

Interactive System Productivity Facility (ISPF)



# Software Configuration and Library Manager (SCLM) Reference

*z/OS Version 1 Release 2.0*



Interactive System Productivity Facility (ISPF)



# Software Configuration and Library Manager (SCLM) Reference

*z/OS Version 1 Release 2.0*

**Note**

Before using this document, read the general information under "Notices" on page 283.

**Second Edition (October 2001)**

This edition applies to ISPF for Version 1 Release 2 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>

© Copyright International Business Machines Corporation 1990, 2001. All rights reserved.  
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

# Contents

<b>Preface</b> . . . . .	<b>vii</b>	Parameters . . . . .	29
Who Should Use This Book . . . . .	vii	Return Codes. . . . .	30
What Is in This Book . . . . .	vii	Examples . . . . .	30
<b>Summary of Changes</b> . . . . .	<b>ix</b>	BUILD—Build a Member. . . . .	32
ISPF Product Changes . . . . .	ix	Command Invocation Format . . . . .	32
ISPF DM Component Changes . . . . .	ix	Call Invocation Format . . . . .	33
ISPF PDF Component Changes . . . . .	xi	Parameters . . . . .	33
ISPF SCLM Component Changes . . . . .	xii	Return Codes. . . . .	35
ISPF Client/Server Component Changes . . . . .	xii	Examples . . . . .	35
ISPF User Interface Considerations . . . . .	xii	DBACCT—Retrieve Accounting Records for a	
ISPF Migration Considerations. . . . .	xiii	Member . . . . .	36
ISPF Profiles. . . . .	xiii	Command Invocation Format . . . . .	36
Year 2000 Support for ISPF . . . . .	xiii	Call Invocation Format . . . . .	36
		Parameters . . . . .	36
		Return Codes. . . . .	37
		Example . . . . .	37
<b>What's in the z/OS V1R2.0 ISPF</b>		DBUTIL—Generate a Tailored Output Data Set and	
<b>library?</b> . . . . .	<b>xv</b>	Report . . . . .	38
z/OS V1R2.0 ISPF . . . . .	xv	Command Invocation Format . . . . .	38
		Call Invocation Format . . . . .	38
		Parameters . . . . .	38
		Return Codes. . . . .	41
		Example . . . . .	41
<b>Elements and Features in z/OS.</b> . . . .	<b>xvii</b>	DELETE—Delete Database Components. . . . .	42
		Command Invocation Format . . . . .	42
		Call Invocation Format . . . . .	42
		Parameters . . . . .	43
		Return Codes. . . . .	43
		Examples . . . . .	44
<b>Chapter 1. Migrating from Previous</b>		DELGROUP—Delete Group Database Components	44
<b>Versions of SCLM</b> . . . . .	<b>1</b>	Command Invocation Format . . . . .	45
FLMCPYLB Statements Required for IOTYPE=A . . . . .	1	Call Invocation Format . . . . .	45
Versioning Data Sets. . . . .	1	Parameters . . . . .	45
Include Sets . . . . .	1	Return Codes. . . . .	47
Year 2000 Support . . . . .	2	Examples . . . . .	48
FLMALLOC Processing for IOTYPE S . . . . .	2	DSALLOC—Allocate Data Sets for Group/Type . . . . .	48
Load Module Accounting Records and SSI		Command Invocation Format . . . . .	48
Information. . . . .	2	Call Invocation Format . . . . .	49
		Parameters . . . . .	49
		Return Codes. . . . .	50
		Examples . . . . .	50
<b>Chapter 2. SCLM Services</b> . . . . .	<b>5</b>	EDIT— Edit a Member of a Controlled Library . . . . .	51
Invoking the SCLM Services . . . . .	5	Command Invocation Format . . . . .	51
Notation Conventions Used in this Chapter . . . . .	5	Call Invocation Format . . . . .	52
Command Invocation of the SCLM Services . . . . .	5	Parameters . . . . .	52
The FLMCMD Interface . . . . .	6	Return Codes. . . . .	54
Call Invocation of the SCLM Services . . . . .	9	Example . . . . .	54
The FLMLNK Subroutine Interface . . . . .	9	END— End an SCLM Services Session . . . . .	54
Types of Parameters . . . . .	12	Command Invocation Format . . . . .	55
ISPF Variables . . . . .	17	Call Invocation Format . . . . .	55
SCLM Service Return Codes. . . . .	20	Parameters . . . . .	55
FLMCMD Command Processor Return Codes . . . . .	21	Return Codes. . . . .	55
FLMLNK Call Processor Return Codes . . . . .	21	Example . . . . .	55
SCLM Service Messages . . . . .	22	EXPORT—Extract SCLM Accounting Information	
SCLM Service Descriptions . . . . .	23	for a Group . . . . .	55
ACCTINFO—Retrieve Accounting Information . . . . .	25	Command Invocation Format . . . . .	56
Command Invocation Format . . . . .	25		
Call Invocation Format . . . . .	26		
Parameters . . . . .	26		
Return Codes. . . . .	27		
AUTHCODE—Retrieve or Set Authorization Code			
for Selected Members . . . . .	28		
Command Invocation Format . . . . .	28		
Call Invocation Format . . . . .	28		

Call Invocation Format . . . . .	56	Return Codes. . . . .	82
Parameters . . . . .	56	Example . . . . .	82
Return Codes. . . . .	57	SAVE—Lock, Parse, and Store a Member . . . . .	83
Examples . . . . .	57	Command Invocation Format . . . . .	83
FREE—Free an SCLM ID . . . . .	58	Call Invocation Format . . . . .	83
Command Invocation Format . . . . .	58	Parameters . . . . .	84
Call Invocation Format . . . . .	58	Return Codes. . . . .	86
Parameters . . . . .	58	Examples . . . . .	87
Return Codes. . . . .	58	START—Generate an Application ID for a Services Session . . . . .	87
Example . . . . .	58	Command Invocation Format . . . . .	88
IMPORT—Import SCLM Accounting Information to Current Project . . . . .	59	Call Invocation Format . . . . .	88
Command Invocation Format . . . . .	59	Parameters . . . . .	88
Call Invocation Format . . . . .	59	Return Codes. . . . .	88
Parameters . . . . .	60	Example . . . . .	88
Return Codes. . . . .	61	STORE—Store Member Information in an Accounting Record . . . . .	89
Examples . . . . .	61	Command Invocation Format . . . . .	89
INIT—Generate an SCLM ID . . . . .	62	Call Invocation Format . . . . .	89
Command Invocation Format . . . . .	62	Parameters . . . . .	90
Call Invocation Format . . . . .	62	Return Codes. . . . .	91
Parameters . . . . .	62	Example . . . . .	91
Return Codes. . . . .	63	UNLOCK—Unlock a Member in a Development Library . . . . .	92
Example . . . . .	63	Command Invocation Format . . . . .	93
LOCK—Lock a Member or Assign an Access Key . . . . .	63	Call Invocation Format . . . . .	93
Command Invocation Format . . . . .	65	Parameters . . . . .	93
Call Invocation Format . . . . .	65	Return Codes. . . . .	94
Parameters . . . . .	66	Examples . . . . .	94
Return Codes. . . . .	67	VERDEL—Delete Version and Audit Information. . . . .	95
Examples . . . . .	67	Command Invocation Format . . . . .	95
MIGRATE—Create Accounting for Selected Members . . . . .	68	Call Invocation Format . . . . .	95
Command Invocation Format . . . . .	68	Parameters . . . . .	95
Call Invocation Format . . . . .	68	Return Codes. . . . .	96
Parameters . . . . .	69	VERINFO—Retrieve Version and Audit Information . . . . .	97
Return Codes. . . . .	70	Command Invocation Format . . . . .	97
Examples . . . . .	70	Call Invocation Format . . . . .	97
NEXTGRP— Retrieve Next Group in SCLM Hierarchy . . . . .	71	Parameters . . . . .	97
Command Invocation Format . . . . .	71	Return Codes. . . . .	99
Call Invocation Format . . . . .	72	VERRECOV—Recover a Version . . . . .	100
Parameters . . . . .	72	Command Invocation Format . . . . .	100
Return Codes. . . . .	72	Call Invocation Format . . . . .	101
Examples . . . . .	72	Parameters . . . . .	101
PARSE—Parse a Member for Statistical and Dependency Information . . . . .	73	Return Codes . . . . .	102
Command Invocation Format . . . . .	73	<b>Chapter 3. Sample Programs Using SCLM Services. . . . .</b>	<b>105</b>
Call Invocation Format . . . . .	74	Pascal Example. . . . .	105
Parameters . . . . .	74	Main Program FLMSRV1 . . . . .	106
Return Codes. . . . .	75	Included Member FLMSRV1D . . . . .	112
Example . . . . .	75	Included Member FLMSRV1S . . . . .	115
PROMOTE—Promote a Member from One Library to Another. . . . .	76	PL/I Example . . . . .	121
Command Invocation Format . . . . .	76	<b>Chapter 4. SCLM Macros . . . . .</b>	<b>127</b>
Call Invocation Format . . . . .	76	Notes on Using the SCLM Macros . . . . .	128
Parameters . . . . .	77	FLMABEG Macro . . . . .	129
Return Codes. . . . .	79	Macro Format . . . . .	129
Examples . . . . .	80	Parameters . . . . .	129
RPTARCH—Generate an SCLM Architecture Report . . . . .	80	Example . . . . .	129
Command Invocation Format . . . . .	81	FLMAEND Macro. . . . .	130
Call Invocation Format . . . . .	81		
Parameters . . . . .	81		

