            **
**                         2000/11/18   08:41:19                              **
**                                                                            **
**                         PROJECT:      SCLM69                               **
**                         GROUP:        USER                                 **
**                         TYPE:         MVS2ADA                              **
**                         MEMBER:       GSPEC                                **
**                         ALTERNATE:    SCLM69                               **
**                         SCOPE:        NORMAL                               **
**                         MODE:         CONDITIONAL                          **
**                                                                            **
**                                                                            **
********************************************************************************


    ******* B U I L D   O U T P U T S   G E N E R A T E D *******  Page 1

MEMBER     TYPE       VERSION      KEYWORD
------     ----       -------      -------
FLM01MD3   OBJ              6      OBJ
FLM01MD5   OBJ              6
FLM01MD6   OBJ              6
FLM01MD3   LIST             6      LIST
FLM01MD5   LIST             6
FLM01MD6   LIST             6
FLM01LD3   LOAD             6      LOAD
FLM01LD3   LMAP             6      LMAP

           ******* B U I L D   M A P S   G E N E R A T E D *******  Page 2

                                  (REASON FOR REBUILD)
MEMBER     TYPE       VERSION      MEMBER     TYPE
------     ----       -------      -------    ----
FLM01LD3   ARCHDEF          3      FLM01MD3   SOURCE
FLM01MD3   SOURCE           6      FLM01MD3   SOURCE
                                   FLM01MD5   SOURCE
                                   FLM01MD6   SOURCE
FLM01MD5   SOURCE           5      FLM01MD5   SOURCE
FLM01MD6   SOURCE           4      FLM01MD6   SOURCE
```

Figure 84. Build Report (Part 1 of 2)

```
    ******* B U I L D   O U T P U T S    D E L E T E D   *******  Page 3

MEMBER     TYPE       VERSION      KEYWORD
------     ----       -------      -------
FLM2M01    OBJ              4      OBJ
FLM2M02    OBJ              4
FLM2M03    OBJ              4
FLM2M01    LIST             4      LIST
FLM2M02    LIST             4
FLM2M03    LIST             4
FLM2LD     LOAD             5      LOAD
FLM2LD     LMAP             5      LMAP

           ******* B U I L D   M A P S    D E L E T E D   *******  Page 4

                                  (REASON FOR DELETE)
MEMBER     TYPE       VERSION      MEMBER     TYPE
------     ----       -------      -------    ----
FLM02LD    ARCHDEF          6      FLM02LD    LOAD
                                   FLM02LD    LMAP
FLM02MD1   SOURCE           6      FLM02MD1   OBJ
                                   FLM02MD1   LIST
FLM02MD2   SOURCE           6      FLM02MD2   OBJ
                                   FLM02MD2   LIST
FLM02MD3   SOURCE           6      FLM02MD3   OBJ
```

Figure 84. Build Report (Part 2 of 2)

Promote (Option 5)

The promote function copies members from any group to the next higher group.

Note: SCLM promote only copies a member over a member at the next level if it has changed. Two members with the same name are considered to be changed if the accounting data and the member statistics are different. If you retrieve the most recent version of a member into the hierarchy, the recovered member at the development group is considered the same as the member residing in the hierarchy. If the member in the hierarchy has been corrupted, but the statistics are still valid, SCLM will not overwrite the existing member during promotion. The promote report indicates that the member was purged but not copied. If you recover the most recent version of a member in order to replace a corrupted member, you must save the member at the development group to refresh the accounting data. You can save the member using SCLM edit, migrate in forced mode, or the SAVE service. Then build and promote the member as usual.

The promote function:

- Determines which components are eligible for promotion
- Verifies that the application is complete and current
- Promotes the components that are at the current group and within the scope of the promote
- Potentially purges the components from the current group (and possibly lower key groups)
- Generates a promote report

Promote gives you an easy and efficient method to move data through a hierarchy. As you build software components, they become eligible for promotion to the next group in the hierarchy. Promote is based on architecture or source members; thus

you must build software components successfully before you can promote them to the next group. Using architecture members, you can promote individual software components or sets of software components during one promote. SCLM processes all data types associated with a component as a unit.

When the promote is complete, the promote function generates a report identifying the components promoted.

You also can specify that only a Promote Report be generated. The Promote Report identifies those components in the hierarchy that would be copied or moved if the promote function were to be invoked.

The panel shown in Figure 85 appears when you select Option 5, Promote, from the SCLM Main Menu.

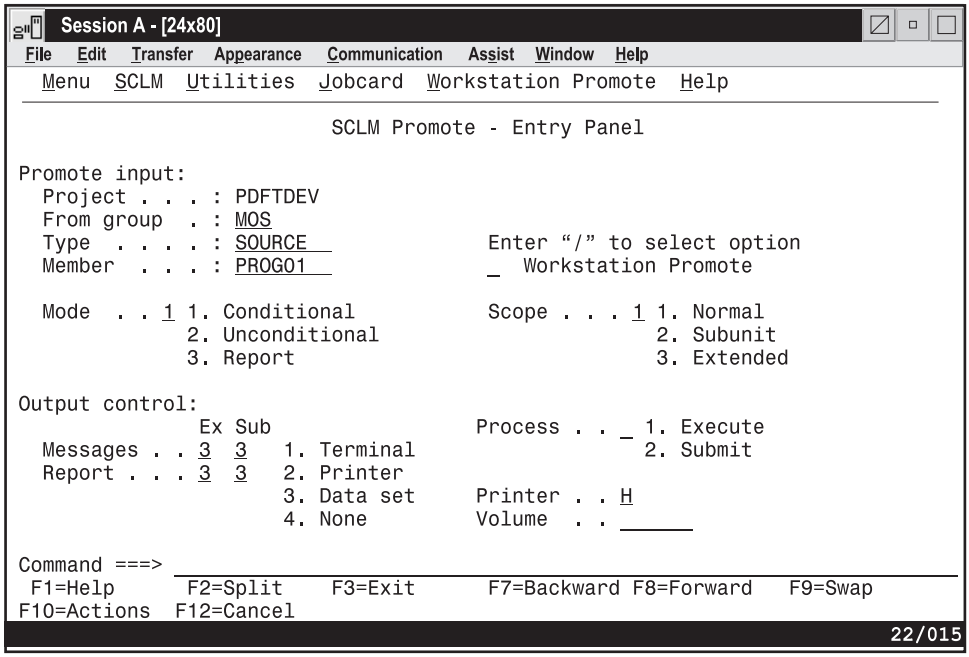


Figure 85. SCLM Promote (FLMP#P)

Note: The NRETREIV command key is enabled to work with this option. See "Name Retrieval with the NRETRIEV command" on page 145 for more information.

The fields on the SCLM Promote - Entry panel are:

Project	The project that you specified on the SCLM Main Menu. An **Alternate** field also appears if you specified an alternate project.
From group	The group from which to promote the member
Type	The type of the member
Member	The name of the member to be promoted

Scope	Select one of the following: **Normal** To process the components and members directly referenced by the specified architecture member. In addition, this scope processes upward dependencies for all Ada-type source members referenced directly by the architecture member and all source members referenced as upward dependencies. **Subunit** To process the components and members processed in normal scope as well as downward dependencies for all Ada-type source members referenced directly by the architecture members. **Extended** To process the components and members processed in normal scope as well as downward dependencies for all source members within the normal scope. **Note:** You must specify a scope equal to or greater than the scope specified with the SCOPE keyword in the FLMLANGL macro.
Mode	Select one of the following: **Conditional** To bypass the copy and purge steps if promote discovers a verification error. Promote compares dates in the build maps against dates in the database for all software components taking part in the promote. Software components are not promoted if they are deemed out of date. Use this mode to guarantee complete project integrity. **Unconditional** To perform copy and purge processing of all members despite verification errors of other members and to promote only those members with correct build map information. Use this mode to promote software components for incomplete or partial applications. For example, if some software components referenced by an architecture member are not complete but are required in the next group of the hierarchy, you can use this mode to promote those software components. The use of the unconditional mode does not guarantee application integrity, and you should use it with caution. It is, however, an effective method of promoting dependent software components that you plan to integrate at a later date. The **Unconditional** mode field is not retained on the Promote panel. If Unconditional is used, the panel is changed to Conditional when the promote returns to the panel. **Report** To perform verification and report generation processing. The report contains a list of members eligible for promotion.
Output control	Specify the destination for messages and the report when they are executed (Ex) or submitted (Sub), by entering the corresponding destination number: 1 for Terminal, 2 for Printer, 3 for Dataset, or 4 for None.

Workstation Promote	Specify whether or not the promote needs a workstation connection. For Foreground, SCLM verifies that an ISPF workstation connection exists before executing the promote. For Batch, SCLM verifies that the information required to initiate an ISPF workstation connection has been set by a previous build or promote or from the workstation build pull-down. If not, SCLM prompts the user to enter this information before the build job is submitted. If the promote does not require a workstation connection, do not use this field.
Process	You can call the processing part of the Promote - Entry Utility from the interactive or batch environment by selecting Execute or Submit, respectively. If you request batch processing by selecting Submit, you must specify the job statement information which is used in the JCL generated for batch processing. For information on using a unique jobname on the jobcard in batch processing, see "Batch Processing" on page 230.
Printer	Specify the printer output class.
Volume	Specify the volume on which SCLM should save data sets.

Promote Report

Figure 86 on page 227 shows an example of the promote report.

The promote report provides an accurate account of the promote. It lists all members promoted to the next group and all members purged from lower groups. It also marks "out-of-scope" software components with an asterisk (*).

Note: An *out-of-scope* software component is an architecture that is referenced with a LINK statement but not with an INCL statement. It is not within the domain of the architecture specified.

The report displays specific information according to the promote modes and scopes you select.

- For a promote of a member from a non-key group to a key group, the report indicates that the member was:
  - Copied to the next group
  - Purged from the "from" group
  - Purged from the last key group.
- For a promote of a member in a key group to a non-key group, it indicates that a copy was made.
- For a promote of a member in a key group to a key group, it indicates that a copy was made and a purge was performed on the source key group.
- For a second promote that follows a failed promote, it indicates the work completed by that promote only.

For more information on key and non-key groups, see "Key/Non-Key Groups" on page 143.

If a verification error occurs for a member, the report displays the message number that identifies the error in the **Message** field.

```
                SOFTWARE CONFIGURATION AND LIBRARY MANAGER (SCLM)

                          PROMOTE REPORT

                            02/15/2000  09:35:41


                            PROJECT:       PROJ1
                            TO GROUP:      INT
                            FROM GROUP:    STAGE1
                            TYPE:          ARCHDEF
                            ARCH. MEM.:    FLMO1LD3
                            ALTERNATE:     PROJ1
                            SCOPE:         NORMAL
                            MODE:          CONDITIONAL


** NOTE: “*” INDICATES “OUT OF SCOPE” ITEMS
```

Figure 86. Promote Report (Part 1 of 7)

```
                                                                    PAGE    2

 TYPE: ARCHDEF


                                                    COPIED TO  PURGED FROM  PURGED FROM
MEMBER      DATE       TIME     MESSAGE             INT        STAGE1       USER1
-------  ----------  --------  --------  ---------  -----------  -----------
FLM1ARH  02/14/2000  16:52:00                X          X            X
FLM1LD3  02/14/2000  16:54:00                X          X            X
```

Figure 86. Promote Report (Part 2 of 7)

```
                                                                    PAGE    3

 TYPE: LIST


                                                    COPIED TO  PURGED FROM  PURGED FROM
MEMBER      DATE       TIME     MESSAGE             INT        STAGE1       USER1
------   ----------  --------  --------  ---------  -----------  -----------
FLO1MD3  02/15/2000  09:30:00                X          X            X
FLO1MD5  02/15/2000  09:29:00                X          X            X
FLO1MD6  02/15/2000  09:29:00                X          X            X
```

Figure 86. Promote Report (Part 3 of 7)

```
                                                                    PAGE    4
 TYPE: LMAP

                                                  COPIED TO  PURGED FROM  PURGED FROM
MEMBER      DATE       TIME     MESSAGE          INT        STAGE1       USER1
-------  ----------  --------  --------  ---------  -----------  -----------
FL01LD3  02/15/2000  09:30:00                X           X            X
```

Figure 86. Promote Report (Part 4 of 7)

```
                                                                    PAGE    5
 TYPE: LOAD

                                                  COPIED TO  PURGED FROM  PURGED FROM
MEMBER      DATE       TIME     MESSAGE          INT        STAGE1       USER1
-------  ----------  --------  --------  ---------  -----------  -----------
FL01LD3  02/15/2000  09:31:00                X           X            X

                                                                    PAGE    6
 TYPE: OBJ

                                                  COPIED TO  PURGED FROM  PURGED FROM
MEMBER      DATE       TIME     MESSAGE          INT        STAGE1       USER1
-------  ----------  --------  --------  ---------  -----------  -----------
FL01MD3  02/15/2000  09:30:00                X           X            X
FL01MD5  02/15/2000  09:29:00                X           X            X
FL01MD6  02/15/2000  09:29:00                X           X            X
```

Figure 86. Promote Report (Part 5 of 7)

```
                                                                    PAGE    7
 TYPE: SOURCE

                                                  COPIED TO  PURGED FROM  PURGED FROM
MEMBER      DATE       TIME     MESSAGE          INT        STAGE1       USER1
-------  ----------  --------  --------  ---------  -----------  -----------
FL01LD3  02/14/2000  16:33:00                X           X            X
FL01LD3  02/14/2000  17:03:00                X           X            X
FL01LD3  02/14/2000  16:48:00                X           X            X

                                                                    PAGE    8
 TYPE: SOURCE2

                                                  COPIED TO  PURGED FROM  PURGED FROM
MEMBER      DATE       TIME     MESSAGE          INT        STAGE1       USER1
-------  ----------  --------  --------  ---------  -----------  -----------
INCLUDE2 02/14/2000  19:49:00                X           X            X
INCLUDE3 02/14/2000  16:50:00                X           X            X
```

Figure 86. Promote Report (Part 6 of 7)

```
                                                                        PAGE    9
********************************************************************************
**                                                                            **
**                          BUILD MAPS                                        **
**                                                                            **
********************************************************************************

                                                                        PAGE   10

 TYPE: ARCHDEF


                                                COPIED TO  PURGED FROM  PURGED FROM
MEMBER      DATE        TIME     MESSAGE         INT          STAGE1       USER1
-------   ----------  --------  --------  ---------  -----------  -----------

FL01LD3   02/15/2000  09:28:35                   X            X            X


                                                                        PAGE   11

 TYPE: SOURCE


                                                COPIED TO  PURGED FROM  PURGED FROM
MEMBER      DATE        TIME     MESSAGE         INT          STAGE1       USER1
--------  ----------  --------  --------  ---------  -----------  -----------