Macro Format . . . . .	130	Parameters . . . . .	189
Parameters . . . . .	130	Examples . . . . .	190
FLMAGRP Macro . . . . .	130	FLMTRNSL Macro . . . . .	190
Macro Format . . . . .	130	Macro Format . . . . .	190
Parameters . . . . .	130	Parameters . . . . .	191
Example . . . . .	130	Examples . . . . .	195
FLMALLOC Macro . . . . .	130	FLMTYPE Macro . . . . .	196
Macro Format . . . . .	132	Macro Format . . . . .	196
Parameters . . . . .	133	Parameters . . . . .	196
Example 1 . . . . .	148	Example . . . . .	196
Example 2 . . . . .	148		
Example 3 . . . . .	148		
FLMALTC Macro . . . . .	148	<b>Chapter 5. SCLM Translators. . . . .</b>	<b>197</b>
Macro Format . . . . .	149	FLMCSPDB DB2 Bind/Free Translator . . . . .	199
Parameters . . . . .	149	FLMDTLC DTL Processor Build Translator . . . . .	202
Example . . . . .	151	FLMLPCBL COBOL Parser . . . . .	203
FLMATVER Macro . . . . .	152	FLMLPFRT FORTRAN Parser . . . . .	206
Macro Format . . . . .	152	FLMLPGEN General Purpose Parser . . . . .	210
Parameters . . . . .	152	FLMLRASM REXX Assembler Parser . . . . .	214
Example . . . . .	154	FLMLRCBL REXX COBOL Parser . . . . .	218
FLMCNTRL Macro . . . . .	155	FLMLRCIS MVS C/C++ parser with include set support . . . . .	222
Macro Format . . . . .	155	FLMLRC2 C, C++, and Resource file parser for workstation source . . . . .	225
Parameters . . . . .	157	FLMLRC37 REXX C370 Parser. . . . .	228
Example . . . . .	172	FLMLRDTL REXX DTL Parser. . . . .	232
FLMCPYLB Macro . . . . .	173	FLMLRIPF Script and OS/2 IPF Source Parser . . . . .	233
Macro Format . . . . .	173	FLMLSS General Purpose Parser . . . . .	236
Parameters . . . . .	173	FLMLTWST Workstation Build Translator . . . . .	240
Example . . . . .	174	FLMTBMAP Build Map Print - Build Translator	256
FLMGROUP Macro . . . . .	174	FLMTMSI Interface to SCRIPT/VS . . . . .	258
Macro Format . . . . .	175	FLMTPRE . . . . .	259
Parameters . . . . .	175	FLMTPST . . . . .	261
Example 1 . . . . .	175	FLMTXFER Workstation Transfer - Build Translator	263
Example 2 . . . . .	176	SCLM Parser Restrictions . . . . .	266
FLMINCLS Macro . . . . .	176	Non-Explicit References . . . . .	266
Macro Format . . . . .	177	Separation of References. . . . .	267
Parameters . . . . .	177		
Example 1 . . . . .	178	<b>Chapter 6. SCLM Variables and</b>	
Example 2 . . . . .	178	<b>Metavariables . . . . .</b>	<b>269</b>
Example 3 . . . . .	179	SCLM Variable and Metavariable Descriptions . . . . .	269
FLMLANGL Macro . . . . .	180	SCLM Variable and Metavariable Tables . . . . .	270
Macro Format . . . . .	180	SCLM Variable Descriptions, Variable Names, and Their SCLM Functions . . . . .	271
Parameters . . . . .	180	SCLM Variables and Their SCLM Functions . . . . .	275
Example 1 . . . . .	182	SCLM Metavariable Descriptions, Metavariable Names, and Their SCLM Functions . . . . .	279
FLMLRBLD Macro . . . . .	182	SCLM Metavariable Contents . . . . .	279
Macro Format . . . . .	183	Description of Group Variables . . . . .	281
Parameters . . . . .	183		
Examples . . . . .	183	<b>Notices . . . . .</b>	<b>283</b>
FLMSYSLB Macro . . . . .	183	Programming Interface Information . . . . .	284
Macro Format . . . . .	184	Trademarks . . . . .	284
Parameters . . . . .	184		
Example . . . . .	184	<b>Glossary of SCLM Terms . . . . .</b>	<b>285</b>
FLMTCOND Macro . . . . .	185		
Macro Format . . . . .	186	<b>Index . . . . .</b>	<b>289</b>
Parameters . . . . .	186		
Examples . . . . .	188		
FLMTOPTS Macro . . . . .	189		
Macro Format . . . . .	189		



---

## Preface

This book provides reference and usage information, along with conceptual and functional descriptions of the Software Configuration and Library Manager (SCLM).

---

## Who Should Use This Book

This book is for:

- Application developers whose projects are controlled by SCLM
- Project administrators 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.

All SCLM users should read Chapters 1, 2, and 3 of *ISPF Software Configuration and Library Manager (SCLM) Developer's and Project Manager's Guide*. The rest of the chapters in this manual assume that you have read and understood Chapter 1 of *ISPF Software Configuration and Library Manager (SCLM) Developer's and Project Manager's Guide*.

Chapter 2. *SCLM Services*, introduces and describes the services that are used to retrieve and process certain information that you store in the project hierarchies. It lists the general categories of SCLM service return codes and provides command and call invocation formats, return codes, and parameters for each service. It also explains the notation conventions used to document the services.

Chapter 3. *Sample Programs Using SCLM Services*, provides sample programs in Pascal and PL/I that allow you to invoke SCLM services.

Chapter 4. *SCLM Macros*, introduces and describes the macros that are used to create project definitions for SCLM. It also explains the notation conventions used to document the macros.

Chapter 5. *SCLM Translators*, describes the translators delivered with SCLM. For each translator, there is a brief description, a list of input parameters, and a list of return codes with the appropriate user and project administrator responses.

Chapter 6. *SCLM Variables and Metavariables*, lists the SCLM variables and identifies each function with which they can be used.

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



---

## Summary of Changes

z/OS V1R2.0 ISPF contains the following changes and enhancements:

- ISPF Product and Library Changes
- ISPF Dialog Manager Component Changes (including DTL changes)
- ISPF PDF Component Changes
- ISPF SCLM Component Changes
- ISPF Client/Server Component Changes

---

### ISPF Product 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
- = 5.2 means ISPF for z/OS Version 1 Release 2.0

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 z/OS 01.02.00**.

New system variables:

#### **ZDAYOFWK**

The day of the week.

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:

- Add support for "VER(&variable,IPADDR4)".
- Add the NOSETMSG parameter to the CONTROL Service.
- Add the LFORMAT parameter to the VDEFINE Service to allow defining like format variables in a list.
- Change tutorial processing to eliminate the "End of data" message on scrollable area panels that display the entire scrollable area on the screen (no More: + - is displayed). This change eliminates the extra enter the user had to execute before continuing to the next panel.

- Issue a TSO line message when a help panel is not found and continue the dialog. Previously ISPF issued a severe error message when a help panel could not be found.
- Display a message indicating a message is not found when running in Dialog Test and allow the dialog to continue.
- Add support for extended SBCS and DBCS CCSIDs:
  - 1159 Traditional Chinese
  - 1364 Korean
  - 1371 Traditional Chinese
  - 1388 Simplified Chinese
  - 1390 Japanese
  - 1399 Japanese
- Add new Z variables to support 5 character code pages and character sets, ZTERMCP5 and ZTERMCS5 respectively.
- Add new variable ZDAYOFWK to show the day of the week.
- Enhance the Reflist function of TEST option 7.6 to allow better list management.
- Enhance Locate and Find for Dialog Test Variables (option 7.3).
- A new exec called ISPCMDTB to convert ISPF command tables to DTL.
- A new Configuration Table variable to allow SCROLL defaults.
- A new Configuration Table variable to allow STATUS AREA defaults.

ISPD TLC enhancements:

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

General improvements:

- New invocation options:
  - no new invocation options in this release
- New tags:
  - DLDIV, DTDIV, DTHDIV for dividers within the DL tag
  - PLDIV, PTDIV for dividers within the PARML tag
- Replication added to predefined entities. For example, &GTSYM(5); will create the string '>>>>' in the substituted text.
- National language text strings are now accessible as entities. For example, &command; will create the string 'Command' or its translated equivalent in the substituted text.
- New ENTITY keywords COPIES, X2C and ATTR.
- New macro tag default initialization processing syntax.
 

```
<?dummy ?var=value>
```
- New Predefined ENTITY keywords cmdpmt (&cmdpmt;) and rptr (&rptr;).

New or changed tag attributes:

Tag name	Attribute update
CHECKI	Add support for "VER(&variable, IPADDR4)"
COMPOPT	Add ADD.

Tag name	Attribute update
DL	Add FORMAT. Support multiple DT tags for each DD tag. Change TSIZE to support multiple values. Each TSIZE value implies a DT tag.
DT	Add FORMAT, NOSKIP.
DTAFLD	Add AUTOTYPE, AUTOVOL, AUTODMEM.
HELP	Add ZUP, ZCONT.
Hn	Add COMPACT.
HP	Add INTENSE.
NOTE	Add NOSKIP.
NT	Add NOSKIP.
PANEL	Add ZUP, ZCONT, AUTONRET, AUTOTCMD.
PARML	Add FORMAT. Support multiple PT tags for each PD tag. Change TSIZE to support multiple values. Each TSIZE value implies a PT tag.
PT	Add FORMAT, SKIP.
SELFLD	Add SELCHECK.
	Support INIT=init-value for single-choice selection fields.

## ISPF PDF Component Changes

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

- A MEMBER command has been added to data set list (option 3.4) to allow the partitioned data sets in the list to be searched for a specific member.
- When the EDIT service is specified with an initial macro, parameters can now be specified for the initial macro.
- A FIND command has been added to member list to allow a string to be searched for in any of the displayed statistics.
- A SRCHFOR command has been added to data set list to allow SuperC to be invoked to search the listed data sets for strings.
- Move/Copy will now dynamically calculate the sized for the IEBCOPY SYSUT3 and SYSUT4 data sets.
- A QUERYENQ service has been added to retrieve ENQ information about a data set in use.
- LMF has been removed from the ISPF product.
- A new SuperC option FINDALL has been added to specify that all strings must be found to issue a "strings found" return code.
- LMPRINT will now allow the INDEX parameter to be specified for a record format U data set.
- Foreground and Batch now support the z/OS C/C++ compiler.

- A new AUTOTYPE command can be set to a PFKEY to retrieve a data set name or pattern entered on a panel based on data sets that start with that partial name.
- Data sets with an LRECL less than 10 bytes can be edited or viewed.
- The Edit CUT and PASTE command defaults have been added to the ISPF Configuration Table.
- The Edit CUT and PASTE default behaviors have been modified to use CUT REPLACE and PASTE KEEP.
- The BARRIER keyword has been added to the SELECT for Edit macros.
- A program called ISREMSPY that can be invoked from an Edit macro to display the current Edit data.
- The Edit macro commands CURSOR, LINENUM and DISPLAY\_LINES can retrieve line numbers greater than 999999.

---

## ISPF SCLM Component Changes

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

- Several enhancements to the Library Utility:
  - A member action to initiate Promotion on a member.
  - REFRESH command to update the member list contents.
  - HIER ON|OFF command to switch between full hierarchy view and single group view.
  - Edit action can create a new member when entered on the command line.
  - Ability to select deletion of accounting data or build map only.
- New FLMLRBLD macro to select automated rebuild for members with a specified language on promotion to listed groups.
- Improved edit models for SCLM services.
- VOL keyword on the FLMCPYLB and FLMSYSLB macros allowing reference to uncatalogued data sets.
- VIO keyword on the FLMALLOC macro to override the SCLM-calculated default unit of DASD or VIO for temporary data sets.
- Supplied parsers and translators are all loaded RMODE(31).