FLM01MD3  02/15/2000  09:28:35                   X            X            X
FLM01MD5  02/15/2000  09:28:35                   X            X            X
FLM01MD6  02/15/2000  09:28:35                   X            X            X
```

Figure 86. Promote Report (Part 7 of 7)

Processing Errors

The Promote function can recover from most SCLM environment errors. However, data set overflow and data contention, as described as follows, can occur during a promote.

Data Set Overflow

Partitioned data sets tend to become full and require compression. When a target data set runs out of space during a promote, promote attempts to recover and continue the promote. Although you get system ABEND messages, the promote ignores the ABEND and continues. However, processing bypasses making a copy to this data set and it also bypasses the subsequent purge step for members that were not copied.

If data set overflow occurs, follow these steps:

1. Compress or reallocate the data set with larger space allocations.
2. Increase the directory block allocation, if necessary.
3. Promote again.

The second promote copies only the members that did not copy in the original promote. If successful, the purge step is normal. The resulting promote report identifies only the copied and purged members in the second promote.

Data Contention

Be careful when you process certain combinations of SCLM builds and promotes simultaneously. You should not promote or build members that have not completed processing for another promote. Compiler errors or promote verification errors in one or more of the concurrent jobs can occur. You can normally recover from most errors by running the failed function again.

Command (Option 6)

To use the SCLM command shell, select Command (option 6) from the SCLM Main Menu. The panel shown in Figure 87 appears.

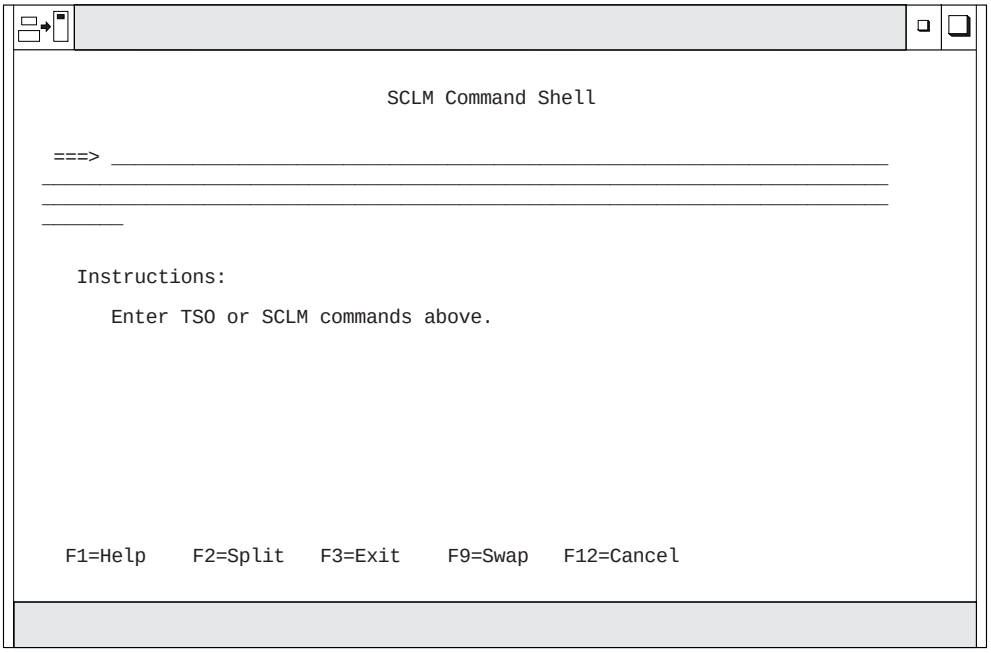


Figure 87. SCLM Command Shell (FLMHETSO)

Use this panel to execute TSO, CLIST, REXX execs, or SCLM commands from within SCLM.

Batch Processing

The Verify Batch Job Information panel shown in Figure 88 on page 231 is the standard panel for the SCLM functions that allow you to select batch processing. When you enter SUBMIT and when the JOB statement is not on the submittal panel, this panel appears. SCLM requires JCL job statements when you process in batch mode.

Note: SCLM can automatically generate unique jobnames. If you use the jobname USERIDx, where `x` is a letter of the alphabet or a digit, SCLM increments this letter or number by one for the next job. For example, if your USERID is SMITH, and your jobcard is submitted with the jobname `SMITH3`, the jobname is updated to `SMITH4`.

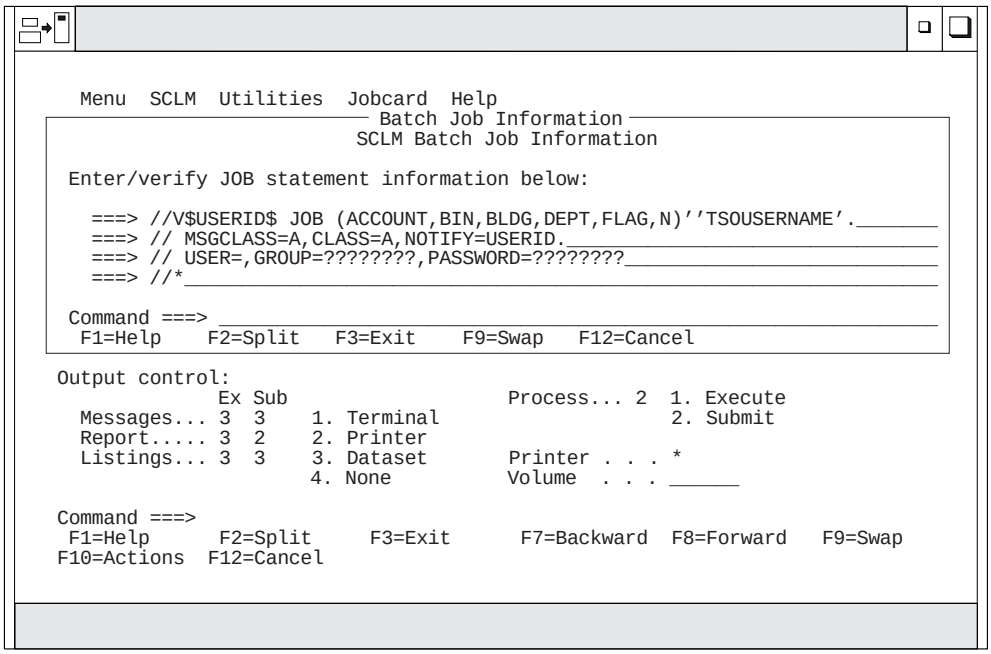


Figure 88. Verify Batch Job Information (FLMDSU#P)

Output Disposition

The Output Disposition panel shown in Figure 89 is the standard end panel for many SCLM functions when you have sent output to a data set. It allows you to determine the disposition of the report or messages data set previously displayed. You can choose between keeping the data set, deleting the data set, printing and keeping the data set, or printing and deleting the data set.

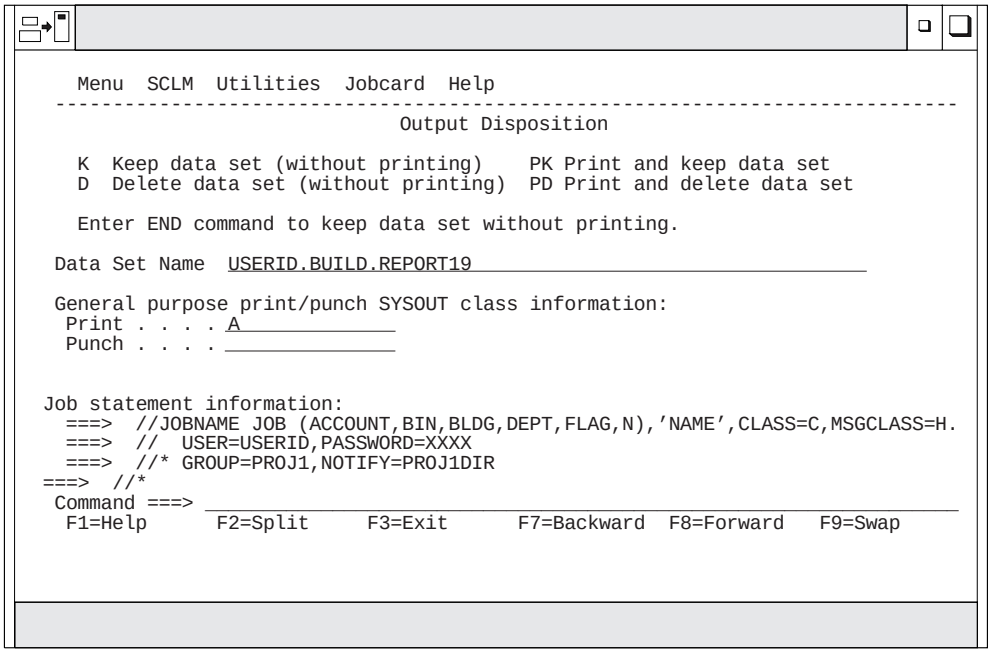


Figure 89. Output Disposition (FLMDEXT)

Output Disposition

When you send output to a data set, the database contents, architecture, build, and promote functions display a report data set if they complete with an acceptable return code. The migration utility displays a message data set because its report is a set of messages.

If you allocate the output to a data set and 99 data sets have already been allocated, SCLM either overlays a new data set over an old one or concatenates a new data set with an old one. To avoid this problem, delete old data sets to allow allocation of new data sets.

If error conditions occur in any of these functions (except build translator errors) and SCLM routes messages to a data set, SCLM displays the message data set, not the report data set. In either case, the Output Disposition panel appears after you finish browsing the displayed data set.

The view, edit, library, sublibrary management, and audit and version utility functions do not create report or message data sets and, consequently, do not display the Output Disposition panel.

Sample Project Utility (Option 7)

The SCLM Sample Project Utility makes it easier to create a sample SCLM project to use in learning the functions of SCLM, or as the basis for building a project for production use. In addition, you can use the Sample Project Utility to delete a project that was built using the utility.

The SCLM Sample Project Create function, Option 10.7.1, creates the data sets required for a simple SCLM project (including the VSAM accounting data base). It also creates a data set listing information about the project.

You must provide the names of several existing data sets on your system (such as, the ISPF macros data set), and the location of the High Level Assembler on your system. You have a choice of including a PLI sample if you have the PLI Optimizing Compiler installed on your system.

You do not need knowledge of assembler or link editing. The utility customizes, assembles, and link edits the project definition for you. The architecture definitions are then imported from the ISPF sample library and the sample application is built and promoted to the top level of the hierarchy. The project is then ready to use for the Development Scenario described in “Chapter 10. Development Scenario” on page 233. Use this scenario to learn the capabilities of SCLM.

The SCLM Sample Project Delete function, Option 10.7.2, deletes a project that was created with the Create utility. This function uses the information data set created by the Create utility to identify the data sets to delete.

Chapter 10. Development Scenario

This chapter uses a sample application to describe the basic tasks you typically perform using SCLM. The sample data sets referred to in the example are shipped with the ISPF product.

"Chapter 1. Defining the Project Environment" on page 3 provides step-by-step instructions for the project manager to define the sample project for this scenario. You can also define the sample project using Option 10.7, the SCLM Sample Project utility. No knowledge of SCLM is required to use the utility. You can use this hierarchy to gain some basic experience using SCLM. After examining some of the project data sets and performing some SCLM operations, you will have a better understanding of how SCLM can help you in your project activities.

This chapter walks you through the functions from the SCLM Main Menu. For a complete description of the SCLM Main Menu options, see "Chapter 9. Using SCLM Functions" on page 145.

Understanding the Hierarchy and the SCLM Main Menu

This section provides an overview of the sample hierarchy and briefly describes the functions available from the SCLM Main Menu.

The sample project uses a three-layer hierarchy composed of four groups. Figure 90 is used to represent the SCLM hierarchy in this sample.

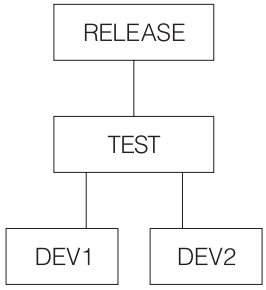


Figure 90. Sample Project Hierarchy

Throughout the remainder of this chapter, this sample project is called PROJ1. If the name established by your project manager is different, or you used a different name to define the project using the SCLM Sample Project utility (Option 10.7), use that name instead.

The sample application is composed of six programs that are used to build an application called FLM01AP1, as shown in Figure 91 on page 234. The programs are linked into four load modules. The four load modules are organized as two subapplications, which in turn are components of FLM01AP1.

Note: If the PLI Optimizing Compiler is not included as a language in the sample project, the application consists of five programs linked into three load modules.

The sample that follows assumes that the SCLM project setup activities have been completed as described in the *ISPF Software Configuration and Library Manager (SCLM) Developer's and Project Manager's Guide* or that you have defined the sample project using the SCLM Sample Project utility (Option 10.7).

After the sample project has been defined, you can take the following steps to begin using SCLM.

1. Log on to MVS.
2. Start ISPF to display the ISPF Primary Option Menu.
3. Select SCLM and press Enter. The SCLM Main Menu is displayed.

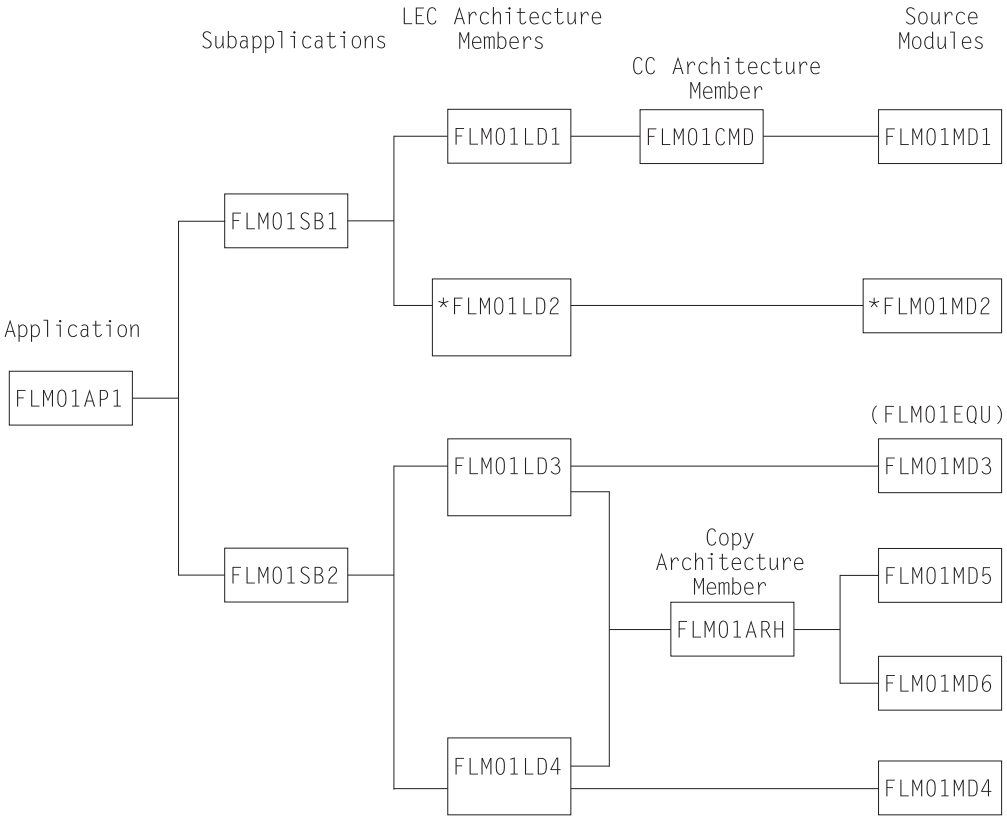


Figure 91. Application FLM01AP1

Note: Source module FLM01MD2 and architecture member FLM01LD2 are included only if PLI Optimizing Compiler is included as a language if the sample is defined using the SCLM Sample Project utility (Option 10.7).

Understanding the Architecture Definition

This section describes the architecture definition and its importance in an SCLM project. The architecture definition describes to SCLM how the components of an application fit together. For more information on architecture definitions, see "Chapter 11. Architecture Definition" on page 247.

There are four types of architecture members:

HL (high level) HL architecture members reference application and subapplication components.

CC (compilation control) CC architecture members contain the information to produce and track software components with object module output.

LEC (link edit control) LEC architecture members contain the information to produce a complete load module.

Generic Generic architecture members identify the source member or groups of source members to be processed by a processor other than a standard compiler. The sample project does not contain examples of generic architecture members.

If you have several architecture definition statements that are used together in many places, you can put them into a member and reference the member using the COPY statement wherever you need the statements. When you use the COPY statement, the contents of the specified member are inserted directly into the respective architecture members.

1. Select View from the SCLM Main Menu. Specify PROJ1 in the **Project** field and specify DEV2 in the **Group** field. Press Enter.
2. Specify ARCHDEF in the **Type** field and leave the **Member** field blank. Press Enter. The architecture members are shown in the following table.

Member	Type	Comments
FLM01AP1	HL	References FLM01SB1 and FLM01SB2 with the INCL statement. A build performed on FLM01AP1 results in a complete build for all the code in the project, if necessary.
FLM01SB1	HL	References FLM01LD1 and FLM01LD2 with the INCL statement. A build performed on FLM01SB1 results in a complete build of the FLM01SB1 subapplication, if necessary. If the PLI Optimizing Compiler is not included as a language in the sample project, FLM01SB1 references only FLM01LD1.
FLM01SB2	HL	References FLM01LD3 and FLM01LD4 with the INCL statement. A build performed on FLM01SB2 results in a complete build of the FLM01SB2 subapplication, if necessary.
FLM01LD1	LEC	Directs SCLM to produce the load module and load map for FLM01LD1. The INCL statement references architecture member FLM01CMD. The PARM statements pass parameters to the SCLM BUILD translators.
FLM01LD2	LEC	Directs SCLM to build load module FLM01LD2 from the source FLM01MD2. The INCLD architecture statement is used to identify FLM01MD2 as the source. Note that LOAD, LMAP, and SOURCE are types identified by the FLMTYPE macro in the project definition. If the PLI Optimizing Compiler is not included as a language in the sample project, FLM01LD2 is not inlcuded.
FLM01CMD	CC	Directs SCLM to produce object code from FLM01MD1. SINC identifies FLM01MD1 as the source member. Note that in addition to object code (OBJ), there is also source listing (SOURCLST). OBJ and SOURCLST are identified in the project definition with the FLMTYPE macro.
FLM01LD3	LEC	References FLM01MD3 with the INCLD statement. Other modules are referenced with the copy of FLM01ARH. In this example, FLM01ARH references FLM01MD5 and FLM01MD6. FLM01LD3 indirectly references FLM01MD5 and FLM01MD6 via the COPY statement in FLM01ARH.

Member	Type	Comments
FLM01LD4	LEC	References FLM01MD4 with the INCLD statement. Other modules are referenced with the copy of FLM01ARH. In this example, FLM01ARH references FLM01MD5 and FLM01MD6. FLM01LD4 indirectly references FLM01MD5 and FLM01MD6 via the COPY statement in FLM01ARH.
FLM01ARH	CC	References modules FLM01MD5 and FLM01MD6 with the INCLD statement. The LEC architecture members FLM01LD3 and FLM01LD4 use the COPY directive to copy the contents of FLM01ARH into their members for a build.

To create an architecture report:

1. Select Architecture Report (option 3.5) from the SCLM Main Menu, and press Enter.
2. Type:

`ARCHDEF`	in the **Type** field
`FLM01AP1`	in the **Member** field
`6`	in the **Report cutoff** field
`1`	in the **Process** field
`1`	in the **Messages** field
`1`	in the **Report** field

Press Enter.

The output shows the hierarchy, the kinds of architecture members (HL, CC, and LEC), and various cross-references. See "Architecture Report Example" on page 191 for an example of the architecture report.

Sample SCLM Development Cycle

Your typical daily operations using SCLM might flow like this: edit (SCLM editor), compile (Build), and test, repeating this cycle until testing is complete, and then promote. After the promote is performed, you or other developers can use the SCLM editor to automatically draw members down to a development group for modification.

The following list includes steps that you might perform in the development cycle of a software component or any type of data that is under SCLM control. Figure 92 on page 238 illustrates the project flow of the following steps. The hierarchy used for this example is shown in Figure 90 on page 233.

1. The developer draws down a source member from group RELEASE to group DEV1 and modifies it. The data at group RELEASE is the current release of the project. Changes are now being made for the next release. When the developer has made the modifications to the member, SCLM parses the member and registers it with SCLM. The successful registering of the update makes this member available for use by other SCLM functions.
2. The Build function is initiated against an architecture definition that includes this parsed and stored source member. This build creates object modules reflecting the changes that were made to the source member. The source, architecture definition, and object module members used here have been given

the same member names. Thus, you can easily see how these members are related, although their types are different. These naming conventions, however, are not required by SCLM.

If the Build function does not complete successfully because of errors in the modified members, you must use the SCLM editor again to correct the errors, and try to build again.

3. The developer can now test the effect the changes have made to the application.
4. The developer then moves all the changed data to the group TEST by invoking PROMOTE using the same architecture definition that was previously built. The data changes are now available to all developers because they have reached a common group. If any changes in data made by the developer conflict with changes other developers are making in their development groups, these changes are found when the other developers build their changes at their development group.

   Alternately, the person appointed as SCLM project manager can do the promote. The SCLM project manager is the person who has UPDATE authority to TEST and promote changes to this group. The SCLM project manager can guarantee all changes promoted to the group TEST have been unit tested (because the project manager can control the promotes).
5. When all changes scheduled for the next release have been promoted to the group TEST, testing the application can occur at this group while other programmers are still developing software in the development groups.
6. Finally, after system testing is complete in the TEST group, the new release of the project can be promoted to the RELEASE group.

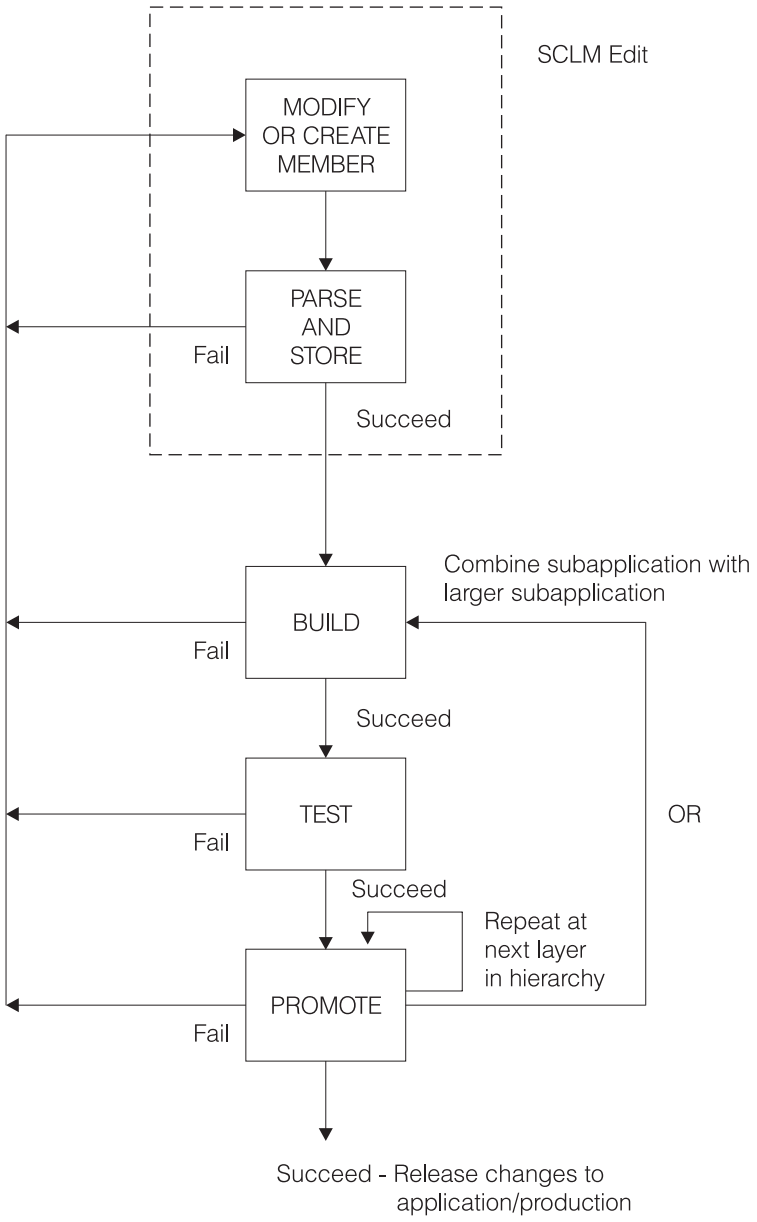


Figure 92. Development Cycle

Using the SCLM Editor

This section describes how to alter code using the SCLM editor. To illustrate how SCLM protects project members from unintentional updates, you will change the FLM01EQU member and create an error situation. This error causes the BUILD to fail and prevents a PROMOTE until you correct the error.

FLM01EQU is an included member in FLM01MD3. SCLM automatically tracks included members, so you do not have to specify their relationship in your architecture definition.

1. Return to the SCLM Main Menu, and specify `DEV2` in the **Group** field. Select the Edit option and press Enter.

2. Select SOURCE in the **Type** field and FIX01 in the **Change code** field. Press Enter to bring up the Edit Member list.
3. Select FLM01EQU from the Edit Member list. Note that FLM01EQU is in the RELEASE group and a draw down from the RELEASE group to the DEV2 group takes place.
4. From the command line, issue the SETUNDO ON command. Different system installations will have different profile defaults set, so issuing this command will ensure that you have PDF Edit UNDO set On.
5. Duplicate the line R4 EQU 4 and change WORK REGISTER in the comment to DEV2 ERROR. Press Enter.
6. From the command line, issue UNDO: type Undo on the command line and press Enter. The change to the comment is removed. The duplicate line remains. Note that UNDO works only if your profile has UNDO set to ON.
7. Reenter the change to create the error situation for this example from 4.
8. Use the split screen option. Select SCLM from the ISPF Primary Option Menu. Select Edit, specify PROJ1 in the **Project** field, and specify DEV1 (DEV1 is another development group in this SCLM project) in the **Group** field.

   Attempt to edit FLM01EQU by typing FLM01EQU in the **Member** field and pressing Enter. Press the Help key twice to retrieve the long message describing the error condition. SCLM locked FLM01EQU for DEV2 at the time of the draw down. FLM01EQU cannot be updated by another group until a PROMOTE is issued from DEV2 or FLM01EQU (member and accounting record) is deleted from DEV2. End split screen.
9. Return to the DEV2 edit screen and issue the SPROF edit command: type SPROF on the command line and press Enter. Note that the language is ASM and the change code is FIX01. SCLM prompts you for a language when a member is created. You can use SPROF to change the language SCLM associates with the member. Press Enter to return from the SCLM Edit Profile Panel to the SCLM Edit panel.
10. Press the End key to save the member and end the edit session. Use the Help key to display the long message, which indicates that SCLM parsed and stored the member.

    Press the End key twice to return to the SCLM Main Menu.

Understanding the Library Utility

This section describes the library utility functions typically used by developers. You can use the library utility to browse and delete components and the accounting information that is generated with edit/save, build, and promote activities.

1. Select Utilities from the SCLM Main Menu, and press Enter.
2. Select Library, and press Enter.
3. To browse the accounting record for PROJ1.DEV2.SOURCE(FLM01EQU), type:

A	on the command line
DEV2	in the **Group** field
SOURCE	in the **Type** field
FLM01EQU	in the **Member** field

   Press Enter.

Notice the date and time of the last update (**Change date** and **Change time** fields) for FLM01EQU.

4. To display the statistics, select the **Display statistics** field and press Enter.
5. Return to the accounting record by pressing the End key once. Note that the FLM01EQU has one change code. To display the change code, select the **Number of change codes** field and press Enter. The change code `FIX01` appears along with the Change date and Change time.
6. Return to the Library Utility panel by pressing the End key twice.
7. To browse the member PROJ1.RELEASE.SOURCE(FLM01MD3), type:

`B`	on the command line
`RELEASE`	in the **Group** field
`FLM01MD3`	in the **Member** field

Press Enter.

Notice that FLM01MD3 contains a COPY statement for FLM01EQU.

8. Press the End key until you are back at the SCLM Main Menu.

Using Build

This section illustrates how to use the SCLM build processor when one of the members has an error. The SCLM build processor translates all members and all modules that have been affected by alterations. A build operation prepares the member for a promote operation.

1. Select the Build option from the SCLM Main Menu, and press Enter.
2. Execute a Build operation by typing:

`DEV2`	in the **Group** field
`ARCHDEF`	in the **Type** field
`FLM01AP1`	in the **Member** field
`/`	in the **Error listings only** field
`1`	in the **Mode** field.
`2`	in the **Scope** field
`1`	in the **Messages** field
`1`	in the **Report** field
`3`	in the **Listings** field

Press Enter.

Notice that you did not have to type **EX** on the command line or re-enter a value in the Process field. You set this value when you created the Architecture Report. The value is carried from panel to panel and is maintained as is until you change it.

3. Note the return code of 8 from the assembler. There is also an error from the translator for FLM01MD5, which contains FLM01EQU. The assembler listing is contained in *tso-prefix*.BUILD.LIST*nn*.

   Because of the assembler error, SCLM Build will place you in Browse of the LISTING data set (*tso-prefix*.BUILD.LIST*nn*). Note that the error is the duplicate symbol R4.

   If you are not using tso-prefix, your user ID will replace tso-prefix.

4. When you are finished browsing the LISTING data set, press the End key. The Output Disposition panel appears. Type `D` to delete the LISTING data set, or type `K` to keep the LISTING data set. After pressing Enter, the Build panel appears.

Because the FLM01EQU member has changed and because FLM01MD5 contains the FLM01EQU member, Build attempts to assemble and link FLM01MD5. However, FLM01EQU contains the error you previously entered (a duplicate symbol for R4) so nothing is assembled or linked.

Editing the Member to Correct Errors

This section describes how to re-edit the FLM01EQU member to correct the error you introduced previously.

1. Select Edit from the SCLM Main Menu, leave PROJ1 in the **Project** field and DEV2 in the **Group** field. Press Enter.
2. Specify `FLM01EQU` to edit the FLM01EQU member in PROJ1.DEV2.SOURCE.
3. Remove the duplicate R4 equate line.
4. Save the changes by pressing the End key.

Attempting to Promote a Member before Performing a Build

This section describes how SCLM protects the integrity of your project hierarchy by not allowing you to promote a member that has not been successfully built. The promote operation copies changed members up into the next group in the library structure.

The build operation you attempted previously was unsuccessful. Therefore, the promote you attempt in this section will also be unsuccessful. SCLM maintains synchronization between source and object by ensuring that only successfully built members can be promoted. This safety feature addresses the common problem of forgetting to recompile changed modules.

1. Select Promote from the SCLM Main Menu.
2. On the Promote panel, type:

`DEV2`	in the **From group** field
`ARCHDEF`	in the **Type** field
`FLM01AP1`	in the **Member** field.
`1`	in the **Mode** field
`1`	in the **Scope** field
`1`	in the **Messages** field
`1`	in the **Report** field

Press Enter.

SCLM issues date and time mismatch error messages because the FLM01EQU source has been updated and the modules that use it have not been recompiled by the build operation. Promote sends a return code of 8 because the date and time mismatch prevented it from copying anything to the next group.

Rebuilding the Changed Member

This section illustrates a successful build operation. Because all members are not affected by the change to the FLM01EQU member, only the members containing FLM01EQU are recompiled and linked. SCLM processes project components efficiently by recompiling and relinking only those modules that were altered since the last build operation.

1. Select Build from the SCLM Main Menu and press Enter.
2. On the Build panel, type:

DEV2	in the **Group** field
ARCHDEF	in the **Type** field,
FLM01AP1	in the **Member** field
1	in the **Mode** field
2	in the **Scope** field
1	in the **Messages** field
1	in the **Report** field
3	in the **Listings** field

   Press Enter.

   Note the traversal of the architecture. FLM01MD2 was not affected by the change to the FLM01EQU member and will not be recompiled. FLM01LD2, which contains only FLM01MD2, will not be relinked.
3. Verify that the build completed successfully (RETURN CODE = 0). If the return code is not zero, check the listing, correct the errors, and try again.

Using the Database Contents Utility

This section illustrates use of the database contents utility to verify that the compilations and links were performed.

1. Select the Utilities option from the SCLM Main Menu.

   Select the Database Contents Utility option from the SCLM Utilities Menu.
2. On the Database Contents Utility panel, type:

DEV2 TEST RELEASE	in the **Group** fields
SOURCE	in the **Type** field
*	in the **Member** field
/	in the **Change additional selection criteria** field
1	in the **Messages** field
1	in the **Report** field
3	in the **Tailored output** field

   Press Enter. The Additional Selection Criteria panel appears.
3. On the SCLM Database Contents - Additional Selection Criteria panel, type * for the **Authorization code, Change code, Change group, Change user id**, and **Language** fields. Do not select the **First occurrence only** field.

   Type:

1	in the **Data type** field
3	in the **Architecture control** field
1	in the **Scope** field

These are the default values.

Press Enter. The Customization Parameters panel appears.

4. On the Customization Parameters panel, select the **Page headers** and **Show totals** fields, and enter `Statistics Report` for the **Report name** field. Type `@@FLMMBR @@FLMLAN @@FLMCML @@FLMNCL @@FLMBLL @@FLMTLS @@FLMCMS @@FLMNCS` for the **Report line format** field after the prompt.

   Put at least 2 spaces between each @@FLMxxx variable. This can wrap to the next line; this field accepts up to 160 characters. These are the default values. Press Enter to execute the database contents utility report.

   Note that only FLM01EQU is in the DEV2 group. The Database Contents Utility panel reappears.

5. On the Database Contents Utility panel, type:

`DEV2 TEST RELEASE`	in the **Group** fields
`OBJ`	in the **Type** field

Do not select the **Change additional selection criteria** field.

Press Enter. Press Enter again on the Customization Parameters panel.

Note that FLM01MD2 does not appear in the DEV2 group. FLM01MD2 was not affected by the changes to FLM01EQU.

6. On the Database Contents Utility panel, type:

`DEV2 TEST RELEASE`	in the **Group** fields
`LMAP`	in the **Type** field

Press Enter. Press Enter again on the Customization Parameters panel.

Note that FLM01LD2 does not appear in the DEV2 group. FLM01LD2 was not affected by the changes to FLM01EQU.

7. On the Database Contents Utility panel, type:

`DEV2 TEST RELEASE`	in the **Group** fields
`LOAD`	in the **Type** field

press Enter. Press Enter again on the Customization Parameters panel.

Note that FLM01LD2 does not appear in the DEV2 group. FLM01MD2 was not affected by the changes to FLM01EQU.

Promoting a Member Successfully

This section illustrates a successful promote operation. The FLM01EQU member is moved from the DEV2 group to the TEST group.

1. Select the Promote option from the SCLM Main Menu, and press Enter.
2. On the Promote panel, type:

`DEV2`	in the **From group** field
`ARCHDEF`	in the **Type** field
`FLM01AP1`	in the **Member** field

1 in the **Mode** field
1 in the **Scope** field
1 in the **Messages** field
1 in the **Report** field

Press Enter.

3. Verify that the promote completed successfully (RETURN CODE = 0). If the return code is not zero, check the messages, correct the errors, and try again.

   When the Promote panel reappears, press the End key to return to the SCLM Main Menu.

4. Select the Utilities option from the SCLM Main Menu.

   Select Database Contents Utility from the SCLM Utilities Menu. On the Database Contents Utility panel, type:

DEV2 TEST RELEASE in the **Group** fields
* in the **Type** field
FLM01EQU in the **Member** field
1 in the **Messages** field
1 in the **Report** field
4 in the **Tailored output** field

Do not select the **Change additional selection criteria** field.

Press Enter. The Database Contents Utility panel reappears.

5. On the Database Contents Utility panel, type:

DEV2 TEST RELEASE in the **Group** field
SOURCE in the **Type** field
* in the **Member** field
/ in the**Change additional selection criteria** field
1 in the **Messages** field
1 in the **Report** field
4 in the **Tailored output** field

Press Enter. The Additional Selection Criteria panel is displayed.

Type FIX01 in the **Change code** field. Press Enter again. Only FLM01EQU should be found, and it should only be found at TEST. The Database Contents Utility panel reappears.

6. Return to the SCLM Main Menu by pressing the End key twice.

Drawing Down a Promoted Member

This section illustrates that a promoted member is available and can be edited by other developers.

1. Specify Edit from the SCLM Main Menu, PROJ1 in the **Project** field, and DEV1 in the **Group** field.
2. Edit the FLM01EQU member, by specifying SOURCE in the **Type** field and FLM01EQU in the **Member** field. However, do not make any changes to the member. Note that FLM01EQU is no longer locked by SCLM.

Performing Project Housekeeping Activities

After you complete the development activities described in this chapter, be sure to perform any cleanup or housekeeping activities in preparation for the next project operations. You can clean up the sample project hierarchy by performing a promote operation using group TEST, type ARCHDEF, and member FLM01AP1. This restores the hierarchy to its original state so that others can use it to execute this scenario. If you made other changes (such as a change to the FLM01EQU member in the last activity), you might need to perform additional build and promote operations.

You can also delete the *tso-prefix*.BUILD.LIST*nn* and *tso-prefix*.DBUTIL.CMD*nn* data sets created during the preceding SCLM Build process.

Chapter 11. Architecture Definition

An architecture definition describes the configuration of an application under SCLM control and how it is to be constructed and integrated. Architecture definitions are created and updated by the developers and describe the architecture of an application. They provide specifications to the Build function for data generation, and to the Promote function for the movement of data from one group to another. Architecture definitions can reference other architecture definitions, thus providing a simple building block tool for complex application definitions.

- Data Generation

  Architecture definitions can specify the following information to the build function:
  - Where inputs to translators (for example, compilers) are to come from
  - Where outputs from translators are to be stored
  - What parameters are needed by a translator.

  A single architecture definition can specify all the data generation to occur for a large, complex application simply by referencing other architecture definitions.
- Data Movement

  All data that is directly or indirectly referenced by an architecture definition is promoted when that architecture definition is promoted. This encompasses included architecture definitions, along with the system components they describe. Thus, specifying a single high-level architecture definition for promotion can cause an entire application to be promoted.

This chapter discusses the methods you can use to define the architecture, provides several different examples of architecture members, and explains the use of architecture member statements.

Architecture Members

Architecture members define the application at a high level by referencing lower level architecture members. You can generate them top down or bottom up, using an iterative approach. Create architecture members by using the edit function.

The capability to define an architecture allows you to control and track any discrete division of an application from the most encompassing definition down to the individual component. You can maintain the architecture members in a separate type in the project data base. Use the architecture members to describe the different versions or variations of a project or application.

Kinds of Architecture Members

SCLM provides four kinds of architecture members that you can use to generate an architecture definition for an application. They are compilation control (CC), linkedit control (LEC), high-level (HL), and generic.

Each kind of architecture member controls a different kind of component that SCLM processes. Table 17 on page 248 categorizes the use of each kind of architecture member.

Table 17. Uses of Architecture Members

Architecture Member	Use
Compilation Control (CC)	Define compiler processed components.
Linkedit Control (LEC)	Define link edit processed components.
High-Level (HL)	Define application and subapplication components.
Generic	Define specially processed components.

Each of these uses is described in the following pages. See "Sample Application Using Architecture Definitions" on page 261 for an example of an application consisting of architecture members.

Defining Compiler Processed Components

Standard compilers produce object modules as output. SCLM can be used to create object modules by using either a Compilation Control (CC) architecture member or a compilable source member as input to the build function. The following discusses both methods for producing object modules.

Compilation Control Architecture Members

One method of creating object modules is through a Compilation Control (CC) architecture definition.

CC architecture definitions contain all the information necessary to produce and track software components with object module output. Use CC architecture definitions to provide the following:

- The inputs to the compiler and other translators
- The outputs of the compiler and other translators
- Compiler options.

To directly identify an input to the compiler, use the SINC statement. If the input is generated from another member in the project, use the INCL and INCLD statements along with the KREF statement. The INCL and INCLD statements identify members built before compiling this member. The KREF statement identifies which outputs of the members on the INCL and INCLD statements are inputs.

CC architecture members must have at least one SINC statement and one OBJ statement. See "Architecture Statements" on page 254 for more information.

Members included by compiler include statements such as COPY are not identified in architecture members. SCLM obtains the list of included members from a parser that is run when a member is stored into SCLM and when members are updated. The information about the parser, the compiler, and include libraries outside the project is specified in a language definition. The language of a member must be identified to SCLM when a member is added to an SCLM project. The language of a member can be changed.

The ddnames used by the compiler are specified in the language definition by FLMALLOC macros. The types of ddnames are identified by different IOTYPEs. An IOTYPE of S identifies the input stream for the compiler. The input stream has two formats. One, identified by KEYREF=SINC, is a sequential work file that

contains all of the inputs to the compiler concatenated together. The other, identified by KEYREF=INCL, is a sequential work file that contains INCLUDE statements for each of the input members. The format of the INCLUDE statement is `INCLUDE DDNAME(MEMBER)`. The DDNAME will be a ddname dynamically allocated by SCLM. If multiple inputs are identified, they are concatenated in the order specified in the architecture member.

You can add information to the input stream passed to the compiler by using the CMD statement. The CMD statement can be used to add compiler directives, force titles, or control listings based on the commands supported by the compiler in the input stream.

You can append translator options to the options specified in the language definition by using the PARM statement. Use the statement as many times as necessary to specify all options you want (up to a string length of 512 characters).

You can pass parameters directly to specific build translators defined in the language definition by using the PARM*x* statement, coupled with the use of the PARMKWD parameter of the FLMTRNSL macro.

SCLM orders compiles to ensure that outputs (such as DB2 DBRMs) are produced before compiling the member that references them. SCLM orders compiles that are within the scope of the build. (See "Build (Option 4)" on page 217 for more information.)

SCLM allows you to track and maintain all forms of generated data. Often, due to space limitations, you do not want to save it all. SCLM gives you the option of saving listings in the database or discarding them. Therefore, the architecture member statement LIST is optional. SCLM can generate listings for viewing after a build.

Specifying Source Members

Specifying a compilable source member to the build function is the alternate method of creating object modules. The language definition of the source member from the project definition determines which translators are called and where outputs are saved during the build. Compiler parameters can only be overridden by creating a CC architecture member.

Defining Link Edit Processed Components

Standard linkage editors produce load modules as output. To define software components with load module outputs from standard linkage editors, use Linkedit Control (LEC) architecture members. LEC architecture members contain all the information necessary to produce a complete load module. Use the LEC architecture member to identify the following:

- The load module name and the type in which you want it saved
- The linkage editor listing name and the type in which you want it saved
- All object and other load modules the load module is to contain
- Linkedit control statements and linkage editor options.

LEC architecture members must have at least one LINK, INCL, INCLD, or SINC statement and one LOAD statement.

Linkedit Control (LEC) architecture members can be constructed by referencing any combination of source members, CC architecture members, generic architecture members or LEC architecture members. Inputs to LEC architecture members are

identified in the same way that inputs to CC architecture members are identified. The one difference is that by default LEC architecture members include object and load modules generated by the OBJ and LOAD statements in the input stream to the linkage editor. SINC statements can be used in LEC architecture members to identify object modules or load modules which are generated outside of the project. If SINC statements are being used to include load modules, the input ddname for the build translator must specify KEYREF=INCL. One additional statement can be used in LEC architecture members to identify an input to the linkage editor. That statement is the LINK statement. It identifies an output in the project that does not need to be rebuilt prior to being included in the input stream.

SCLM verifies that the inputs to the LEC architecture member are up-to-date prior to link editing the inputs. SCLM will rebuild any inputs that are outputs of building other members in the project when those outputs are out-of-date. The inputs specified on LINK statements are an exception. These inputs will not be rebuilt.

You can override default linkage editor options by using the PARM statement. Use the statement as many times as necessary to specify all options you want. SCLM uses the standard S/370 linkage editor as defined by the LE370 language definition unless an LKED statement is used to override the default. See page 258 for more information.

You can specify in the LEC that SCLM pass linkage edit control statements directly to the linkage editor by using the CMD statement. Insert the control statements along with the object and load modules by careful positioning in the LEC architecture member.

The CMD statement can be used to include object modules and load modules that are in data sets outside of the project. The language definition for the linkage editor must include a ddname referencing the data set containing the members to include.

Because of space limitations, you might not want online linkage editor listings. SCLM allows you to save listings in the database or discard them. Therefore, the architecture member statement LMAP is optional. Nonetheless, SCLM generates listings to temporary listing data sets for your viewing during the build.

You cannot use the SETSSI linkage editor command in an LEC architecture member. If SCLM finds a CMD SETSSI statement in an LEC architecture member during a build, the build function overrides the statement with its own SETSSI command.

SCLM Build and Control Timestamps

SCLM uses the Status System Index field to signify that the last update of a load module was made through SCLM. The **SSI** field data that SCLM generates consists of the following: the most significant bit is defined as a flag; the next most significant 11 bits specify hour and minute in binary form; and the least significant 20 bits specify Julian date in packed decimal form. SCLM sets the flag bit and writes these items into the **SSI** field during build processing when it generates a load module.

Table 18. SCLM Status System Index Field Data

Bit	Definition	Form
0	flag	bit

Table 18. SCLM Status System Index Field Data (continued)

Bit	Definition	Form
1-5	hour	binary
6-11	minute	binary
12-31	Julian date	packed decimal

Defining Application and Subapplication Components

You can define applications and subapplications by using High-Level (HL) architecture members. HL architecture members allow you to categorize groups of related load modules, object modules, and other software.

You can maintain one HL architecture member to define an entire application for a project. This HL architecture member references other architecture members that eventually reference every component in the application. It can also reference the source directly, with the language of the source defining the outputs to be produced. By using this HL architecture definition as input to the build or Promote functions you can ensure that the entire application is up to date or is promoted to the next group in the project hierarchy. A build or promote of an HL architecture member results in the building or promotion of every software component referenced. In this way, you can guarantee the integrity of an entire application.

You can also use an HL architecture member to define subapplication software components. Subapplications can be a combination of load modules or merely a list of internal data items to be controlled. Subapplications can, in turn, reference other subapplications to any depth. Conscientious use of HL architecture members contributes to application modularity.

SCLM can control and track ISPF panels, skeletons, and messages that are not processed by a compiler or linkage editor or used to invoke processors. Because these unique forms of software are not processed by compilers, linkage editors, or other processors, they are considered data dependencies and, therefore, can be controlled by using the PROM statement.

In most cases, you do not want panel, skeleton, and message dependencies in LEC, CC, and generic architecture members. Use HL architecture members to control all dialog software. For example, you can use one HL architecture member for panels, one for skeletons, one for messages, and one for the entire dialog that references the three previous HL architecture members.

The PROM statement date_check parameter allows SCLM to bypass date checking for the referenced member, thereby eliminating the need to build before promoting when that member is modified. Careful use of the PROM statement in this manner can eliminate unnecessary SCLM processing and improve efficiency.

Generic Architecture Members

Generic architecture members are used to process members that do not generate object modules. Examples of the outputs that might be produced are documentation and panels. Generic architecture members are almost the same as Compilation Control (CC) architecture members. The difference is that generic architecture members cannot generate object modules using the OBJ statement. If an OBJ statement is added to a Generic architecture member it becomes a CC

architecture member. Other output statements LIST, OUT1, etc. are used in generic architecture members to identify the listings, documentation, panels or other outputs produced.

Build and Promote by Change Code

You can also use architecture definitions to identify the parts associated with a specific change or group of changes. This can be done in any architecture member using the CCODE statement. In addition to the normal contents of an architecture definition, such an architecture member contains a list of CCODE keywords followed by a change code and include flag. An example of such an architecture definition follows:

```
* ARCHDEF FOR PACKAGE PKG00001
CCODE  POY66045 INCLUDE   * Include changes for problem POY66045
CCODE  POY66615 INCL      * Include changes for problem POY66615
INCL   SCLM     ARCHDEF   * SCLM ARCHDEF
```

There are no SCLM-enforced conventions for change codes. The only restriction is that it be a maximum of 8 characters. For SCLM to determine the change code, any change code that contains an embedded blank or whose first character is other than A-Z, 0-9, @, # or $ must be enclosed in delimiters. A delimiter can be any character not specified above. Following are some examples:

```
CCODE A                 * this includes change code A
CCODE ,A B C,  E        * this excludes change code A B C
CCODE /AB/     IN       * this includes change code AB
CCODE 'A B'    EX       * this excludes change code A B
CCODE 1"       EXCLUDE  * this excludes change code 1"
```

Valid values for the include flag are INCLUDE or EXCLUDE. When not specified, the default value is INCLUDE. A value of INCLUDE indicates that *only* the changes specified are included. A value of EXCLUDE indicates that *everything except* the specified changes are included. The following table illustrates the conditions under which SCLM will build and promote by change code.

MEMBER CHANGE CODE	CCODE CCODEX INCLUDE	CCODE CCODEX EXCLUDE
CCODEX	Yes	No
CCODEY	No	Yes
no change code	No	Yes

Multiple CCODE statements can be specified in an architecture definition. An error message is issued when the include flag value is not the same on all statements. Duplicate CCODE statements are ignored. Any CCODE statements whose change code and include flag resolve to the same value are considered duplicates. For example, the following CCODE statements are duplicates:

```
CCODE 1
CCODE '1 ' INCLUDE
```

CCODE and COPY keywords cannot be used in the same architecture definition. Because the COPY keyword causes an actual copy of an architecture definition to be inserted into the first, the architecture definition referenced by the COPY statement must also be free of CCODE statements. To build an architecture

definition containing COPY statements by change code, create a new architecture definition that contains the CCODE statement and an include (INCL) of the original architecture definition.

The concept of a package (group of changes) is supported through the ability to specify multiple CCODE keywords in an architecture definition. To more easily identify and maintain these architecture definitions, you can define a TYPE called PACKAGE with a language of ARCHDEF and use the package identifier or change code as the name for each member name. Or you can define a single architecture member and update the change code values in that member for each new build or promote by change code.

Only those CCODE statements that appear in the architecture definition specified as input to the build or promote will be processed. All other CCODE statements will be ignored. For example, assume that you have architecture definitions ISPF, PDF, SCLM and ISPFSUB. The architecture definitions contain the following statements:

```
* ARCHITECTURE DEFINITION MEMBER ISPF
INCL  ISPFSUB ARCHDEF
INCL  PDF     ARCHDEF
INCL  SCLM    ARCHDEF
CCODE A       INCLUDE


* ARCHITECTURE DEFINITION MEMBER ISPFSUB
CCODE D       INCLUDE


* ARCHITECTURE DEFINITION MEMBER PDF
CCODE B       INCLUDE


* ARCHITECTURE DEFINITION MEMBER SCLM
CCODE C       INCLUDE
```

When the ISPF architecture definition is built, only members with the change code A will be included from the build group. The CCODE statements to include change codes B, C, and D will not be processed for this build because they were found in included architecture definitions.

During the verification phase of build and promote, SCLM will search the change code list for members in the build or promote scope at the specified group. If the member is in scope and the change code appears (or does not appear in the case where EXCLUDE is specified) on the change code list, it will be included. Otherwise, SCLM will continue to search for the member beginning at the next group. Change codes will be processed for all editable members stored in PDS data sets under SCLM control, including architecture definitions. Change codes will be processed on included members when their data sets are allocated with IOTYPE=I, KEYREF=SINC. Included members whose data sets are allocated with a KEYREF of SREF or CREF will not be processed by change code. To process includes referenced by SREF and CREF allocations:

1. Add FLMINCLS macros to reference the desired types.
2. Change the FLMALLOC macros to use KEYREF=SINC.
3. Add an INCLS parameter to the FLMALLOC macros to reference the FLMINCLS macros.

The architecture definition specified as input to the build or promote will always be processed, regardless of its change codes. Change codes are only significant for the build or promote group. In scope members found above this group will be included regardless of change code. If the specified change appears on a member's change code list but is not the last change and INCLUDE is specified, a warning message will be issued.

We recommend you build and promote each change to a member before beginning another. In cases where this is not possible, multiple changes that affect a single member should be built or promoted together. For instance, assume that you have members A, B, and C. Change 1 affects members A and B while change 2 affects members A and C. As both changes affect member A, the inclusion of either change without the other will cause the changes to be unsynchronized. Change codes 1 and 2 should be built and promoted together.

To build an application containing dynamic includes by change code, a build without change codes must occur first. Otherwise, the build can fail because includes are missing.

A promote by change code must always be preceded by a successful build of the same architecture definition. At the completion of a promote by change code, rebuild the application at the higher group. Change codes are used to determine whether or not a member found at the report input group will be included in the Architecture Report when executing the Architecture Report Utility against an architecture definition containing CCODE statements. The Database Contents Utility, on the other hand, does not use change codes specified on CCODE statements to determine whether or not a member will appear in the report or tailored output.

Architecture Statements

You must use a special SCLM architecture language when you create architecture members. This language consists of statements that identify necessary information. The following paragraphs discuss the statements and their formats.

Statement Format

You must use a specific format for architecture members. Architecture definition data sets must be fixed block (FB) with a length of 80 bytes or characters. Only one statement can appear in each 80-byte record. A record ranges from columns 1 through 72, and the records cannot be continued. SCLM ignores information that appears after column 72.

Write the statements in either upper- or lowercase. You can write all statements, except for CMD, PARM, and PARMx statements, in a free format as long as the items within the statements are in the correct order. The number of blank spaces between each item is not significant (except in the CMD statement).

The order of statements is generally not significant. For example, you can place OBJ statements before or after SINC statements. The only statements for which the order is significant are those keywords that cause data to be concatenated into the input stream (INCL, INCLD, CMD and LINK for LEC architecture members; SINC and CMD for CC and generic architecture members); or into the translator options (PARM and PARMx).

Member and type names must follow MVS naming conventions. SCLM does not check parameters and control statements for validity. They can continue up to and including column 72.

All members explicitly referenced by an architecture statement MUST exist in the type specified in the architecture statement. However, SCLM uses extended types and include sets to resolve the parsed dependencies of members referenced by a SINC statement if necessary.

Statement Uses

SCLM distinguishes architecture members from one another by their content. SCLM assumes, for example, that a member containing both an OBJ statement and a SINC statement is a CC architecture member, and that a member containing a LOAD statement is an LEC architecture member.

Architecture statements provide information about the design of applications in the project database.

Table 19 shows valid statements for each type of member.

Table 19. Valid Keywords for Architecture Member Statements

HL	LEC	CC	Generic
*	*	*	*
CCODE	ALIAS	CCODE	CCODE
COPY	CCODE	CMD	CMD
INCL	CMD	COPY	COPY
INCLD	COPY	INCL	INCL
PROM	INCL (2)	INCLD	INCLD
	INCLD (2)	KREF	KREF
	KREF	LINK	LINK
	LINK (2)	LIST	LIST
	LKED	LKED	LKED
	LMAP	OBJ (1)	OUT*x*
	LOAD (1)	OUT*x*	PARM
	OUT*x*	PARM	PARM*x*
	PARM	PARM*x*	PROM
	PARM*x*	PROM	SINC(1)
	PROM	SINC(1)	SREF
	SINC	SREF	
	SREF		

1: Each of the following statements must be present in the architecture definition member:
- An LEC member must contain exactly one LOAD statement
- A CC member must contain exactly one OBJ statement and at least one SINC statement
- A Generic member must contain at least one SINC statement.

2: An LEC member must contain at least one of the following statements: INCL, INCLD, LINK, or SINC.

Each architecture statement is composed of a keyword followed by one or more operands. For those keywords that allow you to specify either a member name or an asterisk (*), specify an asterisk if you expect multiple outputs per DD statement. Otherwise, specify the member name if only a single output is expected. The following list shows the valid statements, their usage, and their format:

***** Identifies an architecture comment statement on a line by itself.

```
* <comment>
```

ALIAS Identifies load module aliases to be generated. Use it only in LEC architecture members. The type_name specified on the ALIAS statement must be the same as the type_name on the LOAD statement of the LEC architecture member.

```
ALIAS <member_name> <type_name> <optional_comment>
```

CCODE Identifies a change code to be included or excluded from a build or promote.

Any change code that contains an embedded blank or whose first character is other than A-Z, 0-9, @, # or $ must be enclosed in delimiters. A delimiter can be any character not specified above.

Valid values for the include flag are INCLUDE and EXCLUDE. The flag can be abbreviated but must be followed by a space. If no value is specified, the default is INCLUDE. Examples of valid flags are I, E, IN, EX, INCL, and EXCL.

```
CCODE change_code <optional_include_flag> <optional_comment>
```

CMD Identifies command statements to be included with inputs to the compiler, linkage editor, or other processors. The statement is positional; therefore, all blanks following this statement starting after the first blank are significant. Do not include the optional_comment with the CMD statement because it will be part of the control statement. The CMD statement is not valid in HL architecture members.

```
CMD <control_statement>
CMD PARMS /Ss /DIPF
CMD ACTION IPFCP
```

The FLMLTWST translator reads the build map for ACTION and PARMS control statements. ACTION may be used for additional workstation commands. PARMS may be used to identify strings to be added to the workstation command. These control statements are different than the ACTION and PARMS keywords that may be used in the OPTIONS list for FLMLTWST. The PARMS value in the OPTIONS list is added to all workstation commands whereas the string following the PARMS control statment in the build map is appended to the workstation command being created at that time. See the "SCLM Reference" book for additional information.

Note:

CMD statements in an architecture definition will be placed in the build map with the control statement. The control statement will only be passed to the build translator in the controlling language definition if there is also an FLMALLOC macro with IOTYPE=S. Translators used for

workstation build may read the control statement from the build map to create a workstation command.

COPY Identifies another architecture member to be inserted into this architecture member.

The COPY statement of the architecture language provides you with the ability to simplify related, complex architecture members. To simplify architecture members with similar contents, use the COPY statement to isolate identical statements into a separate member and reference the member. Referenced members must follow all formatting rules for architecture members.

The COPY statement results in a direct insert of the contents of the specified member into the respective architecture members. Therefore, using a copy architecture member is an efficient way to group sets of commonly used architecture statements into a single area. Additions to and deletions from the common architecture member affect all the architecture members referencing the member.

```
COPY <member_name> <type_name> <optional_comment>
```

Note: Use the COPY statement rather than the INCL statement (see the following description) when the specified member cannot be processed independently from the architecture definition in which it appears.

INCL Identifies another architecture member that this architecture member references. The referenced architecture member will be processed prior to this architecture member.

Additionally, if INCL is used in an LEC architecture member, the output from the INCL is used to create the load module for the LEC.

Only CC and LEC architecture members should be referenced by an INCL statement in another LEC architecture member. For CC architecture members, the output referenced by the OBJ keyword is used to create the load module; for LEC architecture members, the output referenced by the LOAD is used.

```
INCL <member_name> <type_name> <optional_comment>
```

Note: Use the INCL statement rather than the COPY statement (see the previous description) when the specified member can be processed independently from the architecture definition in which it appears.

INCLD Identifies a source member that this architecture member references. The referenced member will be processed prior to this architecture member.

Additionally, if INCLD is used in an LEC architecture member, the output from the INCLD is used to create the load module for the LEC. The language definition for the member referenced by the INCLD statement must have a build output with KEYREF=OBJ.

```
INCLD <member_name> <type_name> <optional_comment>
```

KREF Identifies the output keywords from other members that will become inputs to the member containing the KREF statement. The

keywords identified by the KREF statement must be architecture statements that identify outputs of a build. Examples are OBJ, LOAD and OUT1. Only those outputs of members referenced by INCL or INCLD statements in the architecture member containing the KREF statement will be considered for inclusion.

If the KREF statement is omitted, the outputs that are included depend on the type of architecture definition. For LEC architecture definitions, the default is to include OBJ and LOAD outputs. For all other types of architecture definitions, the default is not to include any outputs produced by referenced members.

If a KREF statement is specified in an LEC architecture definition, the defaults of OBJ and LOAD will be lost. To include another output type in addition to OBJ and LOAD, three KREF statements must be specified: one for OBJ, one for LOAD, and one for the additional output type (OUT1 for example).

Valid reference keywords are: COMP, LIST, LMAP, LOAD, OBJ, and OUTx.

```
KREF <reference_keyword>
```

Note: Although multiple KREF statements can be coded in a single LEC architecture member, duplicate KREF statements will result in an error.

LINK Identifies an output that must be produced prior to this ARCHDEF being processed. The build function only verifies the contents of the output referenced if extended scope is specified. You can substitute the INCL statement to cause this verification to always be performed.

Additionally, if LINK is used in an LEC architecture member, the output referenced is used to create the load module for the LEC.

```
LINK <member_name> <type_name> <optional_comment>
```

LIST Identifies the member(s) and type in which the compiler listing is to reside. The LIST statement is not valid in HL or LEC architecture members.

```
LIST <member_name | *> <type_name> <optional_comment>
```

LKED Identifies the language to be used to process the contents of the architecture member.

Language_id is an 8-character language identifier for a translator. The language ID specified must correspond to a valid language identifier defined in the project definition.

If the LKED keyword is omitted, SCLM uses the default language to process the architecture member. For LEC architecture members the default language is LE370. For CC and Generic architecture members the default language is the language of the member on the first SINC statement.

```
LKED <language_id> <optional_comment>
```

LMAP Identifies the member(s) and type in which the linkage editor listing (load map) is to reside. Use it only in LEC architecture members.

```
LMAP <member_name | *> <type_name> <optional_comment>
```

LOAD Identifies the load module(s) to be created and the type in which the load modules(s) reside. Use it only in LEC architecture members.

```
LOAD <member_name> <type_name> <optional_comment>
```

OBJ Identifies the name of the object module(s) to be created and the type in which the module(s) reside. Use it only in CC architecture members.

```
OBJ <member_name | *> <type_name> <optional_comment>
```

OUT*x* Identifies the output member(s) to be created and the type in which the member(s) reside. Replace the *x* with an integer to identify the specific statement. Valid integer replacements are 0 through 9. You can use these statements to track additional outputs other than the standard outputs described by the statements OBJ, COMP, LIST, LOAD, and LMAP. Use the OUT*x* statement in an LEC, CC, or generic architecture member.

```
OUTx <member_name | *> <type_name> <optional_comment>
```

PARM Identifies parameters (options) to be passed to all build translators of a compiler, linkage editor, or other processor. Use it in generic, CC, or LEC architecture members. Do not use this keyword to pass parameters to non-build translators such as VERIFY, PURGE, and COPY.

SCLM offers a set of variables that you can use to dynamically provide information to compilers, linkage editors, and other processors. Use these variables with the PARM statement.

Do not use the optional_comment with the PARM statement because it will be passed to the build translators.

```
PARM <parameters>
```

PARM*x* Identifies parameters (options) to be passed to build translators of an SCLM language. Replace the *x* with an integer to identify the specific statement. Valid integer replacements are 0 through 9. You can use the SCLM variables, mentioned previously, with the PARM*x* statement. You can use the PARM*x* statement in generic, CC, and LEC architecture members. Do not use this keyword to pass parameters to non-build translators such as VERIFY, PURGE, and COPY.

Do not use the optional_comment with the PARM*x* statement because it will be passed to the build translators.

If the PARM*x* keyword used in the architecture member is not specified in one of the FLMTRNSL macros (using the PARMKWD parameter), SCLM ignores the PARM*x* statement.

```
PARMx <parameters>
```

Notes:

1. The complete options list passed to the build translator has a maximum length of 512 characters and has the following format:

```
 string1
,string2
,string3
```

where

string1
contains the options from the OPTIONS parameter on the FLMTRNSL macro.

string2
contains the options from the PARM statements in the architecture definition. No commas are inserted between PARM statements.

string3
contains the options from the PARMx statements in the architecture definition. Commas are inserted between PARMx statements.

Leading and trailing blanks are removed by SCLM.

For example, suppose that the FLMTRNSL macro specifies that the following options are to be passed to a translator:

```
OPTIONS=(NOXREF)
```

Further suppose that there is an architecture definition for the translator with the following parameters defined:

```
PARM  PARAMETER1
PARM  PARAMETER2
PARM  PARAMETER3
PARM1 PARAMETER4
PARM2 PARAMETER5
PARM3 PARAMETER6
```

The options passed to the translator would look like this:

```
NOXREF,PARAMETER1PARAMETER2PARAMETER3,PARAMETER4,PARAMETER5,PARAMETER6
```

2. Parameters specified on the PARM and PARMx statements in an LEC architecture member are passed to the linkage edit translator but not to any of the compilations needed to produce object or load modules for the linkage edit operation.
3. You should review the documentation of each build translator for unique handling requirements of passed parameters (for example, case and handling of special characters).

PROM Identifies a text member, such as design, data, or test plans, to be promoted along with the modules processed in this architecture member. The member specified is not processed by build (for example, compiled or linked) but is tracked during promotions. You can specify an additional parameter to indicate whether date checking is to be performed for the member.

Date_check is a special optional parameter for the PROM statement to bypass date checking for noncompilable/nonlinkable members. A nonblank, such as N, as a third parameter on the PROM statement indicates to the build and promote functions to bypass date checking for that member (thereby eliminating the need to build before promoting) when you modify the member.

Note: Do not use the optional_comment with the PROM statement because it can cause build and promote to bypass date checking.

```
PROM <member_name> <type_name> <date_check>
```

SINC When used in generic and CC architecture members, the SINC statement identifies the source member. When used in an LEC architecture member, the SINC statement identifies the member or group of members to pass to the linkage edit translator. Use it only in generic, CC, and LEC architecture members.

```
SINC <member_name> <type_name> <optional_comment>
```

You can specify multiple SINC statements in an architecture definition. SCLM copies each statement, in the order they appear, into the temporary file allocated with FLMALLOC IOTYPE=S.

Notes:

1. The input list feature of the Build function is designed to work with direct translations of source members only (source members referenced with an INCLD statement). Using the input list feature with source members controlled by CC or Generic architecture definitions produces undefined results (source members referenced with a SINC statement). For more information on Input List languages and translators, see Part Two of this book.
2. If there is a SINC statement, but no FLMALLOC with IOTYPE=S, in the language definition for the language of the member referenced by the SINC statement, the referenced member is not placed on the SYSIN input stream for the build.

SREF Identifies a type to be allocated during processing. Specifically, use the SREF keyword to allocate a specific type for translators. You can use it in generic, CC, and LEC architecture members.

SREF is a function that identifies an additional type to be allocated during processing. Do not use this function unless you have extremely complex hierarchical concatenation needs.

```
SREF <type_name> <optional_comment>
```

Sample Application Using Architecture Definitions

The following application is composed of two subapplications. Each subapplication consists of two load modules, that are composed of a series of object modules. Load module FLM01LD1 and FLM01LD2 contain one object module each, while FLM01LD3 and FLM01LD4 contain multiple object modules. Figure 93 on page 262 shows a diagram of the design of this application (FLM01AP1) and Figure 94 on page 263 shows the architecture members for the FLM01AP1 application.

Note: SCLM tracks the included members; therefore, there is no need to mention FLM01EQU in the architecture definition.

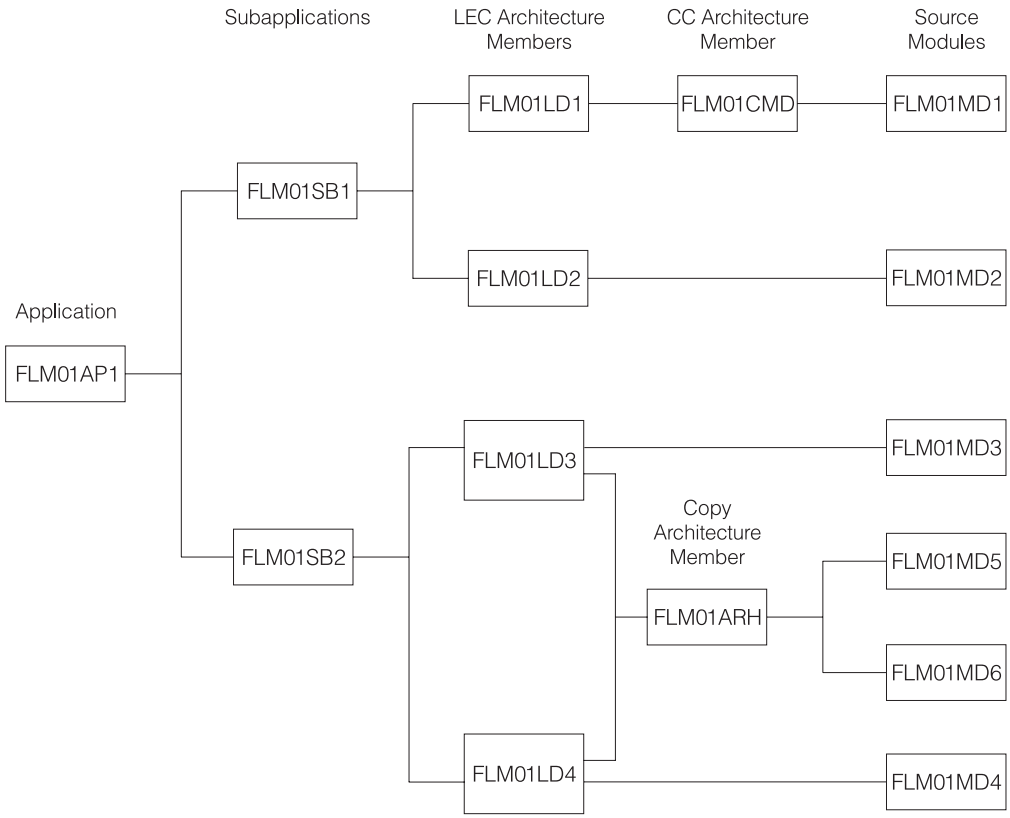


Figure 93. Application FLM01AP1

High-Level Architecture Members

FLM01AP1

```
*
*      Application FLM01AP1
*
INCL    FLM01SB1    ARCHDEF
INCL    FLM01SB2    ARCHDEF
```

FLM01SB1

```
*
*      Subapplication 1
*
INCL    FLM01LD1    ARCHDEF
INCL    FLM01LD2    ARCHDEF
```

FLM01SB2

```
*
*      Subapplication 2
*
INCL    FLM01LD3    ARCHDEF
INCL    FLM01LD4    ARCHDEF
```

Linkedit Control Architecture Members

FLM01LD1

```
*
*      Load Module FLM01LD1
*
LOAD    FLM01LD1    LOAD
LMAP    FLM01LD1    LMAP
INCL    FLM01CMD    ARCHDEF
PARM    MAP, NCAL,
PARM    LET
```

FLM01LD2

```
*
*      Load Module FLM01LD2
*
LOAD    FLM01LD2    LOAD
LMAP    FLM01LD2    LMAP
INCLD   FLM01MD2    SOURCE
```

FLM01LD3

```
*
*      Load Module FLM01LD3
*
LOAD    FLM01LD3    LOAD
LMAP    FLM01LD3    LMAP
COPY    FLM01ARH    ARCHDEF
INCLD   FLM01LD3    SOURCE
PARM    MAP
```

FLM01LD4

```
*
*      Load Module FLM01LD4
*
LOAD    FLM01LD4    LOAD
LMAP    FLM01LD4    LMAP
COPY    FLM01ARH    ARCHDEF
INCLD   FLM01MD4    SOURCE
PARM    MAP
```

Figure 94. Architecture Members for Application Sample FLM01AP1 (Part 1 of 2)

Compilation Control Architecture Members

FLM01MD1

```
*
*      Object Module FLM01MD1
*
OBJ     FLM01MD1   OBJ
LIST    FLM01MD1   LIST
SINC    FLM01MD1   SOURCE
PARM    NOXREF, LC(75)
```

Copy Architecture Members

FLM01ARH