---

## 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 V1R2.0 from OS/390 V2R10.0, there are no migration actions necessary. If you are migrating to z/OS V1R2.0 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.
- LMF has been removed from the ISPF product. For information about how to convert from LMF to SCLM refer to the *ISPF Planning and Customizing* manual.

|  
|  
|

---

## What's in the z/OS V1R2.0 ISPF library?

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

---

### z/OS V1R2.0 ISPF

<b>Title</b>	<b>Order Number</b>
<i>z/OS V1R2.0 ISPF Dialog Tag Language Guide and Reference</i>	SC34-4824-01
<i>z/OS V1R2.0 ISPF Planning and Customizing</i>	GC34-4814-01
<i>z/OS V1R2.0 ISPF User's Guide Volume I</i>	SC34-4822-01
<i>z/OS V1R2.0 ISPF User's Guide Volume II</i>	SC34-4823-01
<i>z/OS V1R2.0 ISPF Services Guide</i>	SC34-4819-01
<i>z/OS V1R2.0 ISPF Dialog Developer's Guide and Reference</i>	SC34-4821-01
<i>z/OS V1R2.0 ISPF Reference Summary</i>	SC34-4816-01
<i>z/OS V1R2.0 ISPF Edit and Edit Macros</i>	SC34-4820-01
<i>z/OS V1R1.0 ISPF Library Management Facility</i>	SC34-4825-01
<i>z/OS V1R2.0 ISPF Messages and Codes</i>	SC34-4815-01
<i>z/OS V1R2.0 ISPF Software Configuration and Library Manager Project Manager's and Developer's Guide</i>	SC34-4817-01
<i>z/OS V1R2.0 ISPF Software Configuration and Library Manager Reference</i>	SC34-4818-01
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 2.0. z/OS V1R2.0 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	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
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)	
SOMobjects Runtime Library (RTL)	SOMobjects Runtime Library (RTL)	base
SOMobjects service classes	SOMobjects service classes	base
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) <ul style="list-style-type: none"> <li>• DCE Security Server</li> <li>• OS/390 Firewall Technologies</li> <li>• Lightweight Directory Access Protocol (LDAP) Client and Server</li> <li>• Open Cryptographic Enhanced Plug-ins (OCEP)</li> </ul>	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 <ul style="list-style-type: none"> <li>• TCP/IP CICS Sockets</li> <li>• TCP/IP IMS Sockets</li> <li>• TCP/IP Kerberos</li> <li>• TCP/IP Network Print Facility (NPF)</li> <li>• TCP/IP OS/390 Communications Service IP Applications</li> <li>• TCP/IP OS/2 Offload</li> </ul>	TCP/IP <ul style="list-style-type: none"> <li>• TCP/IP CICS Sockets</li> <li>• TCP/IP IMS Sockets</li> <li>• TCP/IP Kerberos</li> <li>• TCP/IP Network Print Facility (NPF)</li> <li>• TCP/IP OS/390 Communications Service IP Applications</li> <li>• TCP/IP OS/2 Offload</li> </ul>	base <ul style="list-style-type: none"> <li>• optional</li> <li>• optional</li> <li>• optional</li> <li>• optional</li> <li>• optional</li> <li>• optional</li> </ul>
TIOC R1	TIOC	base
Time Sharing Option Extensions (TSO/E) V2R5	TSO/E	base

<b>Product Name and Level</b>	<b>Name in z/OS</b>	<b>Base or Optional</b>
VisualLift for MVS V1R1.1	<ul style="list-style-type: none"> <li>• VisualLift Run-Time Environment (RTE)</li> <li>• VisualLift Application Development Environment (ADE)</li> </ul>	<ul style="list-style-type: none"> <li>• base</li> <li>• optional</li> </ul>
VTAM V4R3 with the AnyNet feature	VTAM	base
3270 PC File Transfer Program V1R1.1	3270 PC File Transfer Program	base



---

## Chapter 1. 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.

---

### FLMCPYLB Statements Required for IOTYPE=A

In z/OS V1R2.0 ISPF and later, SCLM project definitions must have an FLMCPYLB statement identifying a data set name for every FLMALLOC statement with IOTYPE=A or MALLOC=Y. Project and language definitions with missing statements (including sample languages and the sample project included in prior releases) will receive an assembly error, which will be MNOTEs following the FLMAEND statement. If this error is detected when a project definition is assembled, you can correct it using one of the following:

- Add the missing FLMCPYLB statements with an appropriate data set name (or specify NULLFILE)
- Change the IOTYPE on the FLMALLOC to an appropriate value for the translator being used

---

### Versioning Data Sets

In OS/390 V2R10 ISPF and later, 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 V1R2.0 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.



---

## Chapter 2. SCLM Services

This chapter describes each of the SCLM services and the syntax conventions and return codes for the services. It discusses how to call the services from your terminal with interactive command processing, procedures, or programs. This chapter also provides several brief examples of how to invoke the services.

Each service description includes an example of its use in the command procedure format and the Pascal call format. Default settings for each service call are shown in the command procedure format section for each service; the default values are underscored. Call invocations do not have defaults because some value must be specified for each parameter; a blank is identified for each parameter that will translate a blank into a default value. See Chapter 3. Sample Programs Using SCLM Services for an example of service invocations and declarations coded in Pascal.

---

### Invoking the SCLM Services

Invoke the SCLM services by a program function dialog through a call to FLMCMD or FLMLNK, or by a command function dialog (CLIST or REXX) through the ISPF interface.

### Notation Conventions Used in this Chapter

This chapter uses the following notation conventions to describe the format of the SCLM services:

- Uppercase** Uppercase commands or parameters must be spelled out as shown (in either uppercase or lowercase).
- Lowercase** Lowercase parameters are variables; substitute your own values.
- Underscore** Underscored parameters are the system default.
- Brackets ( [ ] )** Parameters in brackets are optional.
- Braces ( { } )** Braces show two or more parameters from which you must select one.
- OR ( | )** The OR ( | ) symbol separates two or more values (inside braces) from which you must select one.
- Single quotes ( ' ' )** Single quotes show service names, keywords, and parameter values in call invocation examples.

### Command Invocation of the SCLM Services

The SCLM services can be invoked by using the FLMCMD command in a CLIST or REXX command procedure or by issuing the FLMCMD command as a TSO command.

You cannot invoke the following services using the FLMCMD command:

```
DBACCT
PARSE
END
START
```

## Invoking the SCLM Services

FREE  
STORE  
INIT

## The FLMCMD Interface

The general format for a command invocation is:

```
FLMCMD service_name,project_name,prj_def_name,parameter1,parameter2,...
```

The maximum length of the command invocation statement is 512 characters.

### FLMCMD Parameter Conventions

#### **service\_name**

Alphanumeric; up to 8 characters long.

#### **project\_name**

Alphanumeric; up to 8 characters long.

#### **prj\_def\_name**

Alphanumeric; up to 8 characters long.

The remaining parameters are positional and depend on the service being requested.

Lowercase parameters are optional. If a value is not specified for an optional parameter, SCLM will use default values if they exist. All default values are described within the parameter descriptions for each service.

If you omit a parameter, account for it by inserting a comma in its place. The following example shows how you would omit parm2:

```
FLMCMD service_name,project_name,prj_def_name,parm1,,parm3
```

Do not insert blanks in the command format. Blanks entered before a parameter will cause the value passed to the service to be incorrectly padded with leading blanks.

### Using Command Invocation Variables

If you invoke FLMCMD from a CLIST, you can use a CLIST variable anywhere within a statement as the service name or as a parameter. A CLIST variable consists of a name preceded by an ampersand (&). The CLIST processor replaces each variable with its current value before processing the FLMCMD command.

**Note:** SCLM follows all rules pertaining to TSO CLISTs. For more information, refer to *TSO Extensions Command Language Reference* (SC28-1881) and *TSO Extensions CLISTs* (SC28-1876).

### Using the FLMCMD File Format

Use the FILE format of FLMCMD to process multiple commands as a single command invocation. You can enter the multiple commands either in a data set or from your screen. The FILE format of the command invocation is:

```
FLMCMD FILE[,ddname]
```

The ddname is the data definition name allocated to the FLMCMD command data set. The record length of the command data set cannot exceed 255 bytes. If you do not specify the ddname, SCLM enters interactive mode and prompts you for command lines. For more information, see "Interactive Command Processing" on page 8.

### Performance Considerations

The START service loads the SCLM modules that can be processed into memory and initializes the SCLM service environment. The INIT service loads the load module of a project definition into memory. The FREE service closes all of the open project databases. Each of these functions takes time. Therefore, to optimize the SCLM services execution time, minimize the number of START, INIT, and FREE service calls.

You can reduce the number of START, INIT, and FREE service calls by using the FILE format of FLMCMD. As an SCLM service program, the FLMCMD command processor must call the START service to begin a service session. It must also call the INIT and FREE services for every unique project/prj\_lib\_def combination it encounters. Therefore, ten separate invocations of the FLMCMD command processor result in nine more calls to the START service and nine more calls to the INIT and FREE services than one invocation of the FLMCMD command processor that has all ten commands in a data set.

In addition, opening a command file takes time. In processing a single command, the general format of FLMCMD processes faster than the FILE format of FLMCMD.

SCLM opens the VSAM data sets for a project as they are needed; however each open takes time. Projects can reduce the number of opens required by reducing the number of data sets defined on the FLMCNTRL and FLMALTC macros in the project.

### Command Data Set Conventions

Command data sets use the following conventions:

- The sequence numbers of the command data set should be turned off.
- SCLM processes all commands in the command data set regardless of the success or failure of previous commands.
- Each command must start on a new line.
- If a command takes more than one line, the continuation character should be the first character of the continuation line.

If you enter spaces between the continuation character and the character that follows, those spaces will be treated as part of the parameter.

- If a command line exceeds the maximum record length of the command data set, continue the command by adding a plus sign (the continuation character) in the first position of the continuation line. You can add any number of continuation lines for any command.
- The maximum command length is 512 bytes. Note that if a command consists of several command lines, SCLM deletes trailing blanks.
- An asterisk (\*) indicates comment lines. Place it in the first non-blank character of a command line. You can enter any number of comments within the command data set, but you cannot add a comment line within a series of command continuation lines.

The following example shows a command data set. The first command calls the SCLM LOCK service; the second command calls the SCLM UNLOCK service.