```
*
*      COPY ARCHITECTURE
*
INCLD  FLM01MD5   SOURCE
INCLD  FLM01MD6   SOURCE
```

Figure 94. Architecture Members for Application Sample FLM01AP1 (Part 2 of 2)

The HL architecture member in part 1 of Figure 94 includes references to two subapplications: (FLM01SB1 and FLM01SB2). The subapplication HL architecture members reference the LEC architecture members that define the load modules they contain. Note that the referenced LEC architecture members have the same names as the load modules they produce.

The LEC architecture members contain all the information necessary to produce the load modules in the application. Two PARM statements in FLM01LD1 override the default linkage editor options.

Load modules FLM01LD3 and FLM01LD4 contain copy statements. These statements identify the architecture member FLM01ARH, that references two source modules for SCLM to insert into the FLM01LD3 and FLM01LD4 load modules.

Thus, copy architecture members are an efficient technique for grouping commonly used architecture statements into a single member. Additions to and deletions from FLM01ARH affect FLM01LD3 and FLM01LD4 and all the other architecture members that might reference FLM01ARH.

Ensuring Synchronization with Architecture Definitions

SCLM ensures that all modules within the scope of a build are synchronized. If you build a source module, SCLM synchronizes the resulting object and listing with the source. If you build an architecture definition, SCLM synchronizes all members used as input to the builds and all members output from the builds. However, if there are object or load modules outside the scope of a particular build that are dependent on source modules within the scope of that build, those source, object, and load modules might no longer be synchronized.

In the following example, object modules OBJ1, OBJ2 and OBJ3 are produced by compiling source modules SOURCE1, SOURCE2 and SOURCE3, respectively.

SOURCE2 might be the source module for an I/O routine many applications use. Load module LOAD1 is the result of linking OBJ1 and OBJ2, while LOAD2 results from the link edit of OBJ2 and OBJ3. LOAD1 and LOAD2 might be two separate programs that run against the same kind of data and would therefore need to have a common I/O routine (SOURCE2). FLM01AP1 and FLM01AP2 are LEC architecture definitions that describe how to link edit LOAD1 and LOAD2, respectively. Finally, TOPARCH is a high-level architecture definition that includes FLM01AP1 and FLM01AP2.

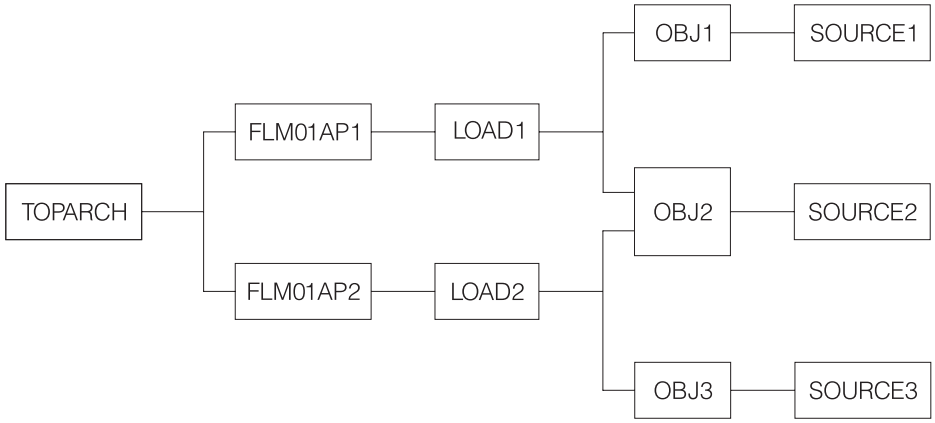


Figure 95. Example of Synchronization

In Figure 95, all of the modules shown in the diagram exist only in the production layer of your SCLM-controlled hierarchy and all source, object and load modules are synchronized. In other words, for each load module, the hierarchy contains the exact version of the object modules that were used to link edit that load module. For each object module, the hierarchy contains the exact version of the source that was compiled to create that object module. You can always recreate exactly (except for time stamps) the object and load modules for the applications.

With this structure, you must pay close attention to which architecture definitions you use to build and promote development changes. The following scenario describes the INCORRECT use of architecture definitions, which leads to a loss of synchronization between source and load.

A user puts in a request for a change to LOAD1 and you decide that the way to implement that change is to modify SOURCE2. Because you are making a change to LOAD1, you also decide (in error as it will turn out) to use FLM01AP1 to drive your builds and promotes. When your changes are made and you are ready to build, you cause SCLM to rebuild OBJ2 (because SOURCE2 changed) and LOAD1 (because OBJ2 changed), by specifying FLM01AP1 on the Build panel. LOAD2 is *not* rebuilt, even though OBJ2 changed, because LOAD2 is outside of the scope of architecture definition FLM01AP1. Herein lies the problem. When you promote FLM01AP1, SCLM checks that everything that needs to be rebuilt (within the scope of FLM01AP1) has been rebuilt. Unfortunately, modules outside the scope of FLM01AP1 should be rebuilt as well.

When complete, all modules within the scope of FLM01AP1 are synchronized and recreatable. However, LOAD2 was outside the scope of the architecture definition you used and is not recreatable. Therefore LOAD2 is not synchronized with its source.

To avoid this problem, you must analyze the architecture of the applications in your SCLM-controlled project and choose an architecture definition with a scope that contains all modules that need to be rebuilt. The correct architecture definition would have been TOPARCH in the example because only TOPARCH has both LOAD1 and LOAD2 within its scope. These modules have to be relinked because of a change to SOURCE2.

It is strongly suggested that you have one high-level architecture definition with a scope that includes every module controlled by an SCLM project. You can use architecture definitions with much smaller scopes in your day-to-day development work. However, if you do that, you should also check the synchronization of all modules in the project by performing a build on the top high-level architecture definition as part of your testing.

Build Outputs

Several architecture definition statements are used to identify the outputs of a build. These statements are: ALIAS, COMP, LIST, LMAP, LOAD, OBJ, and OUTx. These statements have two parameters. The first is the member name of the output and the second is the type name of the output. The type name parameter must be a type name from the project definition. The member name parameter can be either a valid PDS member name or an ″*″. A PDS member name can be used when there is a single output with a predefined member name. PDS member names must be used for the ALIAS and LOAD architecture statements. An ″*″ must be used if there are multiple outputs or the output member name is not predefined.

Build allocates temporary data sets to hold the outputs generated by the build translators. If all the translators complete successfully the outputs from the temporary data sets are copied into the SCLM hierarchy. Since the copy does not take place until all translators have completed the allocation of the output data sets must be retained without over writing the output until after the last translator runs.

Multiple Build Outputs

Multiple output members may be generated for a single output keyword if the IOTYPE on the FLMALLOC for the translator output is ″P″. This allows the translator to store multiple members into a PDS data set. When a PDS member name is specified on the output architecture statement SCLM will copy a member with that name from the temporary data set into the SCLM hierarchy. The member name in the temporary data set must match the SCLM member name. When an ″*″ is specified in the member name parameter then SCLM will copy all outputs in the temporary data sets without changing the member names.

Sequential Build Outputs

A single build output may be generated into a sequential data set by using an FLMALLOC with IOTYPE=O. When the output architecture statement indicates a member name the output will be copied to an SCLM member of that name. When an ″*″ is specified for the output member the member name will be the name of the architecture definition.

Default Output Member Names

When a source member is built directly, either as the input member to the build or by an INCLD statement, the output member name is determined from information

in the project definition or by SCLM defaults. If the FLMALLOC statement for the output specifies a default member name using the DFLTMEM parameter then that member name will be used. When no default member name is specified, the output member name will be the same as the source member. Use an architecture definition when generating multiple outputs to be stored in a partitioned data set. See the previous description of ″Multiple Build Outputs″.

Languages of Output Members

SCLM gets the language of the output member from one of two locations. The first place SCLM looks is on the FLMALLOC statement in the project definition for a LANG parameter. If it is found then it is used as the language of the output member. When no LANG parameter is found and a source member is being built the language of the source member is used as the language of the output member. If an architecture definition is being built and no LANG parameter was found, then the language used to build the architecture definition is used as the language of the output member.

Chapter 12. Managing Complex Projects

This chapter describes additional SCLM features that you can use to define and manage complex projects. Topics discussed in this chapter include:

- Impact assessment techniques
- Dependency processing implementation
- Propagating applications to other databases.

Impact Assessment Techniques

Making updates to a component of an application without full knowledge of their effect on the application can cause a large number of unexpected recompilations. Impact assessment is a technique you can use to assess the impacts of updates to an application *before* they occur. It allows developers to determine what effect changing a given component of the application has on the rest of the application or a given subapplication. Impact assessment enables you to avoid time-consuming recompilations.

Follow the procedure below to use SCLM Build to create an impact assessment:

1. Use the SCLM editor to save the members you want to change
   a. in an empty development group or
   b. save them with a change code.
2. Invoke the build function using the report mode on the top architecture definition for the application affected. If you saved with a change code, create a new top architecture definition that includes the old top architecture definition and uses the CCODE keyword to include the change.
3. Examine the resulting build report. This report reflects all output that regenerates when the build is performed. The build messages data set indicates which translators are invoked.
4. If the results are acceptable, you can proceed with your planned changes. Otherwise delete the members you saved in Step 1 using the SCLM Library utility or the Delete group utility.

You can perform a second method of assessing impacts by using an SCLM architecture report. Examine this report for the members that the developer wants to modify. Starting with the members to be modified, you can identify all architecture members that control the modified members. While this technique is more meticulous than the first, it does not require that the member be drawn down, modified, and built.

Either of the preceding techniques help identify costly recompilation impacts.

Dependency Processing

This section explains how SCLM handles include dependencies. If SCLM does not provide a sample for a language you want to support, use this information to map the language dependencies to SCLM dependencies.

SCLM derives dependency information when a member is parsed. This information is stored as SCLM control data, and it allows SCLM to perform the following functions:

- Process members in the correct order
- Determine when members are out-of-date (changed) and need to be rebuilt
- Determine the scope for functions such as build and promote.

The following describes the processing involved for each include dependency.

A member is included if it is required for completion of a compile of the member that references it. Examples are members referenced by the %INCLUDE directive in Pascal, the COPY operand in Assembler, the COPY command in Cobol, and the imbed (.im) in Script. Assembler macros are also considered to be includes because they must be expanded when the referencing member is assembled.

The primary input to the compiler defines the SCLM controlled data sets to search for includes. The primary input to the compiler is referenced directly on the build panel or via the SINC or INCLD architecture definition keywords in SCLM. If more than one SINC keyword is used in an architecture definition, the primary input is the member referenced by the first SINC.

Any member can have include dependencies. SCLM recursively searches for included members beginning with the primary input to find all of the dependencies that are needed for the compilation.

The language of the primary input defines which types are searched to find includes. The FLMINCLS macro is used to specify which types are searched and the order in which they are searched. For more information on how includes are found, see Part Two of this book.

Included members can be editable or non-editable.

Included members must exist and have valid accounting information when the member that references them is built. Build does not attempt to compile members that have missing include dependencies.

Build rebuilds the primary input member if any of its recursive includes have changed since it was last built.

Propagating Applications to Other Databases

You can use EXPORT or IMPORT to propagate systems by moving code from a development group to a production group.

You can also use the EXPORT and IMPORT utilities to backup and restore data from an SCLM hierarchy. The steps necessary to backup and restore the project database are listed as follows:

1. Export the group to be backed up using the EXPORT service.
2. Save the member text in a PDS for later recovery if necessary.
3. To restore the data, create an alternate definition that specifies a new temporary development group into which you will import the previously exported data.
4. Specify the export data sets to be restored on the FLMCNTRL macro.
5. Copy the saved member text for the backed up group to the new temporary group.
6. Invoke the IMPORT service and specify the new temporary group. Note that after the IMPORT service has completed, the new group contains the same data that was originally exported.

7. If you use the new group, use the DELGROUP service to purge the data in the original group, delete the original data sets, and rename the temporary group to the original group name. Another way of accomplishing the same goal is to delete the accounting data out of the original group and then import directly into it.

Note: The IMPORT service erases the exported data after it successfully imports members. Therefore, you may want to make a copy of the export data sets before invoking the IMPORT service if you want to preserve the backup version of the data sets.

Part 3. DB2 and Workstation Support

Chapter 13. SCLM Support for DB2, General Information

In SCLM, you can have applications that support DATABASE 2 (DB2) processing. Before you can use SCLM with DB2, the DB2 system must be installed and fully operational; otherwise, SCLM cannot interact with it correctly.

In your SCLM project, you must create a DB2 CLIST for each DB2 application plan. The DB2 CLIST must specify the Data Base Request Modules (DBRMs) to be bound into the DB2 application plan. These DBRMs are created by the DB2 preprocessor defined in the appropriate language definitions. Because the DB2 CLIST is controlled by SCLM, it contains accounting information and can be built. This produces build maps. The DB2 CLIST can be referenced from architecture definitions.

The processing of a DB2 CLIST in SCLM has the following stages:

1. During the Editing stage, you must create a DB2 CLIST as described in "Create DB2 CLIST" on page 278. When parsed, the DBRMs to be bound are identified and an entry is placed in the accounting information for the DB2 CLIST.
2. During the Build stage, the DB2 CLIST member is executed to perform the appropriate Bind or Free DB2 operation. An identical copy of the DB2 CLIST is created and placed in the type that is used during the Promote stage. You can browse this new DB2 CLIST but you cannot edit it. SCLM does not allow build outputs to be edited. The new DB2 CLIST is an output of a build process, and SCLM treats all outputs as noneditable.

   The difference between the original DB2 CLIST and the new DB2 CLIST is the language value. The language for the original DB2 CLIST is associated with a language definition that contains the parsing and build translators; the language for the new DB2 CLIST is associated with a language definition that contains the copy and purge translators.
3. During the Promote stage, the DB2 CLIST that was created during the Build process is executed to perform the Copy and the Purge phases of the Promote stage.

In your architecture definitions, always refer to the DB2 CLIST used during the Build stage; do not refer to the DB2 CLIST used during the Promote stage.

Note: When promoting a DB2 CLIST, the members that generated the DBRMs referenced by the DB2 CLIST are also promoted.

Restrictions

The included members that are processed by the DB2 precompiler must reside in the SCLM source library or its extended library for SCLM to track them as included dependencies. Otherwise, the library should be added to the FLMSYSLB macro in the language definitions to prevent SCLM from creating an Include dependency. Additionally, ALCSYSLB=Y should be specified for the language definition, or an FLMCPYLB with the appropriate library specified should be added into the FLMALLOC that has DDNAME=SYSLIB in the COBOL compiler step.

The parser determines the SQL include dependencies by parsing the EXEC SQL INCLUDE statements. Some of the SCLM parsers check for SQL includes.

Refer to the SCLM Translators section of the *SCLM Reference* for more information.

Information For The Project Manager

Generating a Project Environment

Chapter 1. Defining the Project Environment describes the steps to set up and maintain an SCLM project database. For DB2 support, additional considerations within these steps must be performed. This section describes these considerations step-by-step.

Step 1: Determine the Project's Hierarchy

There are no additional considerations.

Step 2: Identify the Types of Data to be Supported

If you are already running an existing SCLM project that has all the data types described in Chapter 1. Defining the Project Environment, additional types must be created. The following types of data must be maintained and are the recommended naming conventions:

- DBRM

  Contains the source member input to a DB2 BIND. It is generated by the DB2 preprocessing step.

- DB2CLIST

  A DB2 CLIST that contains editable source members. These source members are used during SCLM Build to control Bind/Free functions for DB2.

  To have DB2 CLIST members and DBRM members with the same name, an FLMINCLS macro needs to be specified in the language definition for the DB2 CLIST members. The FLMINCLS macro must list the DBRM type first on the TYPES parameter. An example of an FLMINCLS macro to do this follows:

  ```
  *
  * SPECIFY TYPES TO SEARCH FOR DBRMS THAT ARE TRACKED AS
  * INCLUDES TO THE DB2 CLIST MEMBERS
  *
            FLMINCLS TYPES=(DBRM)
  ```

- DB2OUT

  This type contains non-editable build output used during SCLM Promote to control Bind and Free functions for DB2. During a build of a DB2 CLIST (of type DB2CLIST), a copy of the DB2 CLIST is copied in the type DB2OUT into the group that is being built. During a promote, this member is called to bind the plan in the TO group and free the plan in the FROM group.

Step 3: Establish Authorization Codes

There are no additional considerations.

Step 4: Allocate the PROJDEFS Data Sets

The data set characteristics for the new types are described in Table 20.

Table 20. SCLM Data Set Attributes for DB2 Types

Type	PS or PO	RECFM	LRECL	BLKSIZE.
DBRM	PO	FB	80	3120
DB2CLIST	PO	FB	80	3120
DB2OUT	PO	FB	80	3120

You can browse the example project definition, FLM@EXM2, which provides an example of the macros used to support DB2.

Step 5: Allocate the Project Partitioned Data Sets
There are no additional considerations.

Step 6: Allocate and Create the Control Data Sets
There are no additional considerations.

Step 7: Protect the Project Environment
There are no additional considerations.

Step 8: Create the Project Definition
Specify additional types to be supported with the FLMTYPE macro.

SCLM provides many language definitions as examples. The examples serve as a guide in the construction of language definitions for specific applications and environments. Use the COPY macro to include any of the following sample definitions that apply to your DB2 environment:

Table 21. Language Definitions for DB2

Member	Language	Description.
FLM@BD2	DB2CLIST	DB2 BIND/FREE
FLM@BDO	DB2OUT	DB2 BIND/FREE output
FLM@2ASM		DB2 preprocessing + Assembler
FLM@2CO2		DB2 preprocessing + COBOL II
FLM@2C		DB2 preprocessing + C/370
FLM@2FRT		DB2 preprocessing + FORTRAN
FLM@2COB		OS COBOL with DB2
FLM@2PLO		PL/I OPTIMIZER with DB2
FLM@EASM		ASSEMBLER F with CICS V3R2M1 and DB2
FLM@ECOB		OS COBOL with DB2 and CICS
FLM@ECO2		COBOL II with DB2 and CICS
FLM@EC		C/370 with DB2 and CICS
FLM@EPLO		PL/I OPTIMIZER with DB2 and CICS

Define the Language Definitions: If you have a different naming convention for the types or languages, you need to do the following:

- Modify the DFLTTYP and LANG values on the FLMALLOC macros to reflect your naming conventions.
- Modify the DBRMTYPE values in the OPTIONS parameter on the FLMTRNSL macros in the language definitions to reflect your naming conventions.

Step 9: Assemble and Link the Project Definition
There are no additional considerations.

Information For The Developer

Developer Recommendations

- To use multiple environments with DB2, use the naming conventions so that you can distinguish between the DBRMs for different environments. For example, use a type named MTDBRM to denote MVS/TSO and a type MCDBRM for MVS/CICS.
- You can look at the names of included DBRMs for a DB2 CLIST by browsing its accounting information:
  1. Select the Utilities option from the SCLM Main Menu.
  2. Select the Library option from the SCLM Utilities Menu.
  3. From the SCLM Library Utility - Entry Panel, enter the DB2 type to be used during Build.
  4. From the list of members, select the DB2 CLIST that you want to examine and browse its accounting information.
  5. From the Accounting Record for the DB2 CLIST, select the Number of Includes.
  6. Finally, you see the list of included DBRMs in the DB2 CLIST.

Getting Started

Create DB2 CLIST

You must create a DB2 CLIST member for each DB2 application plan. The DB2 CLIST is a TSO CLIST that allows you to BIND or FREE the DB2 application. This CLIST should contain code to perform the following functions:

- Allow different DB2 Subsystem names to be assigned to each group
- BIND the application plan
- FREE the application plan.

You can see the parameters and logic required in Figure 96 on page 279.

The DB2 CLIST member allows you to specify which DBRMs are bound into the application plan. The DB2 CLIST member is editable.

The DB2 CLIST member must have an include statement for each DBRM to be bound in the application plan. The include statement consists of an included directive and the name of the included DBRM. SCLM parses the member and keeps a list of included DBRM names, as well as other accounting information. The include directive and include DBRM name must be on the same line. The include statement format is:

```
/*  %INCLUDE dbrm-name   */
```

The DB2 CLIST is usually built and promoted by using an architecture definition. Use the SINC or INCLD keyword to reference the member from an architecture definition. The member can also be submitted directly to build or promote. When the member is submitted directly or is submitted through an INCLD architecture definition keyword, SCLM uses the defaults defined in the member language definition.