## Invoking the SCLM Services

```
*
* This is an example of a command data set.
*   * Note that comments do not have to start in column 1.
*
* The following command calls the SCLM LOCK service.
LOCK,PROJ1,,USER1,SOURCE,FLM01MD2,TESTAC,XXX#04,USERID
*
* The following command consists of four lines,
* and calls the SCLM UNLOCK service.
UNLOCK,PROJ1,,
+USER1,
+SOURCE,
+FLM01MD2,XXX#04
```

The following example shows a CLIST command procedure that calls the FILE format of FLMCMD.

```
PROC 0
  ALLOC DDNAME(SCLMIN) DA('USERID.FLMCMD.INPUT') SHR
  FLMCMD FILE,SCLMIN
  SET &FLMCMDCC =
  FREE DDNAME(SCLMIN)
  EXIT CODE(&FLMCMDCC)
END
```

### Interactive Command Processing

To use interactive command processing, omit the ddname input parameter when using the FILE format of FLMCMD. You then get a prompt for the Command lines. SCLM processes your input exactly as if the commands were in a command data set. During interactive command processing, you can enter comment lines but you cannot enter continuation lines.

```
| Note: You must perform interactive command processing, like all SCLM
|         processing, from an ISPF environment. Otherwise, the following error
|         message appears:
|         ISPS118 SERVICE NOT INVOKED.  A VALID ISPF ENVIRONMENT DOES NOT EXIST.
```

```
| To end interactive command processing, enter the QUIT command.
```

If you allocate the ddname to your screen and also specify it on the FILE format of FLMCMD, you can get unpredictable results.

Figure 1 on page 9 shows a sample interactive command session.



## Invoking the SCLM Services

Programs in the FLMLNK subroutine interface use the following conventions:

- The `service_name` parameter is positional and required. All other parameters must appear in the order described for each service. Parameter positions on the CALL statement must specify a value up to the last parameter coded. Some services allow for CALLs where the parameter list ends before the last one in the service description, thus taking the default specification for those parameters (see individual service descriptions for details).
- SCLM uses the maximum parameter length when referencing and updating parameter values. Parameter values with fewer characters than the maximum must be padded with blanks for the remainder of the field. Parameters that are not padded with blanks cause unpredictable results. Be sure that all padding is done by inserting trailing blanks. Padding a parameter with leading blanks causes an incorrect value to be passed to the service.
- To omit a parameter, insert a blank enclosed in single quotes ( ' ') in its place.

**Note:** Single quotes show service names and keywords in call invocation examples.

- You must indicate the last parameter in the calling sequence with a '1' as the high-order bit in the last entry of the address list. PL/I, COBOL, Pascal, and FORTRAN call statements automatically generate this high-order bit. In assembler call statements, you must use the VL keyword.

### **FORTRAN, Pascal, and C**

For FORTRAN, Pascal, and C, the general call format for invoking SCLM services from functions by using FLMLNK is:

```
lastrc := FLMLNK(service_name,parameter1,parameter2,...);
```

The parameters for the FORTRAN, Pascal, or C invocation are the same as those shown for the call invocation.

SCLM returns the return code from the specified SCLM service in the FORTRAN, Pascal, or C integer variable specified on the invocation. In these examples, the variable LASTRC is used.

**FORTRAN Example:** For functions written in FORTRAN, pass arguments as FORTRAN variables or literals.

```
INTEGER      LASTRC*4
CHARACTER    SERVIS*8,SCLM_ID*8,GROUP*8
DATA         SERVIS/'DELETE  '/
DATA         SCLM_ID/'SCLM0001'/
DATA         GROUP/'USER1  '/
              .
              .
```

```
LASTRC=FLMLNK(SERVIS,SCLM_ID,GROUP,...)
```

For FORTRAN service requests, initialize parameter variables by using literals in assignment statements. You must use previously-defined constants in assignment statements.

```
CHARACTER    DELET*8,SERVIS*8
DATA         DELET/'DELETE  '/
              .
              .
```

```
SERVIS=DELET
```

**Pascal Example:**

```

CONST
  SERVICE = 'DELETE  ';
  SCLM_ID = 'SCLM0001';
  GROUP   = 'USER1   ';
  :
  .

LASTRC := FLMLNK(SERVICE,SCLM_ID,GROUP,...);

```

For service calls in Pascal, initialize parameter variables by using literals in assignment statements:

```
SERVICE:='DELETE  ';
```

**C Example:** In C programs, include the following declare statements and compiler directives:

```

#pragma linkage(flmlnk,OS);
extern int flmlnk();

```

Example

```

int retcode;
char *SERVICE, *SCLMID,*GROUP, ...;
SERVICE = "DELETE  ";
SCLMID = "SCLM0001";
GROUP = "USER1   ";
  :
  .

lastrc = flmlnk(SERVICE,SCLMID,GROUP,...);

```

### PL/I

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

```

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

```

**PL/I Example::**

```

DECLARE SERVICE CHAR(8) INIT('DELETE  '),
        SCLM_ID CHAR(8) INIT('SCLM0001'),
        GROUP CHAR(8) INIT('USER1   '),
  :
  .

CALL FLMLNK(SERVICE,SCLM_ID,GROUP,...);

```

For service calls in PL/I, initialize parameter variables by using literals in assignment statements:

```
SERVICE='DELETE  ';
```

### COBOL

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

**COBOL Example:**

```

WORKING-STORAGE TYPE.
  77 SERVIS      PICTURE X(8) VALUE 'DELETE  '.
  77 SCLMID      PICTURE X(8) VALUE 'SCLM0001'.

```

## Invoking the SCLM Services

```
77  GROUP          PICTURE X(8) VALUE 'USER1  ' .  
    :  
    :  
PROCEDURE DIVISION  
    CALL 'FLMLNK' USING SERVIS SCLMID GROUP ... .
```

For service calls in COBOL, initialize parameter variables by using literals in assignment statements:

```
MOVE 'DELETE  ' TO SERVIS.
```

## Types of Parameters

The various types of parameters discussed in this section include DDNAME, Character, and Pointer parameters.

### DDNAME Parameters

SCLM services send output to data sets associated with the ddnames you provide in the parameters passed to the service. You should allocate ddnames with the attributes specified in the parameter descriptions. However, if you use different attributes to allocate the ddnames, SCLM accesses the data set using the attributes specified, but the format of the resulting file might not be usable.

As part of the processing for several of its services, SCLM updates partitioned data sets. For instance, the BUILD service copies compiler-produced object modules into an SCLM-controlled object partitioned data set. To eliminate the risk of corrupting a partitioned data set, allocate the data set with DISP=OLD.

### Character Parameters

Left-justify all character input parameters (character strings) to the SCLM services. Left-justify all character output parameters (character strings) from the SCLM services. Make the calling program buffer the length specified in the service descriptions. Failure to provide a buffer of the proper size causes unpredictable results.

### Selection Parameters

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 1 to select accounting information.

Table 1. 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
STAGE1=	STAGE1,STAGE2

**Note:** See *ISPF Software Configuration and Library Manager (SCLM) Developer's and Project Manager's Guide* for an illustration of the hierarchy represented in the last two rows.

### Pointer Parameters

All pointer parameters to the SCLM services provide a fullword address to a predefined array or record structure.

The SCLM services use four pointer parameters:

**\$msg\_array** (message array)  
**\$acct\_info** (accounting information)  
**\$stats\_info** (statistical information)  
**\$list\_info** (list information array)

For Pascal declarations of the services program invocations, see Chapter 3. Sample Programs Using SCLM Services.

**Note:** When creating programs that use SCLM services, the developer must be careful to manipulate the memory for pointer parameters correctly.

#### Input Parameters

The program calling the SCLM service must allocate the memory for the pointer parameter (one word) and the memory for the structure.

#### Output Parameters

The program calling the SCLM service must allocate the memory for only the pointer parameter (one word). If the information in the output structure will be referenced later in the program then the information in the structure must be copied to the program's local storage **before** the next call to an SCLM service. SCLM allocates and deallocates the memory where the output structure is stored.

For example, if you want to pass the \$list\_info array from the PARSE service to the STORE service, you must first copy the \$list\_info array to a local memory buffer. Then you must pass the local buffer pointer to the STORE service.

For examples of copying the \$list\_info array and the \$stats\_info record, see Chapter 3. Sample Programs Using SCLM Services.

### Pointer Parameter Descriptions

This section describes each of the four pointer parameters:

**\$msg\_array:** A pointer to an array of messages SCLM services produce. Each record in the message array is 80 bytes. An END record denotes the end of the message array. Figure 2 on page 14 shows the contents of a message array with one message consisting of two message lines.

## Invoking the SCLM Services

```
Record 1: FLM80500 - ACCESS KEY INCORRECT, ACCESS KEY: WRONG_KEY
Record 2:          GROUP: USER1, TYPE: SOURCE, MEMBER: FLM01MD1
Record 3: END
```

Figure 2. \$msg\_array Contents

**\$acct\_info:** A pointer to a record containing the static portion of an accounting record. The following describes the format of the record fields in the order in which they appear. For additional information on record field contents, refer to the *ISPF Software Configuration and Library Manager (SCLM) Developer's and Project Manager's Guide*.

The following fields contain data common to all members:

Field	Contents
acct_group	8 characters
acct_type	8 characters
acct_member	8 characters
SCLM_version	2 characters ('60 or 70')
accounting_status	1 character: E Editable N Noneditable L Lockout I Initial
change_date	8 characters (YYYYMMDD format)
change_time	6 characters (HHMMSS format)
change_group	8 characters
change_userid	8 characters
-----	3 characters (space for alignment)
member_version	Fullword integer
language	8 characters
authorization_code	8 characters
authorization_code_change	8 characters
access_key	16 characters
creation_date	8 characters (YYYYMMDD format)
creation_time	6 characters (HHMMSS format)
map_date	8 characters (YYYYMMDD format)
map_time	6 characters (HHMMSS format)
predecessor_date	8 characters (YYYYMMDD format)
predecessor_time	6 characters (HHMMSS format)
promote_date	8 characters (YYYYMMDD format)
promote_time	6 characters (HHMMSS format)
promote_userid	8 characters
db_qual	8 characters

The following fields are blank unless the accounting\_status is N. Each field is 8 characters.

- translator\_version
- map\_name
- map\_type
- language\_version

The following fields contain statistical data for a member. Each field is a fullword integer.

- total\_lines
- comment\_lines
- non\_comment\_lines
- blank\_lines
- total\_stmts
- comment\_stmts
- non\_comment\_stmts
- number\_of\_user\_entries
- number\_of\_includes
- reserved\_field
- number\_of\_changeodes
- number\_of\_cus

The fields preceded by an asterisk refer to statistics that the SCLM-supplied parsers do not collect.

**\$stats\_info:** A pointer to a record containing a member's statistical information. Each of the fields is a fullword integer. For a description of the record field contents, see *ISPF Software Configuration and Library Manager (SCLM) Developer's and Project Manager's Guide*. The following describes the format of the record fields.

- total\_lines
- comment\_lines
- non\_comment\_lines
- blank\_lines
- \* prolog\_lines
- total\_stmts
- comment\_stmts
- \* control\_stmts
- \* assignment\_stmts
- non\_comment\_stmts

The fields preceded by an asterisk refer to statistics that the SCLM-supplied parsers do not collect.

**\$list\_info:** A pointer to an array of records containing the dynamic portion of an SCLM accounting record. The array contains records detailing a member's include, change code and user entry information. Each record in the array is 228 bytes.

Some of the SCLM services place restrictions on the data that you can specify with this parameter. See the description of each service to determine if it restricts the \$list\_info parameter data.

The records in the array contain two fields. The first field is 4 characters and indicates the record type. Valid record type values are:

Record Type	Description
END	Indicates the end of the array
INCL	Indicates an include

## Invoking the SCLM Services

Record Type	Description
INCS	Indicates an include with an include-set name
COMP	Indicates the name of an include from the COMPOOL include set
CODE	Indicates a change code
USER	Indicates user data
EXTD	Indicates external dependencies.

The second field varies depending on the record type. For the following discussion, “member” refers to the member whose array contains dynamic accounting record information.

The following table describes the data in the second field for each record type:

Record Type	Description
END	No data
INCL	Member name (8 characters) upon which the “member” has an include dependency.
INCS	<p>A record containing two parts. The first 8 bytes contain the include name; the next 8 bytes contain the include-set name.</p> <p>Include sets are used when different types are to be searched for the includes. For example, an include set of INCLUDE could be used for includes of source code and an include set of SQL could be used for SQL declarations. The include-set name returned by a parser must match the name of an include set in the language definition that included that parser. Include sets are defined using the FLMINCLS macro.</p> <p>Because the include-set name is then associated with a ddname allocation for the translator, there are usually no more include-set names returned by the parser than there are input ddnames supported by the translators in the language definition.</p> <p>Use the INCS record to record dependencies when an include-set name is to be associated with the dependency. Use the INCL record to record a dependency when the dependency is to be associated with the default include set. Do not use both INCL and INCS records for the same dependency name unless two different include dependencies are to be recorded for the same member name.</p> <p>An INCS record with blanks for the include-set name is the same as an INCL record for that dependency.</p>
COMP	Indicates an include in the COMPOOL include-set \$list_info entries returned by SCLM will always use the INCS record type to return information for includes in the COMPOOL include set. The preferred method of recording dependencies in the COMPOOL include set is to use INCS records. This record type is available for compatibility purposes only.
CODE	A record detailing a change code associated with the “member”. The total record length is 22 bytes. The record contains a change code (8 characters), a change code date stamp (8 characters, YYYYMMDD format), and a change code time stamp (6 characters, HHMMSS format). The change code value will be translated to upper case before it is passed to the SCLM service.
USER	User data (128 characters) associated with the “member”.

Record Type	Description
EXTD	A record that describes an external dependency for an SCLM-controlled member. This record contains the following information: <ul style="list-style-type: none"> <li><b>group</b> Name of the SCLM group that is equivalent to the group where the external dependency resides (8 characters)</li> <li><b>type</b> Name of type (8 characters)</li> <li><b>name</b> Name of the external dependency (43 characters)</li> <li><b>date/time</b> Date and time in SCLM format when the external dependency was last changed (14 characters: date in format YYYYMMDD, time in format HHMMSS).</li> </ul>

**Note:**

The SAVE service restricts the \$list\_info record type to CODE and END. SCLM deletes all existing user data records if you use the SAVE service.

Figure 3 shows the contents of a list information array. Two change codes (PR1234 on 12/16/93 at 12:01:33 and CR000032 on 1/4/94 at 00:53:16) and a user entry indicating a customized member are associated with the “member”.

```

| Record 1: CODEPR1234 19931216120133
| Record 2: CODECR00003219940104005316
| Record 3: USERTEST MEMBER - CUSTOMIZED
| Record 4: END

```

Figure 3. \$list\_info Contents

## ISPF Variables

Some SCLM services use ISPF variables to communicate information with the caller. All variables contain character data. Integer data is converted to character format. The following table lists the ISPF variables which are used:

Variable	Max Size	Services	Description
zlockdsn	44	LOCK	Data set name for the member at the lock group
zsaackey	16	VERINFO, ACCTINFO	Access key (see lock and unlock services)
zsaastmt	8	VERINFO, ACCTINFO	Parser statistic - number of assignment statements
zsaauth	8	VERINFO, ACCTINFO	Authorization code
zsaauthc	8	VERINFO, ACCTINFO	Authorization code change
zsabdate	8	VERINFO, ACCTINFO	Baseline (predecessor) date
zsabdat4	10	VERINFO, ACCTINFO	Baseline (predecessor) date with a 4-character year
zsabline	8	VERINFO, ACCTINFO	Parser statistic - number of blank lines

## Invoking the SCLM Services

Variable	Max Size	Services	Description
zsabtime	8	VERINFO, ACCTINFO	Baseline (predecessor) time
zsacctnt	8	VERINFO, ACCTINFO	Number of change codes for the member
zsacdate	8	VERINFO, ACCTINFO	SCLM creation date
zsacdat4	10	VERINFO, ACCTINFO	SCLM creation date with 4-character year
zsacline	8	VERINFO, ACCTINFO	Parser statistic - number of comment lines
zsacstmt	8	VERINFO, ACCTINFO	Parser statistic - number of comment statements
zsactime	8	VERINFO, ACCTINFO	SCLM creation time
zsacucnt	8	VERINFO, ACCTINFO	Number of ADA Compilation units
zsadsn	44	VERINFO, ACCTINFO	Physical data set name for the group and type
zsagrp	8	ACCTINFO	SCLM group name
zsaincnt	8	VERINFO, ACCTINFO	Number of includes for the member
zsalang	8	VERINFO, ACCTINFO	SCLM language name for the member
zsaldate	8	ACCTINFO	Date the member was last changed
zsaldat4	10	ACCTINFO	Date, with a 4-character year, that the member was last changed
zsalgrp	8	VERINFO, ACCTINFO	Group where the member was last changed
zsalstmt	8	VERINFO, ACCTINFO	Parser statistic - number of control statements
zsaltime	8	ACCTINFO	Time the member was last changed
zsaluser	8	VERINFO, ACCTINFO	Userid that last changed the member
zsambr	8	ACCTINFO	SCLM member name
zsamdate	8	VERINFO, ACCTINFO	Date of the build map that generated the member or if the member is not generated this is the last change date
zsamdate	10	VERINFO, ACCTINFO	Date, with a 4-character year, of the build map that generated the member or if the member is not generated this is the last change date
zsamnbr	8	VERINFO, ACCTINFO	Name of the build map that generated the member or blank if not a generated member
zsamtime	8	VERINFO, ACCTINFO	Time of the build map that generated the member or if the member is not generated this is the last change time
zsamtver	8	VERINFO, ACCTINFO	Version of the translator that generated the member

## Invoking the SCLM Services

Variable	Max Size	Services	Description
zsamtype	8	VERINFO, ACCTINFO	Type of the build map that generated the member or blank if not a generated member
zsanline	8	VERINFO, ACCTINFO	Parser statistic - number of non-comment lines
zsanstmt	8	VERINFO, ACCTINFO	Parser statistic - number of non-comment statements
zsapdate	8	VERINFO, ACCTINFO	Date the member was promoted from a lower group or zeros if no promote was done to get the member into the current group
zsapdat4	10	VERINFO, ACCTINFO	Date, with a 4-character year, the member was promoted from a lower group or zeros if no promote was done to get the member into the current group
zsapline	8	VERINFO, ACCTINFO	Parser statistic - number of prolog lines
zsaptime	8	VERINFO, ACCTINFO	Time the member was promoted from a lower group or zeros if no promote was done to get the member into the current group
zsapuser	8	VERINFO, ACCTINFO	Userid last promoting the member from a lower group or blank if no promote was done to get the member into the current group
zsastat	8	VERINFO, ACCTINFO	Status of the accounting record. Possible values are: EDITABLE, NON-EDIT, LOCKOUT, INITIAL, and ERROR
zsatline	8	VERINFO, ACCTINFO	Parser statistic - total number of lines
zsatstmt	8	VERINFO, ACCTINFO	Parser statistic - total number of statements
zsatype	8	ACCTINFO	SCLM Type
zsauecnt	8	VERINFO, ACCTINFO	Number of user entries for the member
zsaver	8	VERINFO, ACCTINFO	Version number of the member
zsctime	8	VERINFO, ACCTINFO	Last time a change code was assigned to a member
zsdname	110	VERINFO, ACCTINFO	Name of an ADA compilation unit
zsdtype	8	VERINFO, ACCTINFO	Type of an ADA compilation unit. Possible values are: SPEC, BODY, and XREF
zsiiset	8	VERINFO, ACCTINFO	Include-set name for an include
zsimbr	8	VERINFO, ACCTINFO	An include for a member
zsuentry	128	VERINFO, ACCTINFO	Data from a user entry
zsunum	8	VERINFO, ACCTINFO	Number of the user entry

## Invoking the SCLM Services

Variable	Max Size	Services	Description
zsvactn	8	VERINFO	Action generating the audit record. Possible values are: PUT and PURGE
zsvambr	8	VERINFO	SCLM member name for action taken
zsvcfmt	8	VERINFO	Current format of the version member. Possible values are: DELTA, FULL, and AUDIT
zsvdate	8	VERINFO	Date the audit record was generated.
zsvdat4	10	VERINFO	Date, with a 4-character year, that the audit record was generated.
zsvfmsg	8	VERINFO	SCLM message id if versioning of the member failed.
zsvgrp	8	VERINFO	SCLM group name for action taken
zsvldate	8	VERINFO	Last change date of the member
zsvldat4	10	VERINFO	Last change date, with 4-character year, of the member
zsvltime	8	VERINFO	Last change time of the member
zsvmbr	8	VERINFO	Member in the versioning pds containing the version of the member or blank if only auditing was performed
zsvpds	44	VERINFO	The versioning pds containing the version of the member
zsvreslt	8	VERINFO	Result of the versioning action. Possible values are: ATTEMPT, COMPLETE, and FAILED
zsvrfmt	8	VERINFO	Requested format of the version. Possible values are: DELTA, FULL, and AUDIT
zsvsdate	8	VERINFO	SCLM change date for the member for which a version was requested.
zsvsdat4	10	VERINFO	SCLM change date, with 4-character year, for the member for which a version was requested.
zsvserv	8	VERINFO	SCLM service generating the audit record. Possible values are: BLDDDEL, BUILD, DELETE, FREE, IMPORT, LOCK, EXT LIB, PROMOTE, STORE, UPTATHCD, UPTCHGCD, UNLOCK, and UPTUENTY
zsvstime	8	VERINFO	SCLM change time of the member.
zsvtime	11	VERINFO	Time the audit record was generated.
zsvtype	8	VERINFO	SCLM type for action taken
zsvuser	8	VERINFO	Userid performing the service which generated the audit record.

## SCLM Service Return Codes

Each service returns a numeric code, called a *return code*, indicating the results of the operation. The following are possible return codes:

- 0 Indicates successful completion. SCLM may generate messages.
- 2 Indicates successful completion. No action taken.
- 4 Indicates a warning condition. SCLM may generate messages.