```
PROC 0 OPTION() GROUP()
  CONTROL MSG FLUSH
  /*--------------------------------------------------------------------*/
  /*  DBRM PROXY DSN CLIST for a DB2 Application Plan                   */
  /*                                                                    */
  /* INPUT PARAMETERS:                                                  */
  /*   OPTION()         BIND OR FREE                                    */
  /*   GROUP()          GROUP NAME FOR BIND OR FREE                     */
  /*                                                                    */
  /* RETURN CODES:                                                      */
  /*                 0 : SUCCESS                                        */
  /*                 4 : WARNING                                        */
  /*                 8 : ERROR                                          */
  /*                16 : FATAL ERROR                                    */
  /*               312 : INVALID GROUP                                  */
  /*               316 : INVALID OPTION                                 */
  /*                                                                    */
  /*--------------------------------------------------------------------*/
  /* INSTRUCTIONS FOR CUSTOMIZATION:                                    */
  /*                                                                    */
  /* 1) CHANGE THE ------- NAMES FOR YOUR DBRM MODULES.                 */
  /* 2) SPECIFY VARIABLES:                                              */
  /*    PLAN NAME (&PLAN -CHANGE PLANDEV, ETC...) FOR EACH GROUP        */
  /*    SUBSYSTEM (&SYS  -CHANGE DB2C) FOR EACH GROUP                   */
  /* 3) USE THE SCLM GROUPS (DEV1, DEV2, ETC...) ACCORDING TO           */
  /*    YOUR PROJECT.                                                   */
  /*                                                                    */
  /*--------------------------------------------------------------------*/
  /* SPECIFY AN INCLUDE FOR EACH DBRM TO BE INCLUDED IN THE             */
  /* DB2 APPLICATION PLAN                                               */
  /*                                                                    */
  /*   %INCLUDE   dbrm-name                                             */
  /*--------------------------------------------------------------------*/
  SET &RCODE = 0
  /*--------------------------------------------------------------------*/
  /* SPECIFY THE BIND MEMBER LIST IN &DBRMS                             */
  /*--------------------------------------------------------------------*/
  SET &DBRMS = &STR(dbrm-name)
  /*--------------------------------------------------------------------*/
  /* SPECIFY PLAN NAME, BIND PARMS, AND SYSTEM FOR EACH GROUP           */
  /*                                                                    */
  /* Note that the different bind parameters could be used at           */
  /* different groups.                                                  */
  /*                                                                    */
  /*--------------------------------------------------------------------*/
  SELECT (&GROUP)
    WHEN (group-name) DO
                      SET &PLAN = plan-name
                      SET &SYS  = system-name
                      SET &BPARM = &STR(FLAG(I) EXPLAIN(NO) +
                              VALIDATE(BIND) ISOLATION(CS)  )
                    END
    OTHERWISE       DO
                      SET &RCODE = 312
                    END
  END
```

Figure 96. DB2 CLIST Generic Example (Part 1 of 2)

```
/*-------------------------------------------------------------------*/
/* INVOKE DSN COMMAND PROCESSOR TO BIND OR FREE                      */
/*-------------------------------------------------------------------*/
SET &ENDDSN = END
IF &RCODE = 0 THEN +
  DO
    SELECT (&OPTION)
      WHEN (BIND) DO
                    DSN SYSTEM(&SYS)
                      BIND PLAN(&PLAN) MEMBER(&DBRMS) &BPARM;
                      &ENDDSN;
                    SET &RCODE = &MAXCC;
                  END
      WHEN (FREE) DO
                    DSN SYSTEM(&SYS)
                        FREE PLAN(&PLAN)
                        &ENDDSN;
                      SET &RCODE = &MAXCC;
                  END
      OTHERWISE   DO
                    SET &RCODE = 316
                  END
    END
  END
EXIT CODE(&RCODE)
```

Figure 96. DB2 CLIST Generic Example (Part 2 of 2)

Chapter 14. SCLM Support for Workstation Builds

You can store the source for workstation applications in SCLM. You can then use the configuration functions to build and promote the application. The build function transfers the source to an ISPF connected workstation, runs the compiler or other workstation tool, and then stores the results back into SCLM.

Storing workstation applications in SCLM provides several benefits:

- You can use SCLM as a single point of access for the workstation code.
- You can protect and back up the application source, executables, and outputs using the host.
- Host applications and workstation applications can share source.
- You can use SCLM's configuration management to ensure that the application is current.
- You can use the library management and versioning capabilities to track the application parts through the hierarchy and to retain backup versions.

Requirements

Because of the differences in MVS and the workstation operating system, you must meet the following requirements for SCLM to store the application source:

- The file names must follow ISPF member naming conventions and cannot be more than 8 characters. Workstation file names can be in uppercase, lowercase, or have initial capital followed by lowercase letters. This mapping is specified using the **WSCASE** keyword in the ACTINFO file.
- Use consistent naming conventions for the extension names and subdirectory layout. The workstation build translator provided with SCLM (FLMLTWST) maps type names to extensions and subdirectories. Consistent use of the extension and subdirectory names across the workstations that you use will make sure that the mapping will work properly.
- Use consistent command names. The commands are defined by input data to the FLMLTWST translator.

Overview of Workstation Build

The only distinction that SCLM makes between a workstation application and a host application is where the compiler and other tools reside. The application source and the outputs from builds are stored in PDS data sets on the host. The result is that all of the SCLM functions work the same for a workstation application as they do for a host MVS application except for build.

The difference between building a workstation application and a host application is that special build translators are used for the workstation application. The user doing the workstation build must use a workstation.

SCLM provides three build translators to build workstation applications. One translator, FLMLTWST, is the driver and calls the other translators to perform various tasks. In order to allow customization of the events that take place during a workstation build, the FLMLTWST translator is written in REXX. This allows the translator to be customized to meet the project's needs. The FLMLTWST translator performs the following tasks:

- Initialization and set up

  SCLM checks the parameters, retrieves and checks the workstation information, sets up file name mapping information, and sets up command information.
- Build map parsing

  FLMLTWST calls the FLMTBMAP translator to get the contents of the build map for the member being built. FLMLTWST parses the information in the build map to get the list of inputs that must be transferred to the workstation and any additional parameters that have been specified for the workstation command, such as a compiler or other tool. FLMLTWST also gets the list of outputs after the command is complete.

  At the same time, the SCLM member names are mapped to workstation file names based on the file name mapping information.
- Construct command parameters

  FLMLTWST supports running multiple workstation commands during each invocation. The parameters for each of the commands are put together based on the parameters passed to FLMLTWST, the contents of the build map (input and output file names can be included in the parameters), and on the workstation command information.
- Response file construction

  Some workstation commands support passing parameters using a file called a *response* file. If the workstation command information specifies a response file, one is created in a temporary data set and will be sent to the workstation with the other workstation command inputs.

  If multiple workstation commands will be issued, the response file for the first workstation command is sent with the input files. Response files for later commands are sent just before each command is run.

  Response files are only generated and sent to the workstation if the workstation command information indicates that one is to be used. If no response file is used, the command parameters are specified with the workstation command.
- Transfer inputs to the workstation

  FLMLTWST constructs a list of the input files (includes, source members, and response file) to be sent to the workstation. The FLMTXFER translator is then called to send the files to the workstation. FLMTXFER uses the FILEXFER service to transfer files to the workstation.

  The FLMTXFER translator keeps track of the SCLM members that have been sent to the workstation. This record is used to ensure that include members and source members are only transferred to the workstation once to reduce the time required to build a workstation application. The record of what has been transferred to the workstation is preserved in memory allocated by SCLM build. The result is that, within a single SCLM build, FLMTXFER only downloads a member once no matter how many source members that include it are built.

  If the date and time of the host member's statistics are the same as the date and time of its workstation counterpart, SCLM assumes that they are the same, and does not download the member a second time.
- Perform the workstation command

  FLMLTWST constructs the workstation command based on the information obtained in the set-up step. The command is issued on the workstation and SCLM waits for the result.

  Repeat this step for each workstation command that will run for the member being built. Before each command is issued, a response file is constructed and transferred to the workstation if needed.

- Transfer the outputs to the host system

  FLMLTWST uses a list of outputs obtained from the build map to construct a list of files to transfer from the workstation to the host system. The FLMTXFER translator performs the transfer from the workstation to the host. The data sets where the files are transferred are the data sets allocated to the ddname specified in the translator definition for FLMLTWST. If FLMLTWST ends successfully, build transfers the members into the SCLM hierarchy.

  If you have set the FLMALLOC macro IOTYPE=P, the date and time on the host member statistics are synchronized with the date and time of the corresponding workstation file, so that if the member is used for another build step, it will not be downloaded again.

Information For The Project Manager

Project Setup Considerations

You must consider several things when setting up a project to support workstation applications. This section covers items that are specific to workstation applications. Please see "Chapter 1. Defining the Project Environment" on page 3 for information on general project setup.

Naming Conventions

Determine what SCLM type names to use and the mapping between SCLM type names and workstation file extensions.

The recommended approach is to have a one-to-one mapping between the SCLM type and the workstation extension. In addition to the type-to-extension mapping, SCLM needs to know the format of the data within each type (ascii text or binary). To avoid having to define a mapping for each type, use something in the type name that indicates the format of the data. For example, add `BIN` to the workstation extension to create the SCLM type names for types that will contain binary data. This will minimize the number of mapping definitions for the ACTINFO file, because the wildcard character can be used to define a pattern in the type and extension names.

Another approach is to merge several workstation extensions into the same SCLM type. In this case, the workstation file names without the extension must be unique. The drawback of this approach is that after the files are combined into one SCLM type, they lose their individual extensions. The mapping is **from** the type **to** the workstation. SCLM does not know what a file was once called on the workstation. Only one extension can be defined for each type. This means that when the files are combined, SCLM will use the same extension for all of them when transferring them from or to the workstation. This might or might not be a problem, depending on the type of data combined. It would not be a good idea, for example, to combine C++ header files with H and HPP extensions into the same SCLM type, because the C++ source members might include header files with one or both of those extensions and would not find them if the extensions were changed. There might be other situations where the loss of the extension identity wouldn't matter.

Workstation file names, excluding the paths and extensions, must be valid ISPF PDS member names. Workstation file names can be in uppercase, lowercase, or have initial capital followed by lowercase letters. This mapping is specified using the **WSCASE** keyword in the ACTINFO file.

Languages

Next, you need to know which languages you will need.

One way to do this is to create a complex language definition that performs all of the steps required to go from source to executable code or to whatever you want the final result to be. The drawback to this approach is that when anything changes all of the steps are performed rather than the minimal set. For example, suppose there was a language that:

1. Compiled C source to an .obj
2. Compiled the resource source to an .res
3. Linked the .obj files into an .exe
4. Ran the resource compiler to add the resources from the .res to the .exe file

If the resource source changes, all of those steps are performed when some of them could be avoided.

Another approach is to create a language for each step. However, some tools produce outputs that are only needed until the next command is run. For example, the output from step 3 should not be saved into the hierarchy until after the resource compiler has been run. Saving one .exe into the SCLM hierarchy from the compiler and another copy from the resource compiler increases the project data set size and the time required to build.

A better approach is to create languages for each step that produces outputs that are kept permanently in the hierarchy. So, for the previous example, you would need three languages:

1. One language to compile C source and store the .obj files
2. One language to compile the resource source and store the .res files
3. One language to link the .obj files and add the resources from the .res files.

What Workstation Tools Will You Use?

The ACTION parameter on the FLMLTWST translator determines the workstation command that is run. The FLMLTWST translator maps the actions to a workstation command, determines the basic parameters to pass to that command, maps the workstation extensions to input and output parameters, and then orders the parameters.

In addition to the ACTION specified by the language definition, you can perform other actions in a build step by use of the CMD ACTION statement. For more information, refer to the FLMLTWST section of the *SCLM Reference* manual.

What Parameters Do You Need For the Workstation Tools?: Specify parameters in three places:

- In the translator (FLMLTWST). The parameters specified in FLMLTWST are used for every member of every language that calls it. They should be only the parameters that FLMLTWST requires, such as the parameters that specify the input and output file names.

  You can specify parameters to FLMLTWST for the workstation command in three ways:
  - In the language definition and on architecture PARM statements
  - On the architecture CMD statement (Refer to the FLMLTWST section of the *SCLM Reference* manual for more information on the CMD statement and its use with workstation applications).
  - Using parameters that are associated with inputs and outputs.

The order of the parameters is specified in the input data to the FLMLTWST translator and is the order required by the workstation command.

- On the FLMTRNSL macro in the language definition. These parameters are used for every member of the language. These should be parameters that the project requires. For example, the /Kg+ parameter can be specified to ensure that messages are produced for all goto statements.
- In an architecture member. These parameters are specific to a member. For example, the /DAPPL=A parameter can be used to define a preprocessor macro.

Workstation Information

The FLMLTWST translator needs information about the workstation such as the response file name and the directory name to prefix all files transfered to or from the workstation. It gets this information by reading from a data set.

The naming convention for the data set must be identified so that you can specify it in all the language definitions. Typically, the same information is used for all languages, although it is not required. The naming convention can include the variables to substitute the userid, project, group or other information into the data set name pattern. The variables used depend on where builds take place and on local data set naming standards. If the user determines the workstation, the userid should be part of the data set name. If the group determines the workstation, the group variable should be used without the userid variable. For more information on the USERINFODD parameter and the FLMCPYLB macro, refer to the *SCLM Reference*

How to Find What You Need

The International Technical Support Centers (ITSC) *Version 4 of ISPF and SCLM Implementation Guide* , GG24-4407, provides a good overview of SCLM and the ISPF Client/Server.

For information on setting up SCLM or PDF to view and edit on the workstation, see “The z/OS V1R1.0 ISPF User Interface” in the *ISPF User’s Guide*

Information on SCLM Workstation Build is available in both SCLM manuals. The *SCLM Developer’s and Project Manager’s Guide* contains information on SCLM support for workstation builds on OS/2 and Windows. The *SCLM Reference* manual, under “SCLM Translators”, contains information on the FLMLRC2 and FLMLRIPF sample parsers, as well as the FLMLTWST translator. For information on the ACTINFO files, USERINFO files, and workstation language definitions, see “FLMLTWST” in the *SCLM Reference*

The ISPF Sample and Macro libraries contain a number of files to support SCLM workstation builds. The ISPF Sample Library contains the following:

- FLMWBMIG - Sample migration EXEC for IBM CSET++ for OS/2 “Hello World 6” sample
- FLMWBUSR - Sample USERINFO file
- FLMWBAIO - Sample ACTINFO file for IBM CSET++ for OS/2 “Hello World 6” sample
- FLMWBAIW - Sample ACTINFO file for Borland (TM) C++ “Hello World” sample
- FLMWBPRJ - Sample workstation project definition
- FLMWBJCL - Sample JCL to allocate the data sets for the FLMWBPRJ sample project.
- FLMWBTMP - Sample workstation language definition template

- FLMWBOS2 - High-level architecture definition to build IBM CSET++ for OS/2 “Hello World 6” sample
- FLMWBIPF - Architecture definition to build IBM CSET++ for OS/2 “Hello World 6” help file
- FLMWBDLL - Architecture definition to build IBM CSET++ for OS/2 “Hello World 6” DLL file
- FLMWBEXE - Architecture definition to build IBM CSET++ for OS/2 “Hello World 6” EXE file
- FLMWBWIN - High-level architecture definition to build Borland C++ “Hello World” sample

The Macro Library contains sample language definitions for OS/2 and Windows. The IBM CSET++ for OS/2 language definitions are:

- FLM@WICC - Compile
- FLM@WDUM - Compile dummy object to hold DLLs
- FLM@WEXE - Link EXE
- FLM@WIPF - Build Help
- FLM@WLNK - Link386 to Link the DLL
- FLM@WRC - Resource compile

The Borland (TM) C++ for Windows language definitions are:

- FLM@WBCC - Compile
- FLM@WBRC - Resource Compile
- FLM@WTLK - TLINK OBJ to EXE

Information For The Developer

Migrating Applications into SCLM

To migrate a workstation application into SCLM:

1. Get the project information from the project manager. The information you need is:
   - The name of the development group where the members will be stored
   - The type names and their mapping to workstation file extensions
   - The languages to use for source members
   - The default parameters specified in the language definition for each language.
   - The actions and defaults specified in the ACTINFO file for workstation build.
2. Transfer the application source to the MVS system into the data sets for the development group based on the *workstation file to SCLM type name* mapping established for the project.

   Files containing data that can be edited on MVS must be transferred with ASCII-to-EBCDIC translation. Other files can be transferred in binary format (no translation). The FILEXFER service is recommended to avoid possible translation problems.
3. Migrate the members into SCLM using the languages supplied by the project manager.
4. Create architecture definition members as needed.

Architecture Definition Members for Workstation Applications

Architecture definition members must be created in any of the following cases:

- The source member requires options that were not specified in the language definition or action info data set.
- You need to override the inputs or outputs used in the language definition.
- The output member names are not the same as the source member name. See "Statement Uses" on page 255 for a description of the output keywords for architecture members.

  Some things can be done in the language definition to support adding a prefix or suffix to the output member name, but these capabilities do not support all possibilities. For more information, refer to the DFLTMEM parameter on the FLMALLOC macro in the *SCLM Reference Guide*.
- Outputs from the builds of other members are inputs to this build, for example, linking object modules together.
- Multiple workstation commands must be issued to complete the build step.
- To specify a relationship between components other than the source-to-include and input-to-output relationships generated by SCLM. An example would be to specify a relationship between the executable, DLL, and help components of a workstation application.

Specifying Options

Options can be specified to the workstation compiler, linker, or other tool by using the architecture definition CMD statement. This statement must be followed by the keyword PARMS and the parameters that are passed to the workstation tool. In the following example, the option '/Ss' is added to the options passed to the workstation tool.

```
SINC  SAMPLE   C       * source member
OBJ   SAMPLE   OBJBIN  * generated object member
LIST  SAMPLE   LISTING * listing file
*
* The following CMD statement has compile options for this member
*
CMD PARMS /Ss
```

Figure 97. Specifying Options in a Workstation Architecture Definition

If multiple CMD PARMS statements appear in the architecture member, the options are passed to the workstation tool in the order they appear in the architecture member. They are added to the workstation command as specified in the ACTINFO input to the FLMLTWST translator.

If you want to add options to be passed to the FLMLTWST translator, you can use the PARM and PARMx architecture statements. However, these options are considered FLMLTWST options rather than options for the workstation command.

Including Outputs From Other Build Steps

Use the architecture definition statements INCLD, INCL, and SINC to include members that are outputs from building other members. Using the INCLD and INCL statements ensures that SCLM builds the correct member to generate the output.

When a CC or generic architecture definition is built, SCLM uses the language definition of the member on the first SINC statement. For LEC architecture

definitions, the LE370 language is used. To override the language, specify the LKED architecture statement with the name of the language definition to use.

The following example shows an architecture member that can link several object members together to produce an .exe file. The language of EXE is used.

```
INCL  SAMPLE   ARCHDEF * archdef which produced sample object
INCLD COMMON   C       * source member which produced common object
*
LKED  EXE
*
LOAD  PROG1    EXEBIN  * .exe file
LMAP  PROG1    MAP     * listing file
```

Figure 98. Including Outputs as Inputs

Running Multiple Workstation Commands

Building some members requires that multiple workstation commands be issued. The FLMLTWST translator issues a workstation command for each action it finds. The first action is the one specified by the ACTION parameter to FLMLTWST in the language definition, or the default action if none is specified. Additional actions can be performed by using the architecture CMD statement with the ACTION keyword. The ACTION keyword must be followed by an action defined in the FLMLTWST translator.

The following example shows an architecture member that links two object modules together and then runs another workstation command prior to transferring the outputs to the MVS system. In this example, the second command runs the OS/2 resource compiler to add the information from a binary resource file to the .exe generated by the link.

```
*
LKED EXE                * link language
*
KREF OBJ                * include generated object modules
*
INCL  MAHJONGC ARCHDEF   * archdef that produces MAHJONGG OBJBIN
INCL  TILE     ARCHDEF   * archdef that produces TILE OBJBIN
SINC  MAHJONGG DEF       * DEF source
*
LOAD  MAHJONGG EXEBIN    * Generated .exe file
LMAP  MAHJONGG MAP       * Generated .map file
*
* Run resource compiler after the link completes
*
CMD ACTION RCEXE
*
KREF OUT1               * include generated .res file
*
INCLD MAHJONGG RC        * Source that produces MAHJONGG RESBIN
*
```

Figure 99. Multiple Workstation Commands

The order of the INCL and INCLD statements in the previous example is not important. The FLMLTWST translator determines which files are inputs to each step based on information defined in the translator. The appropriate options are also added for each of the inputs and outputs by the FLMLTWST translator.

Specifying Options

Options can be specified to the workstation compiler, linker, or other tool by using the architecture definition CMD statement. This statement must be followed by the keyword PARMS and the parameters that are passed to the workstation tool. In the following example, the option '/Ss' is added to the options passed to the workstation tool.

```
SINC  SAMPLE   C       * source member
OBJ   SAMPLE   OBJBIN  * generated object member
LIST  SAMPLE   LISTING * listing file
*
* The following CMD statement has compile options for this member
*
CMD PARMS /Ss
```

Figure 100. Specifying Options in a Workstation Architecture Definition

If multiple CMD PARMS statements appear in the architecture member, the options are passed to the workstation tool in the order they appear in the architecture member. They are added to the workstation command as specified in the ACTINFO input to the FLMLTWST translator.

If you want to add options to be passed to the FLMLTWST translator, you can use the PARM and PARMx architecture statements. However, these options are considered FLMLTWST options rather than options for the workstation command.

Including Outputs From Other Build Steps

Use the architecture definition statements INCLD, INCL, and SINC to include members that are outputs from building other members. Using the INCLD and INCL statements ensures that SCLM builds the correct member to generate the output.

When a CC or generic architecture definition is built, SCLM uses the language definition of the member on the first SINC statement. For LEC architecture definitions, the LE370 language is used. To override the language, specify the LKED architecture statement with the name of the language definition to use.

The following example shows an architecture member that can link several object members together to produce an .exe file. The language of EXE is used.

```
INCL  SAMPLE   ARCHDEF * archdef which produced sample object
INCLD COMMON   C       * source member which produced common object
*
LKED  EXE
*
LOAD  PROG1    EXEBIN  * .exe file
LMAP  PROG1    MAP     * listing file
```

Figure 101. Including Outputs as Inputs

Running Multiple Workstation Commands

Building some members requires that multiple workstation commands be issued. The FLMLTWST translator issues a workstation command for each action it finds. The first action is the one specified by the ACTION parameter to FLMLTWST in the language definition, or the default action if none is specified. Additional

actions can be performed by using the architecture CMD statement with the ACTION keyword. The ACTION keyword must be followed by an action defined in the FLMLTWST translator.

The following example shows an architecture member that links two object modules together and then runs another workstation command prior to transferring the outputs to the MVS system. In this example, the second command runs the OS/2 resource compiler to add the information from a binary resource file to the .exe generated by the link.