- 8 Indicates an error condition. SCLM generates messages detailing the error.
- 12 Indicates a severe error condition. SCLM generates messages detailing the error.

Return codes and their meanings vary for each service and are listed with each service description. In addition to these return codes, the `FLMCMD` and `FLMLNK` interfaces each generate return codes.

For command invocation, SCLM returns the code in the `CLIST` variable. For call invocation, SCLM returns the code in registers 15 and 0. When using the `FILE` format of `FLMCMD` command invocation, SCLM sets the return code to the maximum return code encountered while processing the command data set.

Programs coded in Pascal or FORTRAN can examine the return code by using an integer variable, such as `lastrc`, in the following example:

```
lastrc := FLMLNK(service_name,parameter1,parameter2,...);
```

Programs coded in PL/I can examine the return code by using `PLIRETV`, a built-in function. You need the following declare statements:

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

Programs coded in COBOL can examine the return code by using `RETURN-CODE`, a built-in variable.

---

### FLMCMD Command Processor Return Codes

Possible return codes are:

- 12 Maximum application ID limit exceeded. `FLMCMD` has attempted to initialize an SCLM session, but the maximum number of SCLM sessions have already been started. End one or more of the active sessions and reissue the command.
- 16 The SCLM table verification failed. The version of the SCLM project definition macros used to compile the specified project definition does not match the version of SCLM being used. Verify that the project definition specified in the line command is correct. If the project definition was specified correctly, contact the project administrator.
- 20 The NLS table verification failed. The version of the NLS table did not match the version of SCLM being used. Contact the project administrator.
- 24 Unable to load the SCLM table (`FLMTABLE`). Contact the project administrator.
- 28 Unable to load the NLS table or the SCLM I/O load module (`FLMIO24`). Contact the project administrator.

---

### FLMLNK Call Processor Return Codes

Possible return codes are:

- 20 Severe error condition. SCLM does not produce messages because the SCLM ID is not valid.
- 24 Severe error condition. SCLM does not produce messages because the

## Invoking the SCLM Services

- | SCLM services have not been initialized. See “START—Generate an  
| Application ID for a Services Session” on page 87 for information about  
| initializing an SCLM services session.
- | 32 Severe error condition. An invalid parameter list was passed to the  
| requested service.
- | 34 Severe error condition. An invalid service was requested.
- | 36 Severe error condition. The version of the FLMLNK subroutine does not  
| match the version of the SCLM services module.
- | Return codes and their meanings vary for each service and are listed with each  
| service description.

---

## SCLM Service Messages

SCLM services issue two types of messages:

### FLMMSG

SCLM uses the ddname FLMMSG for special services messages such as a completion status or return code message, and for error messages associated with the specified service parameters. These messages are usually routed to the default output device, such as your terminal. In order to suppress or re-route these messages, allocate the FLMMSG ddname before invoking the SCLM service.

### Service Specific Messages

Many of the services have parameters for handling messages. There are three types of message parameters:

#### **msg\_line**

Services that only write one message have a **msg\_line** parameter. Define a program variable that is 80 characters to hold the contents of this message line. This parameter only applies to services called through the FLMLNK interface.

#### **\$msg\_array**

Some services that can produce more than one message have a **\$msg\_array** parameter. Define program storage as described in “DDNAME Parameters” on page 12 to store the service messages. The \$msg\_array is available only from services invoked through the FLMLNK interface.

#### **ddname**

Many of the services offer a **ddname** parameter which you can allocate to a file that stores the messages. Information for allocating the ddname is included in the description for each applicable service. If you leave the ddname parameter blank, the messages go to the default output device, for example, your terminal.

## SCLM Service Descriptions

This section contains information about the services available for SCLM.

Table 2. Services

Service	Description	Page
ACCTINFO	Retrieve accounting information	"ACCTINFO—Retrieve Accounting Information" on page 25
AUTHCODE	Retrieve or set authorization code for selected members	"AUTHCODE—Retrieve or Set Authorization Code for Selected Members" on page 28
BUILD	Build a member	"BUILD—Build a Member" on page 32
DBACCT	Retrieve accounting records for a member	"DBACCT—Retrieve Accounting Records for a Member" on page 36
DBUTIL	Generate a tailored output data set and report	"DBUTIL—Generate a Tailored Output Data Set and Report" on page 38
DELETE	Delete database components	"DELETE—Delete Database Components" on page 42
DELGROUP	Delete group database components	"DELGROUP—Delete Group Database Components" on page 44
DSALLOC	Allocate data sets for group or type	"DSALLOC—Allocate Data Sets for Group/Type" on page 48
EDIT	Edit a member of a controlled library	"EDIT— Edit a Member of a Controlled Library" on page 51
END	End an SCLM services session	"END— End an SCLM Services Session" on page 54
EXPORT	Extract SCLM accounting information for a group	"EXPORT—Extract SCLM Accounting Information for a Group" on page 55
FREE	Free and SCLM ID	"FREE—Free an SCLM ID" on page 58
IMPORT	Import SCLM accounting information to current project	"IMPORT—Import SCLM Accounting Information to Current Project" on page 59

## SCLM Service Descriptions

Table 2. Services (continued)

Service	Description	Page
INIT	Generate an SCLM ID	"INIT—Generate an SCLM ID" on page 62
LOCK	Lock a member or assign an access key	"LOCK—Lock a Member or Assign an Access Key" on page 63
MIGRATE	Create accounting for selected members	"MIGRATE—Create Accounting for Selected Members" on page 68
NEXTGRP	Returns the name of the next group in a given hierarchy.	"NEXTGRP—Retrieve Next Group in SCLM Hierarchy" on page 71
PARSE	Parse a member for statistical and dependency information	"PARSE—Parse a Member for Statistical and Dependency Information" on page 73
PROMOTE	Promote a member from one library to another	"PROMOTE—Promote a Member from One Library to Another" on page 76
RPTARCH	Generate an SCLM architecture report	"RPTARCH—Generate an SCLM Architecture Report" on page 80
SAVE	Lock, parse, and store a member	"SAVE—Lock, Parse, and Store a Member" on page 83
START	Generate an application ID for a services session	"START—Generate an Application ID for a Services Session" on page 87
STORE	Store member information in an accounting record	"STORE—Store Member Information in an Accounting Record" on page 89
UNLOCK	Unlock a member in a development library	"UNLOCK—Unlock a Member in a Development Library" on page 92
VERDEL	Delete version/audit information	"VERDEL—Delete Version and Audit Information" on page 95
VERINFO	Retrieve version/audit information	"VERINFO—Retrieve Version and Audit Information" on page 97
VERRECOV	Recover a version	"VERRECOV—Recover a Version" on page 100

Each service description consists of the following information:

- Description** A description of the function and operation of the service. This description also refers to other services that you can use with this service.
- Each service description shows the formats for:
- Command invocation, for use in a CLIST or REXX command procedure or as a TSO command
  - Call invocation from a program module.
- Format** The syntax that you use to code the service, showing both command invocation and call invocation.
- Because this chapter shows command and call invocation formats in Pascal, a semicolon (;) ends statements. This is a Pascal convention, but you should use the syntax appropriate for your programming language.
- Parameters** A description of any required or optional keywords or parameters. For additional information on parameters, refer to *ISPF Software Configuration and Library Manager (SCLM) Developer's and Project Manager's Guide*.
- Return Codes** A description of the codes the service returns. For all services, a return code of 12 or higher implies a severe error. This error is usually a syntax error, but it can be any severe error detected when using the services.
- Examples** Sample usage of the service.

FLMLNK requires that the parameters be padded with blanks if the value specified is not as long as the maximum length allowed. Therefore, the examples of call invocations are padded with blanks to the maximum length allowed for each parameter.

---

## ACCTINFO—Retrieve Accounting Information

The ACCTINFO service retrieves the information about an SCLM controlled member into ISPF variables and tables. The information is retrieved from the accounting file defined in the project definition for the group specified to the service. The service can search up the hierarchy for the member, search a group for the next matching member, or retrieve the information for a specific member. See “ISPF Variables” on page 17 for a list of the variables updated by this service.

### Command Invocation Format

```
FLMCM ACCTINFO,project
           ,[prj_def]
           ,group
           ,type
           ,member
           ,[user_info_table]
           ,[include_table]
           ,[change_code_table]
           ,[ada_cu_table]
           ,[SEARCH|FORWARD|MATCH]
           ,[dd_msgs]
```

## Call Invocation Format

```
lastrc := FLMLNK('ACCTINFO ',sclm_id,
                ,group
                ,type
                ,member
                ,user_info_table
                ,include_table
                ,change_code_table
                ,ada_cu_table
                ,SEARCH|FORWARD|MATCH
                ,msg_array);
```

## Parameters

### project

The project name. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

### prj\_def

The project definition name. It defaults to the project name. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

### sclm\_id

An SCLM ID associated with a given project and project definition. The INIT service generates the SCLM ID. The maximum parameter length is 8 characters. This parameter is used for FLMLNK only.

### group

The group associated with the accounting record. The maximum parameter length is 8 characters.

### type

The type associated with the accounting record. The maximum parameter length is 8 characters.

### member

The member under SCLM control. The maximum parameter length is 8 characters.

### user\_info\_table

The name of the ISPF table to contain the user entries from the account record. The table must be open prior to calling the ACCTINFO service. A TBADD will be performed for each user entry in the account record. The maximum parameter length is 8 characters. The following ISPF variables must be used in the table definition in order to have their value stored in the table:

- ZSUNUM - the user entry number
- ZSUENTRY - the user entry data

### include\_table

The name of the ISPF table to contain the includes from the account record. The table must be open prior to calling the ACCTINFO service. A TBADD will be performed for each include in the account record. The maximum parameter length is 8 characters. The following ISPF variables must be used in the table definition in order to have their value stored in the table:

- ZSIMBR - the include member name
- ZSISET - the include set name

### change\_code\_table

The name of the ISPF table to contain the change codes from the account record. The table must be open prior to calling the ACCTINFO service. A TBADD will be performed for each change code in the account record. The

maximum parameter length is 8 characters. The following ISPF variables must be used in the table definition in order to have their value stored in the table:

- ZSCCODE - the change code
- ZSCDATE - the change code date
- ZSCDAT4 - the change code date in 4-character date format
- ZSCTIME - the change code time

#### **ada\_cu\_table**

The name of the ISPF table to contain the ADA compilation units from the account record. The table must be open prior to calling the ACCTINFO service. A TBADD will be performed for each ADA compilation unit in the account record. The maximum parameter length is 8 characters. The following ISPF variables must be used in the table definition in order to have their value stored in the table:

- ZSDNAME- the ADA compilation unit name
- ZSDTYPE - the ADA compilation unit type

#### **SEARCH|FORWARD|MATCH**

SEARCH indicates that SCLM is to look up the hierarchy to find the accounting record if it does not exist at the specified group. This is the default.

MATCH indicates to check only the specified group for the accounting record.

FORWARD indicates that if the member and type names do not exactly match an accounting record the information from the next accounting record in the group is to be returned.

To retrieve all of the accounting records within a group use FORWARD and start with the member and type names set to all blanks. If an accounting record is found increment the last character of the member name before calling the ACCTINFO service again. Repeat this process until the service indicates that no record was found.

The maximum parameter length is 8 characters.

#### **dd\_msgs**

The ddname indicating the destination of the messages generated by the ACCTINFO service. If you specify a blank ddname, SCLM routes the ACCTINFO messages to the default output device, such as your terminal. Otherwise, before you call the ACCTINFO service, you must allocate the ddname. The following attributes should be used: RECFM=F, LRECL=80, BLKSIZE=80. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

#### **\$msg\_array**

An output parameter pointing to the message array. See "Pointer Parameters" on page 13 for more information on \$msg\_array. This parameter is used for FLMLNK only.

## **Return Codes**

Additional special services messages are written to the FLMMMSGs ddname. See "SCLM Service Messages" on page 22 for more information.

Other return codes might be produced by the FLMCMD or the FLMLNK processor. See "SCLM Service Return Codes" on page 20 for more information.

## ACCTINFO Service

Possible return codes are:

- 0 Normal completion. An account record exactly matching the specified criteria was found and the information was stored successfully.
- 8 Error completion. No account record was found for the specified member.
  - If FORWARD was specified, then there are no accounting records for the group which match or follow the specified type and member name.
  - If MATCH was specified, then there is not an account record with the specified group, type and member name.
  - If SEARCH was specified, then there are no matching account records found when searching up the hierarchy starting from the specified group.
- 12 Error completion. Refer to the messages for more information.

---

## AUTHCODE—Retrieve or Set Authorization Code for Selected Members

The AUTHCODE service changes or retrieves the authorization code in the SCLM accounting information for members in a library that match a given pattern. The AUTHCODE service does not change the member's statistics or any other value in the accounting record, including the change date and time.

The AUTHCODE service can either set all authorization codes that match a given member and type pattern, or set only those authorization codes that also already have a particular authorization code.

To set the authorization code for all members that match a pattern, leave the **from\_authcode** parameter blank.

If only members with a certain authorization code are to be set, use the **from\_authcode** parameter to tell SCLM to change only those members with the given authorization code.

To retrieve the authorization code, leave both the **from\_authcode** and the **to\_authcode** parameters blank. The existing authorization code is returned in variable **ZSAAUTH** if a single member is requested. If a pattern is requested, the existing authorization codes can be retrieved from the AUTHCODE report.

## Command Invocation Format

```
FLMCMD AUTHCODE,project
           ,[prj_def]
           ,group
           ,type
           ,member
           ,[from_authcode]
           ,[to_authcode]
           ,[C|U]
           ,[dd_authmsgs]
           ,[dd_authrept]
```

## Call Invocation Format

```
|          lastrc := FLMLNK('AUTHCODE',sclm_id,
|                      ,group
|                      ,type
|                      ,member
|                      ,from_authcode
```

```

, to_authcode
, C|U
[, dd_authmsgs
[, dd_authrept]]);

```

## Parameters

### project

The project name. The maximum parameter length is 8 characters.

### prj\_def

The project definition name. It defaults to the project name. The maximum parameter length is 8 characters.

### sclm\_id

An SCLM ID associated with a given project and project definition. The INIT service generates the SCLM ID. The maximum parameter length is 8 characters.

### group

The group at which the member's authcode is to be changed. The maximum parameter length is 8 characters.

### type

A pattern used to select the types of members whose authcode is to be changed. The maximum parameter length is 10 characters.

### member

A pattern used to select the members whose authcode is to be changed. You must specify a valid member name or a valid pattern. The maximum parameter length is 10 characters.

### C|U

Indicates whether the AUTHCODE service is to execute conditionally (C) or unconditionally (U). This parameter only applies if the member name is a pattern. If C is selected, processing stops after the first error (default). If U is selected, the service continues to the next member even if an error occurs.

### from\_authcode

The authorization code to be changed from. If the from\_authcode is blank and the to\_authcode is given, then all members matching the pattern have the authcode updated. If the from\_authcode is not blank, only those members matching the pattern, and whose authcode matches the from\_authcode, are updated. The maximum parameter length is 8 characters.

### to\_authcode

The authorization code to be changed to. If the to\_authcode and the from\_authcode are both blank, no changes are made. If the from\_authcode is given, then the to\_authcode is required. The maximum parameter length is 8 characters.

### dd\_authmsgs

DDNAME of the destination of the AUTHCODE messages. If you specify a blank ddname, SCLM routes the authcode messages to the default output device, such as your terminal. Otherwise, before you call the AUTHCODE service, you must allocate the ddname. The following attributes should be used:

- DISP=MOD
- RECFM=F
- LRECL=80
- BLKSIZE=80.

## AUTHCODE Service

The maximum parameter length is 8 characters.

### **dd\_authrept**

DDNAME of the destination of the AUTHCODE report. If you specify a blank ddname, SCLM routes the authcode report to the default output device, such as your terminal. Otherwise, before you call the AUTHCODE service, you must allocate the ddname. The following attributes should be used:

- RECFM=FBA
- LRECL=80
- BLKSIZE=3120.

The maximum parameter length is 8 characters.

## Return Codes

Other return codes might be produced by the FLMCMD or the FLMLNK processor. See "SCLM Service Return Codes" on page 20 for more information.

Possible return codes are:

- 0 Normal completion. Authcode changed or reported successfully.
- 2 Normal completion. Authcode not changed. One of the following occurred:
  - To\_authcode = existing authcode (no change needed)
  - From\_authcode requested does not equal existing authcode (no change wanted)
  - Member is not editable.
- 4 Warning condition. Segment exists at a lower level with an authcode not equal to the **to\_authcode**, which could overlay the current segment.
- 8 Error condition. Invalid type, member, or mode parameter. See the dd\_authmsgs for details.
- 12 Severe error condition. Accounting record not found or severe error.
- 16 Severe error condition. One of the following occurred:
  - Not authorized to update **to\_authcode**, access\_key mismatch, or not authorized to update data set.
  - Verification failed.
  - Error updating accounting record.
  - Invalid group.

SCLM might not produce messages because there was an error invoking the AUTHCODE module.

## Examples

### **Command Invocation Format**

This example shows a command interface to the AUTHCODE service, to update the authorization code of SCLM70.USER.SOURCE(A) from base to private.

```
FLMCMD AUTHCODE,SCLM70,SCLM7010,USER,SOURCE,A,BASE,PRIVATE
```

This example shows a command interface to the AUTHCODE service to unconditionally update the authorization code from base to private for all members beginning with FLM in all types of group USER in project SCLM70.

```
FLMCMD AUTHCODE,SCLM70,SCLM7010,USER,*,FLM*,BASE,PRIVATE,U
```

This example selects the FLMCMD AUTHCODE service with no from\_authcode or to\_authcode. It then gets the authcode value from variable ZSAAUTH in the ISPF SHARED pool.

```
/* rexx exec to retrieve an authcode */
PARMS = 'SCLM70,SCLM7010,USER,SOURCE'
MEM = 'BES3 '
address ispxexec 'select cmd(FLMCMD AUTHCODE,'PARMS','MEM')'
address ispxexec 'vget zsaauth shared'
say 'authcode is' zsaauth
```

**Program Invocation Format**

This example shows a program call to the AUTHCODE service. The example assumes that the START and INIT services have already completed successfully.

```
CALL FLMLNK('AUTHCODE ',SLMID,'USER ', 'SOURCE ', 'A ',
           'BASE ', 'PRIVATE ', 'C ', 'MSGDD',
           REPTDD) RETCODE(R15);
```

This example shows a program call to the AUTHCODE service to unconditionally update the authorization code from 'base' to 'private' for all members beginning with FLM in all types of group USER in project SCLM70.

```
CALL FLMLNK('AUTHCODE ',SLMID,'USER ', '* ', 'FLM* ',
           ' ', 'PRIVATE ', 'U ', 'MSGDD',
           REPTDD) RETCODE(R15);
```

**Example of an AUTHCODE Report**

```
*****
*****
**
**
**              SOFTWARE CONFIGURATION AND LIBRARY MANAGER (SCLM)
**
**              AUTHCODE UPDATE REPORT
**
**              1999/03/08   11:02:15
**
**              PROJECT:      PDFTDEV
**              ALTERNATE:    PDFTNEWL
**              GROUP:        BETH
**              TYPE:         ARCHDEF
**              MEMBER:       B*
**              FROM_AUTHCODE: PRIVATE
**              TO_AUTHCODE:  BASE
**              MODE:         UNCONDITIONAL
*****
*****
MEMBER      TYPE      STARTING   ENDING    COMPLETION
-----      -
BETHTEST   ARCHDEF   BASE      BASE      NOT_ATTEMPTED
BETH7      ARCHDEF   PRIVATE   BASE      SUCCEEDED
BETH8      ARCHDEF   PRIVATE   BASE      SUCCEEDED
BETH9      ARCHDEF   PRIVATE   BASE      FAILED
BROKE      ARCHDEF   BASE      BASE      NOT_ATTEMPTED
BXC        ARCHDEF   BASE      BASE      NOT_EDITABLE
```

### BUILD—Build a Member

The BUILD service compiles, links, and integrates software components according to a project's architecture definition. Before a member is built, the member's dependency information must exist in the project database. For this reason, either the STORE or SAVE service must complete successfully for the member before you call the BUILD service.

#### Command Invocation Format

```
FLMCMD BUILD,project
           ,[prj_def]
           ,group
           ,type
           ,member
           ,[userid]
           ,[E|L|N|S]
           ,[C|F|R|U]
           ,[Y|N]
           ,[Y|N]
           ,[prefix_userid]
           ,[dd_bldmsgs]
           ,[dd_bldrept]
           ,[dd_bldlist]
           ,[dd_bldexit]
```

## Call Invocation Format

```

lastrc := FLMLNK('BUILD ' ,sclm_id
                ,group
                ,type
                ,member
                ,{userid|' '}
                ,{E|L|N|S}
                ,{C|F|R|U}
                ,{Y|N}
                ,{Y|N}
                ,[{prefix_userid|' '}
                ,[dd_bldmsgs
                ,[dd_bldrept
                ,[dd_bldlist
                ,[dd_bldexit]]]]);

```

## Parameters

### project

The project name. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

### prj\_def

The project definition name used for the build. It defaults to the project parameter. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

### sclm\_id

An SCLM ID associated with a given project and project definition. The SCLM ID is generated by the INIT service. The maximum parameter length is 8 characters. This parameter is used for FLMLNK only.