```
*
LKED EXE                * link language
*
KREF OBJ                * include generated object modules
*
INCL  MAHJONGC ARCHDEF   * archdef that produces MAHJONGG OBJBIN
INCL  TILE     ARCHDEF   * archdef that produces TILE OBJBIN
SINC  MAHJONGG DEF       * DEF source
*
LOAD  MAHJONGG EXEBIN    * Generated .exe file
LMAP  MAHJONGG MAP       * Generated .map file
*
* Run resource compiler after the link completes
*
CMD ACTION RCEXE
*
KREF OUT1               * include generated .res file
*
INCLD MAHJONGG RC        * Source that produces MAHJONGG RESBIN
*
```

Figure 102. Multiple Workstation Commands

The order of the INCL and INCLD statements in the previous example is not important. The FLMLTWST translator determines which files are inputs to each step based on information defined in the translator. The appropriate options are also added for each of the inputs and outputs by the FLMLTWST translator.

Sample Language Definition

The following sample shows a language definition for compiling C source members on the workstation. A description of the items in the language definition follows.

```
*********************************************************************
*                                                                   *
*     SCLM LANGUAGE DEFINITION FOR IBM CSET/2 OR CSET++ FOR OS/2    *
*                      COMPILE SOURCE TO OBJECT                     *
*                                                                   *
*********************************************************************
*
*
CPPOS2   FLMLANGL    LANG=CPPOS2,                                    C
               VERSION=2,                                           C
               CHKSYSLB=IGNORE
*
          FLMINCLS TYPES=(H,HPP,@@FLMTYP,@@FLMETP)
H         FLMINCLS TYPES=(H)
HPP       FLMINCLS TYPES=(HPP)
*
* PARSER
*
          FLMTRNSL   CALLNAM='C/C++ PARSE',                        C
               FUNCTN=PARSE,                                       C
               CALLMETH=TSOLNK,                                    C
               COMPILE=FLMLRC2,                                    C
               PORDER=1,                                           C
               OPTIONS=(STATINFO=@@FLMSTP,                         C
               LISTINFO=@@FLMLIS,                                  C
               LISTSIZE=@@FLMSIZ)
*
*          (* SOURCE *)
           FLMALLOC  IOTYPE=A,DDNAME=SOURCE
            FLMCPYLB @@FLMDSN(@@FLMMBR)
*
* BUILD
*
          FLMTRNSL   CALLNAM='C/C++',                              C
               FUNCTN=BUILD,                                       C
               CALLMETH=ISPLNK,                                    C
               COMPILE=SELECT,                                     C
               VERSION=1,                                          C
               GOODRC=0,                                           C
               PORDER=1,                                           C
               OPTIONS='CMD(FLMLTWST ACTION=COMPILE,BMAPINFO=@@FLM$MP,SC
               CLMINFO=@@FLMINF,BLDINFO=@@FLMBIO,PARMS='
*
```

Figure 103. Workstation C Language Definition (Part 1 of 2)

```
*          (* OBJ *)
           FLMALLOC  IOTYPE=P,RECFM=VB,LRECL=1024,                    C
               RECNUM=4000,DDNAME=OBJ,CATLG=Y,KEYREF=OBJ,             C
               DFLTTYP=OBJBIN,DFLTMEM=*,LANG=EXE
*          (* LIST *)
           FLMALLOC  IOTYPE=O,RECFM=VB,LRECL=256,                     C
               RECNUM=4000,DDNAME=LIST,CATLG=Y,PRINT=I,               C
               KEYREF=LIST,DFLTTYP=LST
*          (* USERINFO *)
           FLMALLOC  IOTYPE=A,DDNAME=USERINFO
             FLMCPYLB @@FLMUID.SCLM.USERINFO
*          (* ACTINFO *)
           FLMALLOC  IOTYPE=A,DDNAME=ACTINFO
             FLMCPYLB @@FLMPRJ.PROJDEFS.ACTINFO
*          (* MESSAGE *)
           FLMALLOC  IOTYPE=W,RECFM=VB,LRECL=256,DISP=MOD,            C
               RECNUM=4000,DDNAME=MESSAGE,PRINT=I
*          (* MSGXFER *)
           FLMALLOC  IOTYPE=W,RECFM=VB,LRECL=256,CATLG=Y,             C
               RECNUM=4000,DDNAME=MSGXFER
*          (* BMAP *)
           FLMALLOC  IOTYPE=W,RECFM=VB,LRECL=256,                     C
               RECNUM=4000,DDNAME=BMAP,PRINT=I
*          (* FILES *)
           FLMALLOC  IOTYPE=W,RECFM=VB,LRECL=256,CATLG=Y,             C
               RECNUM=4000,DDNAME=FILES,PRINT=I
*          (* RESPONSE *)
           FLMALLOC  IOTYPE=W,RECFM=VB,LRECL=256,                     C
               RECNUM=4000,DDNAME=RESPONSE,PRINT=I,CATLG=Y
*
```

Figure 103. Workstation C Language Definition (Part 2 of 2)

FLMLANGL macro
This macro specifies the language name, CPPOS2, the language version, ″1″, and that SCLM is to ignore any includes that are not in the project hierarchy.

FLMINCLS macro
This macro indicates the types searched when looking for includes. Includes with the workstation file extension 'h' are found in the H type. Other includes are found in the type of the source member or its extended type.

FLMTRNSL macro (functn=parse)
This macro identifies the parser to use when the members of this language are updated. The parser scans the member for include dependencies and counts statistics. See the *SCLM Reference Guide* for a description of the FLMLRC2 parser.

FLMTRNSL macro (functn=build)
This is the definition of the build translator. It calls FLMLTWST to perform the compile on the workstation. The ACTION parameter is set to compile to indicate that the compiler is to be called. The PARMS parameter at the end of the parameter string allows for PARM keywords in the language definition to specify additional parameters. The other parameters are used to pass information between SCLM build and the translators that FLMLTWST calls.

FLMALLOC macro (ddname=obj)
This macro allocates the ddname that will hold the .obj file generated on the workstation. The RECFM and LRECL values must match the allocation of the data set in the hierarchy where the .obj file will be stored.

IOTYPE=O
indicates that a sequential data set will be allocated to hold the output

IOTYPE=P
indicates that a partitioned data set will be allocated to hold the output. Using IOTYPE=P can improve build performance for builds with more than one step by copying the date and time of the workstation file to the host member. If the file is needed for subsequent build steps, the copy on the workstation will be used rather than downloading the file that was just uploaded.

DFLTMEM=*
indicates that the output member in the PDS will have the same name as the member being built

RECNUM
Indicates the maximum number of records that can be stored in the data set

CATLG=Y
Allows the file to be transferred from the workstation to the data set allocated to this ddname

KEYREF=OBJ
indicates that this is an object module. This references the architecture OBJ statement. See the *SCLM Developer's Guide* for more information on architecture statements.

DFLTTYP
indicates the type in the hierarchy where the member is stored.

LANG
Gives the language to associate with the output member. This can be used later if the member is the input to another translator.

Because the KEYREF parameter is OBJ, the FLMLTWST translator requires the ddname to be OBJ also or the OBJ parameter must be specified giving the ddname. For example, to use the ddname OBJBIN for outputs with a KEYREF of OBJ, you must specify "OBJ=OBJBIN" in the options string of the FLMLTWST translator.

FLMALLOC macro (ddname=list)
This is the allocation for the ddname to hold the .lst (listing) file that was generated on the workstation. This FLMALLOC has IOTYPE=O to allocate a sequential data set to hold the listing that will be stored back in the hierarchy. The PRINT parameter is also specified to initialize the data set and then copy it to the user's BUILD.LISTnn data set if needed. The IOTYPE=O or IOTYPE=P is needed because of the PRINT parameter.

FLMALLOC macro (ddname=userinfo)
This macro allocates the USERINFO data set. The FLMCPYLB macro that follows it allocates an existing data set to the ddname. The data set has the userid as the high-level qualifier, followed by SCLM.USERINFO. See the description of the FLMLTWST translator for the contents of this data set.

FLMALLOC macro (ddname=actinfo)
This is the allocation for the ACTINFO data set. The FLMCPYLB macro that follows it allocates an existing data set to the ddname. The data set has the project as the high-level qualifier, followed by "PROJDEFS.ACTINFO".

FLMALLOC macro (ddname=message)
This ddname stores messages from the translators that FLMLTWST calls. If the FLMTXFER translator fails, this is the first place to look.

FLMALLOC macro (ddname=msgxfer)
This ddname is used to transfer message files from the workstation to the host. After the messages are transferred to the host, they are appended to the messages ddname.

FLMALLOC macro (ddname=bmap)
This is the ddname where the FLMTBMAP translator writes the build information.

FLMALLOC macro (ddname=files)
This is the ddname to which FLMLTWST writes the list of files for FLMTXFER to transfer.

FLMALLOC macro (ddname=response)
This is the ddname where FLMLTWST generates the response file that is sent to the workstation. ACTION=COMPILE uses a response file; but if no response file is needed for the action, this ddname can be omitted.

Workstation Setup

Workstation build expects the workstations to transfer files and issue commands in a consistent way. However, some information can vary from workstation to workstation. This information is contained in the user info data set allocated to the ddname that is specified by the USERINFO parameter when calling the FLMLTWST translator. Refer to the description of the FLMLTWST translator in the *SCLM Reference* for information on the contents of this data set.

Directories and File Names

FLMLTWST constructs workstation file names from four components:

- The data directory is obtained from the userinfo data set (as specified by the DATA_DIR keyword). It can contain drive letters and whatever is necessary to establish the base path for the files and subdirectories.
- The subdirectory is obtained from the ACTINFO data set. The subdirectory is based on the type of the member. Subdirectories can be used to place different types of members in different directories for the workstation command or tool.
- The file name is the SCLM member name.
- The extension is obtained from the ACTINFO data set that maps SCLM types to extensions.
- The case (upper or lower) of the workstation file name is set based on the **WSCASE** value specified in the ACTINFO data set.

When SCLM constructs the full file name from the above components, it does not add or remove any characters from each of the components. Each component must be set up so that when it is combined with the others it will make a valid file name.

The FLMLTWST translator as it is shipped expects the data directory name not to end with a '/' or '\', but the subdirectory should start and end with these characters. The extension contains the '.' character.

Following are some examples of how FLMLTWST would put these four components together:

Data Directory	Subdirectory	File Name (Member)	Extension	Generated File Name
e:\temp	\	example1	.c	e:\temp\example1.c
e:	\temp\	example2	.h	e:\temp\example2.h
\temp	\bin\	example3	.exe	\temp\bin\example3.exe

The FLMLTWST translator does not clean out the directories after the workstation command is complete and the outputs have been transferred to the MVS system. The workstation owner must clean out the directories periodically to ensure that the workstation disk(s) do not fill up.

Multiple Builds on One Workstation

SCLM supports using a single workstation for doing multiple builds either for a single user or multiple users. However, if the builds are taking place at different groups, either the base directory or the subdirectory must differ based on the group. This will avoid the problem of different builds overlaying one another's files.

One setup would have all builds at a specific group in the SCLM hierarchy occur on a specific workstation. In this case, the hierarchy view for all builds taking place on the workstation will be consistent so a single set of directories can be used or the directory names can vary based on the user performing the build.

Another setup would have a separate workstation for each user. In this case, either each user would need to ensure that all builds running concurrently are for the same group or the directory names would need to vary based on the group where the build is taking place.

Two methods to vary the directory name by the build group are:

- Include the @@FLMGRP variable in the FLMCPYLB allocation of the USERINFO data set. Then ensure that the USERINFO data sets that now include the group name in the data set name also vary the base directory based on the group name.
- Update the logic of FLMLTWST to accept a parameter with the group name where the build is taking place. Then generate the subdirectory based on the group. The language definition must set the group parameter to @@FLMGRP to pick up the build group.

Part 4. Appendixes

Appendix. SCLM Variables and MetaVariables

SCLM Variable and Metavariable Descriptions

SCLM variables are character strings that SCLM replaces with a value. SCLM replaces these variables with eight-character values except for the following:

- @@FLMBD4 variable has a value with a maximum length of 10
- @@FLMCD4 variable has a value with a maximum length of 10
- @@FLMDOx variable has a value with a maximum length of 44 (*x* is an integer between 0 and 9).
- @@FLMDSD variable has a value with a maximum length of 44
- @@FLMDSF variable has a value with a maximum length of 44
- @@FLMDSN variable has a value with a maximum length of 44
- @@FLMDST variable has a value with a maximum length of 44
- @@FLMICN variable has a value with a maximum length of 110
- @@FLMID4 variable has a value with a maximum length of 10
- @@FLMINC variable contains an address in decimal character format
- @@FLMINF variable contains an address in decimal character format
- @@FLMLIS variable contains an address in decimal character format
- @@FLMMD4 variable has a value with a maximum length of 10
- @@FLMPD4 variable has a value with a maximum length of 10
- @@FLMSTP variable contains an address in decimal character format
- @@FLMXCN variable has a value with a maximum length of 110
- @@FLM$C4 variable has a value with a maximum length of 10
- @@FLM$MP variable has a value with a maximum length of the build map.
- @@FLM$UD variable has a value with a maximum length of 128
- @@FLM$XD variable has a value with a maximum length of 110
- @@FLM$XN variable has a value with a maximum length of 110
- @@FLM$XU variable has a value with a maximum length of 110

In addition to these variables, SCLM has metavariables that represent SCLM internal tracking data. Table 25 on page 309 lists the SCLM metavariables and their corresponding SCLM variables. Use a metavariable in place of a combination of single SCLM variables. Variables are listed in the order in which their data values appear in the database contents utility report. There are metavariables for the fixed portion of the data and for the long (repeating) portion of the data. Table 24 on page 308 lists the SCLM metavariables and a short description of each.

You can use SCLM variables in the following places:

- On the FLMINCLS macro TYPES parameter. The following variables are supported for this parameter:
  - @@FLMCRF
  - @@FLMECR
  - @@FLMETP
  - @@FLMTYP
- With the PARM and PARMX architecture definition keywords
- On the FLMTRNSL macro OPTIONS parameter
- On the FLMALLOC macro MEMBER parameter. The following variables are supported for this parameter:
  - @@FLMMBR
  - @@FLMONM

- On the FLMCPYLB macro. The following variables are supported for FLMCPYLB statements associated with an IOTYPE I or an IOTYPE A FLMALLOC macro:
  - @@FLMALT
  - @@FLMDBQ
  - @@FLMDSN
  - @@FLMGRP
  - @@FLMMBR
  - @@FLMPRJ
  - @@FLMSRF
  - @@FLMTYP
  - @@FLMUID
- On the Database Contents Utility line format parameter (DBUTIL)
- On the DSNAME parameter on the FLMCNTRL and FLMALTC macros. The following variables are supported for these parameters:
  - @@FLMGRP
  - @@FLMPRJ
  - @@FLMTYP
- On the EXPACCT and EXPXREF parameters of the FLMCNTRL and FLMALTC macros. The following variables are supported for these parameters:
  - @@FLMGRP
  - @@FLMPRJ
  - @@FLMUID
- On the VERPDS parameter of the FLMCNTRL and FLMALTC macros. The following variables are supported for these parameters:
  - @@FLMDSN
  - @@FLMGRP
  - @@FLMPRJ
  - @@FLMTYP

Many of the variables can be used only for certain translator types and the SCLM utilities. Table 22 lists the SCLMvariables in alphabetic order by description and indicates for which translator types they can be used. Table 23 on page 305 lists the SCLM variables in alphabetic order by variable name.

SCLM Variable and Metavariable Tables

The following tables illustrate SCLM variables and metavariables and their SCLM functions. Pass these variables to a translator using the OPTIONS= parameter of the FLMTRNSL macro.

Variables marked with a `P` are passed to PDS member (`PDSDATA=Y` on the FLMTRNSL macro) translators.

Variables marked with an `I` are passed to Ada Intermediate translators (`PDSDATA=N` on the FLMTRNSL macro.)

Variables marked with an `E` are passed to the External dependency translators (such as CSP/370AD.)

Variables marked with a ✓ are passed to the DBUTIL service.

Note: Certain variables are passed to multiple translators depending on their function and data.

SCLM Variable Descriptions, Variable Names, and Their SCLM Functions

Table 22 lists the SCLM variables in alphabetic order by their short description.

Table 22. SCLM Variable Descriptions, Names, and Their SCLM Functions

SCLM Short Description	Variable	Build	Copy	Parse	Purge	Verify	Utils
Access Key	@@FLMACK						✓
Accounting Group	@@FLMGRP	P	P I E	P	P I E	P	✓
Accounting Group Data Set Name	@@FLMDSN	P	P	P	P	P	✓
Accounting Member	@@FLMMBR	P	P	P	P	P	✓
Accounting Record Type	@@FLMATP						✓
Accounting Status	@@FLMSTA						✓
Accounting Type	@@FLMTYP	P	P	P	P	P	✓
Alternate Project Definition	@@FLMALT	P	P I	P	P I	P	✓
Assignment Statements	@@FLMASG						✓
Authorization Code	@@FLMACD						✓
Authorization Code Change	@@FLMACC						✓
Blank Lines	@@FLMBLL						✓
Buffer Size in Bytes	@@FLMSIZ	P E	E	P	E	E	
Build Group	@@FLMGRB	P					✓
Build Map	@@FLM$MP	P					✓
Build Map Information	@@FLMBIO	P					
Build Map Date	@@FLMMDT		P		P	P	✓
Build Map Date with 4-character year	@@FLMMD4		P		P	P	✓
Build Map Name	@@FLMMNM						✓
Build Map Time	@@FLMMTM		P		P	P	✓
Build Map Type	@@FLMMSC						✓
Build Mode	@@FLMBMD					E	
Calling Function Name	@@FLMFNM		P I		P I	P	
Change Code	@@FLM$CC						✓
Change Code Date	@@FLM$CD						✓

Table 22. SCLM Variable Descriptions, Names, and Their SCLM Functions (continued)

SCLM Short Description	Variable	Build	Copy	Parse	Purge	Verify	Utils
Change Code Date with 4-character year	@@FLM$C4						✓
Change Code Time	@@FLM$CT						✓
Change Date	@@FLMCDT		P		P	P	✓
Change Date with 4-character year	@@FLMCD4		P		P	P	✓
Change Group	@@FLMCLV						✓
Change Time	@@FLMCTM		P		P	P	✓
Change User ID	@@FLMCUS						✓
Comment Lines	@@FLMCML						✓
Comment Statements	@@FLMCMS						✓
Control Statements	@@FLMCNS						✓
Creation Date	@@FLMIDT						✓
Creation Date with 4-character year	@@FLMID4						✓
Creation Time	@@FLMITM						✓
CREF Type	@@FLMCRF						
CU List	@@FLMLST		I		I		
Database Qualifier	@@FLMDBQ	P	I		I		✓
Data Set Name for OUT0	@@FLMDO0	P E					
Data Set Name for OUT1	@@FLMDO1	P E					
Data Set Name for OUT2	@@FLMDO2	P E					
Data Set Name for OUT3	@@FLMDO3	P E					
Data Set Name for OUT4	@@FLMDO4	P E					
Data Set Name for OUT5	@@FLMDO5	P E					
Data Set Name for OUT6	@@FLMDO6	P E					
Data Set Name for OUT7	@@FLMDO7	P E					
Data Set Name for OUT8	@@FLMDO8	P E					
Data Set Name for OUT9	@@FLMDO9	P E					
DDNAME Substitution List	@@FLMDDN			P			
Default Type	@@FLMSRF	P					

Table 22. SCLM Variable Descriptions, Names, and Their SCLM Functions (continued)

SCLM Short Description	Variable	Build	Copy	Parse	Purge	Verify	Utils
Dependencies Pointer	@@FLMLIS	P E	E	P	E	E	
Destination Group	@@FLMGRD		P		P	P	
Destination Group Data Set Name	@@FLMDSD		P		P	P	
Dynamic Includes Pointer	@@FLMINC	P					
Extended CREF Type	@@FLMECR						
Extended Type of Source Member	@@FLMETP						
Function Invocation Date	@@FLMFDT	P	P		P	P	
Function Invocation Time	@@FLMFTM	P	P		P	P	
Group Found	@@FLMGRF		P		P	P	
Group Found Data Set Name	@@FLMDSF		P		P	P	
Include	@@FLM$IN						✓
Include-Sets for Includes	@@FLM$IS						✓
Language	@@FLMLAN		P		P	P	✓
Language Version	@@FLMLVS						✓
Member Version	@@FLMMVR						✓
Number of Change Codes	@@FLMNCC						✓
Number of Includes	@@FLMNIN						✓
Number of Noncomment Lines	@@FLMNCL						✓
Number of Noncomment Statements	@@FLMNCS						✓
Number of User Entries	@@FLMNUE						✓
Output Member Name	@@FLMONM						
OUT0 Member Name	@@FLMOU0	P					
OUT1 Member Name	@@FLMOU1	P					
OUT2 Member Name	@@FLMOU2	P					
OUT3 Member Name	@@FLMOU3	P					

Table 22. SCLM Variable Descriptions, Names, and Their SCLM Functions (continued)

SCLM Short Description	Variable	Build	Copy	Parse	Purge	Verify	Utils
OUT4 Member Name	@@FLMOU4	P					
OUT5 Member Name	@@FLMOU5	P					
OUT6 Member Name	@@FLMOU6	P					
OUT7 Member Name	@@FLMOU7	P					
OUT8 Member Name	@@FLMOU8	P					
OUT9 Member Name	@@FLMOU9	P					
Predecessor Date	@@FLMBDT						✓
Predecessor Date with 4-character year	@@FLMBD4						✓
Predecessor Time	@@FLMBTM						✓
Project	@@FLMPRJ	P	P I	P	P I	P	✓
Prolog Lines	@@FLMPRL						✓
Promote Date	@@FLMPDT						✓
Promote Date with 4-character year	@@FLMPD4						✓
Promote Time	@@FLMPTM						✓
Promote User ID	@@FLMPUS						✓
SCLM Internal Data Pointer	@@FLMINF	P E	P I E		P I E	P E	
SCLM Version	@@FLMVER						✓
Static Pointer	@@FLMSTP			P			
Sysprint DDNAME	@@FLMDDO		P I		P I	P	
System User ID	@@FLMUID	P	P		P	P	
Target Group	@@FLMTOG		P I E		P E	P	
Target Group Data Set Name	@@FLMDST		P		P	P	
Top CU Name	@@FLMCUN	P					
Total Lines	@@FLMTLL						✓
Total Statements	@@FLMTLS						✓
Translator Version	@@FLMTVS						✓
User Data Entry	@@FLM$UD						✓

SCLM Variables and Their SCLM Functions

Table 23 lists the SCLM variables in alphabetic order by variable name.

Table 23. SCLM Variables and Their SCLM Functions

Variable	SCLM Short Description	Build	Copy	Parse	Purge	Verify	Utils
@@FLMACC	Authorization Code Change						✓
@@FLMACD	Authorization Code						✓
@@FLMACK	Access Key						✓
@@FLMALT	Alternate Project Definition	P	P I	P	P I	P	✓
@@FLMASG	Assignment Statements						✓
@@FLMATP	Accounting Record Type						✓
@@FLMBDT	Predecessor Date						✓
@@FLMBD4	Predecessor Date with 4-character year						✓
@@FLMBIO	Build Map Information	P					
@@FLMBLL	Blank Lines						✓
@@FLMBMD	Build Mode					E	
@@FLMBTM	Predecessor Time						✓
@@FLMCDT	Change Date		P		P	P	✓
@@FLMCD4	Change Date with 4-character year		P		P	P	✓
@@FLMCLV	Change Group						✓
@@FLMCML	Comment Lines						✓
@@FLMCMS	Comment Statements						✓
@@FLMCNS	Control Statements						✓
@@FLMCRF	CREF Type						
@@FLMCTM	Change Time		P		P	P	✓
@@FLMCUN	Top CU Name	P					
@@FLMCUS	Change User ID						✓
@@FLMDBQ	Database Qualifier	P	I		I		✓
@@FLMDDN	DDNAME Substitution List			P			
@@FLMDDO	Sysprint DDNAME		P I		P I	P	
@@FLMDO0	Data Set Name for OUT0	P E					
@@FLMDO1	Data Set Name for OUT1	P E					
@@FLMDO2	Data Set Name for OUT2	P E					
@@FLMDO3	Data Set Name for OUT3	P E					

Table 23. SCLM Variables and Their SCLM Functions (continued)

Variable	SCLM Short Description	Build	Copy	Parse	Purge	Verify	Utils
@@FLMDO4	Data Set Name for OUT4	P E					
@@FLMDO5	Data Set Name for OUT5	P E					
@@FLMDO6	Data Set Name for OUT6	P E					
@@FLMDO7	Data Set Name for OUT7	P E					
@@FLMDO8	Data Set Name for OUT8	P E					
@@FLMDO9	Data Set Name for OUT9	P E					
@@FLMDSD	Destination Group Data Set Name		P		P	P	
@@FLMDSF	Group Found Data Set Name		P		P	P	
@@FLMDSN	Accounting Group Data Set Name	P	P	P	P	P	✓
@@FLMDST	Target Group Data Set Name		P		P	P	
@@FLMECR	Extended CREF Type						
@@FLMETP	Extended Type of Source Member						
@@FLMFDT	Function Invocation Date	P	P		P	P	
@@FLMFNM	Calling Function Name		P I		P I	P	
@@FLMFTM	Function Invocation Time	P	P		P	P	
@@FLMGRB	Build Group	P					
@@FLMGRD	Destination Group		P		P	P	
@@FLMGRF	Group Found		P		P	P	
@@FLMGRP	Accounting Group	P	P I E	P	P I E	P	✓
@@FLMIDT	Creation Date						✓
@@FLMID4	Creation Date with 4-character year						✓
@@FLMINC	Dynamic Includes Pointer	P					
@@FLMINF	SCLM Internal Data Pointer	P E	P I E		P I E	P E	
@@FLMLAN	Language		P		P	P	✓
@@FLMLIS	Dependencies Pointer	P E	E	P	E	E	
@@FLMLST	CU List		I		I		

Table 23. SCLM Variables and Their SCLM Functions (continued)

Variable	SCLM Short Description	Build	Copy	Parse	Purge	Verify	Utils
@@FLMLVS	Language Version						✓
@@FLMMBR	Accounting Member	P	P	P	P	P	✓
@@FLMMDT	Build Map Date		P		P	P	✓
@@FLMMD4	Build Map Date with 4-character year		P		P	P	✓
@@FLMMNM	Build Map Name						✓
@@FLMMSC	Build Map Type						✓
@@FLMMTM	Build Map Time		P		P	P	✓
@@FLMMVR	Member Version						✓
@@FLMNCC	Number of Change Codes						✓
@@FLMNCL	Number of Noncomment Lines						✓
@@FLMNCS	Number of Noncomment Statements						✓
@@FLMNIN	Number of Includes						✓
@@FLMNUE	Number of User Entries						✓
@@FLMONM	Output Member Name						
@@FLMOU0	OUT0 Member Name	P					
@@FLMOU1	OUT1 Member Name	P					
@@FLMOU2	OUT2 Member Name	P					
@@FLMOU3	OUT3 Member Name	P					
@@FLMOU4	OUT4 Member Name	P					
@@FLMOU5	OUT5 Member Name	P					
@@FLMOU6	OUT6 Member Name	P					
@@FLMOU7	OUT7 Member Name	P					
@@FLMOU8	OUT8 Member Name	P					
@@FLMOU9	OUT9 Member Name	P					
@@FLMPDT	Promote Date						✓

Table 23. SCLM Variables and Their SCLM Functions (continued)

Variable	SCLM Short Description	Build	Copy	Parse	Purge	Verify	Utils
@@FLMPD4	Promote Date with 4-character year						✓
@@FLMPRJ	Project	P	P I	P	P I	P	✓
@@FLMPRL	Prolog Lines						✓
@@FLMPTM	Promote Time						✓
@@FLMPUS	Promote User ID						✓
@@FLMSIZ	Buffer Size in Bytes	P E	E	P	E	E	
@@FLMSRF	Default Type	P					
@@FLMSTA	Accounting Status						✓
@@FLMSTP	Static Pointer			P			
@@FLMTLL	Total Lines						✓
@@FLMTLS	Total Statements						✓
@@FLMTOG	Target Group		P I E		P E	P	
@@FLMTVS	Translator Version						✓
@@FLMTYP	Accounting Type	P	P	P	P	P	✓
@@FLMUID	System User ID	P	P		P	P	
@@FLMVER	SCLM Version						✓
@@FLM$CC	Change Code						✓
@@FLM$CD	Change Code Date						✓
@@FLM$C4	Change Code Date with 4-character year						✓
@@FLM$CT	Change Code Time						✓
@@FLM$IN	Include						✓
@@FLM$IS	Include-Sets for Includes						✓
@@FLM$MP	Build Map						✓
@@FLM$UD	User Data Entry						✓

SCLM Metavariable Descriptions, Metavariable Names, and Their SCLM Functions

Table 24 lists the SCLM metavariables in alphabetic order by description. Metavariables are only used with the DBUTIL service.

Table 24. SCLM Metavariable Descriptions, Names, and Their SCLM Functions

SCLM Short Description	Metavariable	Build	Copy	Parse	Purge	Verify	Utils
Account Report Fixed	@@FLM#AF						✓
Account Report Long	@@FLM#AL						✓

SCLM Metavariable Contents

Table 25 lists the SCLM metavariables and their corresponding SCLM variables. A metavariable represents a list of predefined SCLM variables. Specifying a metavariable is equivalent to specifying its corresponding list of SCLM variables in the order listed in Table 25.

Table 25. SCLM Metavariables and Their Corresponding Variables

Metavariable	Variable
@@FLM#AF	@@FLMPRJ @@FLMALT @@FLMGRP @@FLMTYP @@FLMMBR @@FLMVER @@FLMSTA @@FLMCDT @@FLMCTM @@FLMCLV @@FLMCUS @@FLMMVR @@FLMLAN @@FLMATP @@FLMLVS @@FLMACD @@FLMACC @@FLMACK @@FLMIDT @@FLMITM @@FLMMDT @@FLMMTM @@FLMBDT @@FLMBTM @@FLMPDT @@FLMPTM @@FLMPUS @@FLMDBQ @@FLMTVS @@FLMMNM @@FLMMSC @@FLMTLL @@FLMCML @@FLMNCL @@FLMBLL @@FLMPRL @@FLMTLS @@FLMCMS @@FLMCNS @@FLMASG @@FLMNCS @@FLMNUE @@FLMNIN @@FLMNCC @@FLMNCU @@FLM$IN @@FLM$IS @@FLM$CC @@FLM$CD @@FLM$CT

Table 25. SCLM Metavariables and Their Corresponding Variables (continued)

Metavariable	Variable
@@FLM#AL	@@FLM$XT @@FLM$XN @@FLM$UD

Description of Group Variables

This section further explains the use of group variables. Table 26 lists each group variable and associated group data set name variable. This shows the relationship between SCLM groups and the data sets defined in the project definition for each group.

Table 27 on page 311 is an example that lists the values of each group variable during the phases of a promote. After Table 26 is an overall description of the four group variables and why each is needed. Each group variable has a corresponding data set name variable due to the flexible data set name capability.

Table 26. SCLM Group Variable List

Group Variable	Group Data Set Name Variable	Description
@@FLMGRP	@@FLMDSN	Accounting Group and Accounting Group Data Set Name
@@FLMGRF	@@FLMDSF	Group Found and Group Found Data Set Name
@@FLMTOG	@@FLMDST	Target Group and Target Group Data Set Name
@@FLMGRD	@@FLMDSD	Destination Group and Destination Group Data Set Name

The following hierarchy will be used in the description:

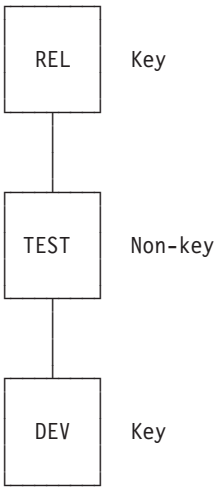


Figure 104. Hierarchy Example for Group Description

Given the preceding hierarchy, the following table describes what each group variable would contain during which translator phase of a PROMOTE from TEST to REL.

Table 27. SCLM Group Variable Description

Translator	Accounting Group	Group Found	Target Group	Destination Group
Verify	TEST	TEST	REL	REL
Copy	TEST	TEST	REL	REL
Purge key	DEV	TEST	DEV	REL
Purge non-key	TEST	TEST	TEST	REL

The purge translator is invoked twice during this promote due to the promotion from a non-key group to a key group.

Notices

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

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

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

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

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

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

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

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

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

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact the IBM Corporation, Department TL3B, 3039 Cornwallis Road, Research Triangle Park, North Carolina, 27709–2195, USA. Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

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

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non_IBM products should be addressed to the suppliers of those products.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Programming Interface Information

This book documents information that is not intended to be used as programming interfaces of ISPF.

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

BookManager
C++
DFSMSdfp
DFSMSdss
DFSMShsm
DFSMSrmm
DFSMS/MVS
DFSORT
ESCON
FFST
GDDM
IBM
Language Environment
MVS
MVS/ESA
OS/2
OS/390
OS/390 Security Server
RACF
Resource Access Control Facility
SOMobjects
System View
VisualLift
VTAM

Microsoft and Windows are registered trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, and service names may be trademarks or service marks of others.

Glossary of SCLM Terms

A

access key. An identifier used to restrict access to a member.

accounting information. Accounting information is stored in the SCLM VSAM accounting data sets and consists of accounting and build map records.

accounting record. An SCLM control data record containing statistical, historical, and dependency information for a member under SCLM control.

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

alternate project definition. A project definition that provides a version of the project environment which differs from the default project definition.

application. Software that performs a function for an end user.

API. Application Programming Interface

APT. Application Programming and Test

architecture. The organization of software components to form integrated applications.

architecture definition. A means of organizing components of an application into conceptual units. It is SCLM's method of defining an application's configuration. It describes how the components of an application fit together and is used to drive both the build and promote functions. Architecture definitions are used to group components into applications, sub-applications, and load modules.

architecture member. Defines an individual software component, which may be a collection of other architecture members, by specifying its relationship to other software components of an application.

audit information. Information associated with a member which describes when a member was modified, how it was modified, and who modified it. This information is stored in the SCLM VSAM audit data sets.

audit trail. See *audit information*.

authorization code. An identifier used by SCLM to control authority to update and promote members within a hierarchy. These codes can be used to allow concurrent development without the risk of module collisions (overlayed changes).

authorization group. An identifier associated with a set of authorization codes.

B

build. The process of transforming inputs into outputs through the invocation of translators specified in the language definition. Compilers, preprocessors, and linkage editors are examples of translators that might be invoked at build time.

build map. Internal data record containing a complete analysis of the database at the time of the build; it includes the names of all referenced members and the last change date and version number of each member.

C

change code. An eight-character identifier used to indicate the reason for an update or modification to a member controlled by SCLM.

code. Program(s) written in a language that is subject to a given translation process.

compilable member. A member recognized by the compiler or translator as an independent unit or a controlling unit for the language.

component. See *software component*.

concurrent updates. Concurrent updates occur when two programmers update the same member at the same time. This is supported through the use of authorization codes and the Edit Compare tool or alternate project definitions.

configuration management. See *software configuration management*.

configuration management plan. See *software configuration management plan*

control data. Information that SCLM stores about each member under its control. The control data is stored in the accounting and audit VSAM data sets defined for a project.

copylib. A library containing include referenced source code.

cross-reference record. Internal data record containing Ada compilation unit/member relationship information.

D

data base. SCLM-controlled VSAM data sets for a project.

database administrator. See *project administrator*.

ddname substitution list. A string of ddnames allocated for the translator. The ddname substitution list is usually documented in the Programmer's Guide for compilers and linkage editors.

default architecture definition. Architecture definition that is generated by SCLM when one is not specified as input to a build. This is done when a source member is built directly.

default project definition. The main project definition used by an SCLM project.

dependency. Dependency describes a relationship between a source member and the members it includes. A source member has a dependency on a member which it includes.

dependency information. Information on dependencies is stored in the SCLM accounting record.

development group. All groups in the lowest level of the hierarchy are known as "development groups". These groups represent end-nodes with no other lower groups promoting into them.

development layer. Layer of an SCLM hierarchy consisting of development groups.

development life cycle. The process followed to create an application. The process starts at the program requirements gathering phase, moves to the design phase, the development phase, and continues to the release of the final product.

downward dependency. A dependency indicating a compilation unit which must be compiled after the current compilation unit is compiled.

draw down. During edit, SCLM copys the member from its first occurrence in a key group in the library concatenation into a development group and locks it.

dynamic include. An include for a source member that cannot be resolved until after the translator invocation.

dynamic reference. A reference that involves a variable.

E

editable/non-editable. Source members (created by an edit session) are editable; members produced by a processor during a build are non-editable.

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

F

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

G

group. A set of project data sets with the same middle-level qualifier in the SCLM logical naming convention.

H

hierarchical view. A path of groups (concatenation) through the hierarchy. The path may start at any group in the hierarchy and follows the promote path to the topmost group in the hierarchy.

hierarchy. The organization of groups in a ranked order, where each group is subordinate to the one above it.

I

include. A member that is required to complete a compile of the member that references it.

include-set. An include-set is used to associate an included member name with the type or types in the project which are searched to find a member with that name.

integrate. To merge two or more software components of an application into a single software application.

K

key group. Data is copied into this group and then purged from the previous group, effectively "moving" the data. Non-key groups are used when a simple copy is desired.

L

language definition. Specifies the set of translators to be executed for SCLM functions PARSE, VERIFY, BUILD, COPY, and PURGE. A language definition is composed of one FLMLANGL macro followed by an FLMTRNSL macro for each translator to be executed for members of SCLM libraries whose language attribute matches the value of the LANG keyword in the FLMLANGL macro.

layer. A given tier of the hierarchy, made up of groups of equivalent rank.

level. See *layer*.

library (MVS). A partitioned data set.

lock. When a user locks a member, only that user can change it. All other users are unable to change that member until the member is promoted or unlocked. When you lock a member, you specify an authorization code. If two users need to change a part, they can use different authorization codes.

lock service. Restricts (locks) a member to a development group.

M

maximum promotable group. The topmost group to which a member can be promoted.

member. The discrete element of an SCLM database, representing a single data type of a software component.

metavariable. A variable that includes many other SCLM variables.

migrate. Registering software components in SCLM: this includes identifying the component language, and possibly the change code and authorization code.

migration. The process of introducing members into SCLM control. Migration locks the member, parses it according to the requested language, and stores the information in the accounting data base. You can user the migration utility to enter a large number of members into a project's data base, such as during conversion to SCLM.

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

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

N

nested dependencies. Nested dependencies occur when a source member includes another member, which in turn includes another member. SCLM tracks nested dependencies, so that when a member changes, any member that includes it is rebuilt, no matter how many levels of nesting there are.

non-key group. A group that data is copied into (as opposed to moved into) during promotion.

P

parser. A program that reads an editable member to determine dependency and statistical information about the member. This information is stored in the SCLM accounting data base.

predecessor date/time. The last modified date/time stamp taken from the previous version of the current member.

point-and-shoot text. Text on a screen that is cursor sensitive.

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

predecessor verification. The process of verifying that the previous version of a member has not changed.

predecessors. Previous versions of a member existing at a higher level within the same hierarchical view.

primary commands. Editing commands that are entered on the Command line.

primary group. A key or non-key group with two or more groups promoting into it that must be allocated when a hierarchy is to be accessed.

private library. A partitioned data set or partitioned data set extended belonging to a group in the development layer of the hierarchy.

project. A collection of libraries representing an integrated SCLM data base, under a single high-level qualifier.

project administrator. The person who maintains an SCLM project.

project definition. Defines the SCLM library structure, project control information, and language definitions. A project definition is a load module used by SCLM at run time. The source code for a project definition is composed of macros.

project definition data. Project definitions and language definitions which are used to create and control an SCLM project.

project environment. Information which makes up an SCLM project. There are three types of information:

- Project Definition Data
- User Applications Data
- Control Data

project identifier. The name assigned to the project definition.

Project Partitioned Data Sets. MVS Partitioned Data Sets where user application data is stored.

promote. The process of moving an application or its components from one level in the project hierarchy to the next. Promotion out of a development group removes the lock on editable members that were successfully promoted.

promote path. The link between two groups along which data moves from one subordinate group to the next group in the hierarchy.

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

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

S

SCLM_id. Identifier used to communicate information between the SCLM services. There is a unique SCLM_id generated for each invocation of the INIT service.

scope. The set of members (including architecture definitions) which will be processed (verified, copied, compiled, purged, etc.) by build or promote.

service. An SCLM function available via a command or programming interface.

service parameter list. The options supplied when invoking an SCLM service.

software component. Any input or output member associated with an application, which together make up all or a member of the application.

software configuration management. The method of controlling and integrating software components to produce high quality applications. Provides a common point of integration for all planning and implementation activities for a project.

software configuration management plan. A formalized procedure for software configuration management.

subapplications. Separate parts of an application being developed within a project. Once the project is completed, the parts are integrated to form the final product.

syslib. A library containing source code not under SCLM control. No dependency information is maintained for members in a syslib.

T

text. Data present in its natural language form (not translatable).

traceability. Capability to access and maintain records of information about a software component, including when the component was last changed and why.

translator. A load module, CLIST, or REXX program that receives control from SCLM for execution. The name of the translator is specified as the value of the COMPILE keyword for the FLMTRNSL macro. Examples of translators are compilers, assemblers, linkage editors, text processors, DB2 preprocessors, CICS preprocessors, utilities, and customer tools.

type. The third qualifier of the SCLM naming convention for project partitioned data sets. Typically identifies the kind of data maintained for a project hierarchy. Examples of types are SOURCE, OBJECT and LOAD.

U

unlock. To make a member (formerly locked out) available for updating (usually associated with promote).

unlock service. Removes the restriction (unlocks) on a member to a development group.

upward dependency. A dependency indicating a compilation unit that must be compiled before the current compilation unit is compiled.

V

Version. A copy of a member as it existed at a previous point in time.

Versioning. A function that enables you to retrieve a version of a member. Useful for ″backing out″ changes.

Index

Special Characters

A

B

C

D

P

R

S

T

U

V

W

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

Interactive System Productivity Facility (ISPF)
Software Configuration and Library Manager (SCLM) Project Manager's and Developer's Guide
z/OS Version 1 Release 1.0

Publication No. SC34-4817-00

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	☐	☐	☐	☐	☐

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	☐	☐	☐	☐	☐
Complete	☐	☐	☐	☐	☐
Easy to find	☐	☐	☐	☐	☐
Easy to understand	☐	☐	☐	☐	☐
Well organized	☐	☐	☐	☐	☐
Applicable to your tasks	☐	☐	☐	☐	☐

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? ☐ Yes ☐ No

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

Name _____ Address _____

Company or Organization _____

Phone No. _____

Readers' Comments — We'd Like to Hear from You
SC34-4817-00

IBM

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
Software Reengineering
Department G7IA / Bldg 503
Research Triangle Park, NC
27709-9990

Fold and Tape | **Please do not staple** | Fold and Tape

Cut or Fold
Along Line

File Number: S370/4300-39
Program Number: 5694-A01

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SC34-4817-00