### group

The group in which the build occurs. The maximum parameter length is 8 characters.

### type

The type containing the member to be built. The maximum parameter length is 8 characters.

### member

The member to be built. The maximum parameter length is 8 characters.

### userid

The user ID of the person requesting the build. If no value is specified for FLMCMD or a blank ( ' ') is specified for FLMLNK, it defaults to your TSO prefix or user ID if no TSO prefix has been created. The maximum parameter length is 8 characters.

## BUILD Service

### EILNLS

Indicates the build scope (E=extended, L=limited, N=normal, S=subunit). For the FLMCMD interface, the default is N. There is no default for FLMLNK. The maximum parameter length is 24 characters.

### CIFRIU

Indicates the build mode (C=conditional, F=forced, R=report, U=unconditional). For FLMCMD, the default is C. There is no default for FLMLNK. The maximum parameter length is 24 characters.

### YIN

Y indicates that translator listings are to be copied to the dd\_bldlist ddname only if errors occur. N indicates that all translator listings are to be copied to the dd\_bldlist ddname. For FLMCMD, the default is Y. There is no default for FLMLNK. The maximum parameter length is 24 characters.

### YIN

Y indicates that a build report is to be produced and routed to the dd\_bldrept ddname. N indicates that a build report is not to be produced. For FLMCMD, the default is Y. There is no default for FLMLNK. The maximum parameter length is 24 characters.

### prefix\_userid

The data set name prefix to be used when locating and cataloging temporary data sets. If no value is specified for FLMCMD or a blank ( ' ') is specified for FLMLNK, it defaults to the user ID parameter. The maximum parameter length is 17 characters.

### dd\_bldmsgs

The ddname indicating the destination of the build messages. If you specify a blank ddname, SCLM routes the build messages to the default output device, such as your terminal. Otherwise, before you call the BUILD service, you must allocate the ddname; the following attributes should be used: RECFM=F, LRECL=80, BLKSIZE=80. You cannot specify a blank ddname for FLMLNK. The maximum parameter length is 8 characters.

### dd\_bldrept

The ddname indicating the destination of the build report. If you specify a blank ddname, SCLM routes the build report to the default output device, such as your terminal. Otherwise, before you call the BUILD service, you must allocate the ddname; the following attributes should be used: RECFM=FBA, LRECL=80, BLKSIZE=3120. The maximum parameter length is 8 characters.

### dd\_bldlist

The ddname indicating the destination of the build listings. If you specify a blank ddname, SCLM does not generate the build listings. Otherwise, before you call the BUILD service, you must allocate the ddname; the following attributes should be used: DISP=MOD, RECFM=VBA, LRECL=137, BLKSIZE=3120. The maximum parameter length is 8 characters.

### dd\_bldexit

The ddname indicating the destination of the build user exit data. Specify this parameter only if your project definition defines a build user exit routine. Ask your project manager if your project is using a build user exit routine. If you specify a blank ddname, SCLM routes the build user exit data to NULLFILE. Otherwise, before you call the BUILD service, you must allocate the ddname; the following attributes should be used: RECFM=FB, LRECL=160, BLKSIZE=3200. The maximum parameter length is 8 characters.

## Return Codes

Additional special services messages are written to the FLMMMSGs ddname. See “SCLM Service Messages” on page 22 for more information.

Other return codes might be produced by the FLMCMD or the FLMLNK processor. See “SCLM Service Return Codes” on page 20 for more information.

Possible return codes are:

- 0 Normal completion. See the SCLM messages for more information.
- 4 Warning condition. See the SCLM messages for more information.
- 8 Error condition. See the SCLM messages for more information.
- 12 Severe error condition. SCLM does not produce messages because there was an error invoking the build module.
- 16 Severe error condition. SCLM does not produce messages because it was unable to retrieve SCLM ID information.

## Examples

These examples call the BUILD service.

### Command Invocation

```
FLMCMD BUILD,PROJ1,,USER1,ARCHDEF,FLM01CMD,,U,,N
```

This service command builds the FLM01CMD member of the ARCHDEF type in the USER1 group. The project name is PROJ1. The build mode is unconditional and SCLM does not generate a build report. SCLM sends messages and listings to the terminal. All other parameters are defaults.

### Call Invocation

**Note:** This example shows general syntax. Call invocations are language-specific. See “Chapter 3. Sample Programs Using SCLM Services” on page 105 for specific examples.

```
lastrc := FLMLNK('BUILD ', (* service *)
                 sclm_id, (* SCLM ID *)
                 'USER1 ', (* group *)
                 'ARCHDEF ', (* type *)
                 'FLM01CMD', (* member *)
                 ', ', (* user ID *)
                 'N ', (* scope *)
                 'F ', (* mode *)
                 'N ', (* listings *)
                 'Y ', (* report *)
                 'PROJECT.WORKFILE ', (* temp high-level qualifier *)
                 'BLDMSGs ', (* dest. of messages *)
                 'BLDREPT ', (* dest. of report *)
                 'BLDLIST ', (* dest. of listings *)
                 'BLDEXIT '); (* exit routine *)
```

The service call builds the FLM01CMD member of the ARCHDEF type in the USER1 group. The sclm\_id parameter contains a valid SCLM ID returned from the INIT service. The build scope is normal and the build mode is forced. SCLM copies all build listings to the build listings data set and generates a build report. All temporary data sets are allocated with the high-level qualifier of

## BUILD Service

PROJECT.WORKFILE. The ddnames for the messages, report, listings, and user exit data set (BLDMSG, BLDREPT, BLDLIST, and BLDEXIT, respectively) must be allocated before calling FLMLNK.

---

## DBACCT—Retrieve Accounting Records for a Member

The DBACCT service retrieves accounting records from the project database and returns the information to you. SCLM retrieves the first occurrence of the accounting record in the hierarchy, starting at the specified group. Accounting records exist for any member for which the LOCK, SAVE, or STORE service completes successfully. For more information on SCLM accounting records, see “\$acct\_info” on page 14.

### Command Invocation Format

You cannot use command procedures to call this service.

### Call Invocation Format

```
lastrc := FLMLNK('DBACCT ',sclm_id
                ,group
                ,type
                ,member
                ,found_group
                ,$acct_info
                ,$list_info
                ,$msg_array);
```

### Parameters

#### sclm\_id

An SCLM ID associated with a given project and project definition. The SCLM ID is generated by the INIT service. The maximum parameter length is 8 characters.

#### group

The group in which the accounting record search begins. The maximum parameter length is 8 characters.

#### type

The type containing the accounting record to retrieve. The maximum parameter length is 8 characters.

#### member

The member whose accounting record is to be retrieved. The maximum parameter length is 8 characters.

#### found\_group

An output parameter that indicates the group in which SCLM finds the first occurrence of the member's accounting record within the hierarchy. The maximum parameter length is 8 characters.

**\$acct\_info**

An output parameter pointing to a record containing the static portion of the member's accounting record. See "Pointer Parameters" on page 13 for more details on \$acct\_info.

**\$list\_info**

An output parameter pointing to an array of records containing the dynamic portion of the member's accounting record. See "Pointer Parameters" on page 13 for more details on \$list\_info.

**\$msg\_array**

An output parameter pointing to the message array. See "Pointer Parameters" on page 13 for more details on \$msg\_array.

**Return Codes**

Additional special services messages are written to the FLMMMSG ddname. See "SCLM Service Messages" on page 22 for more information.

Other return codes might be produced by the FLMLNK processor. See "SCLM Service Return Codes" on page 20 for more information about these.

Possible return codes are:

- 0 Normal completion.
- 4 Warning condition. SCLM could not find the accounting record.
- 8 Error condition. The \$msg\_array parameter contains the error message associated with this condition.

**Example**

This example calls the DBACCT service.

**Call Invocation**

**Note:** This example shows general syntax. Call invocations are language-specific. See "Chapter 3. Sample Programs Using SCLM Services" on page 105 for specific examples.

```
lstrc := FLMLNK('DBACCT ',      (* service      *)
               sclm_id,        (* SCLM ID      *)
               'USER1 ',      (* group        *)
               'SOURCE ',     (* type         *)
               'FLM01MD1',    (* member       *)
               found_group,    (* group found  *)
               $acct_info,     (* accounting information pointer *)
               $list_info,     (* list information pointer *)
               $msg_array);    (* message array pointer *)
```

This service call returns the first occurrence of the accounting record for the FLM01MD1 member of the SOURCE type beginning in the USER1 group. The sclm\_id parameter contains a valid SCLM ID returned from the INIT service. SCLM returns all messages produced via the \$msg\_array.

## DBUTIL—Generate a Tailored Output Data Set and Report

The DBUTIL service retrieves information from the project database and creates tailored output and a report. SCLM generates the tailored output in the format you specify. It also describes the contents of the project database based on the selection criteria you supply. You can use the tailored output as input to future FLMCMD command invocations (using the FILE format of FLMCMD) or as input to other project-defined processors.

If you use the FILE format of FLMCMD to call the DBUTIL service, you can save the input parameters in a data set, then use the data set for future invocations of the DBUTIL service. See “Using the FLMCMD File Format” on page 6 for details on using the FILE format of FLMCMD.

The report indicates the contents of the project database based on the selection criteria you supply to the DBUTIL service.

### Command Invocation Format

```
FLMCMD DBUTIL,project,[prj_def]
      ,[acct_group1|_*],[acct_group2]
      ,[acct_group3],[acct_group4]
      ,[acct_group5],[acct_group6]
      ,[acct_type|_*],[acct_member|_*]
      ,[authcode|_*],[change_code|_*]
      ,[change_group|_*],[change_userid|_*]
      ,[language|_*],[YES|NO]
      ,[ACCT|BMAP|_*]
      ,[IN|OUT|_*]
      ,[arch_group],[arch_type],[arch_member]
      ,[EXTENDED|NORMAL|SUBUNIT]
      ,[YES|NO]
      ,[YES|NO]
      ,[report_name],[dd_msgs]
      ,[dd_rept],[dd_tailor]
      ,[report_line]
```

### Call Invocation Format

You cannot use call procedures to start this service.

### Parameters

#### project

The project name. The maximum parameter length is 8 characters.

**prj\_def**

The project definition name to be used for the data extraction. It defaults to the project. The maximum parameter length is 8 characters.

**acct\_group1 - acct\_group6 | \***

The group containing the members, accounting records, and/or build maps to be reported on. The maximum parameter length is 18 characters. You can specify up to six individual acct\_groups, an asterisk for all, or up to six valid patterns. Only groups from the project definition are reported. The default is all account groups (\*).

**acct\_type | \***

The type containing the members, accounting records, and/or build maps to be reported on. Only types from the project definition are reported. The maximum parameter length is 18 characters. You can specify an individual acct\_type, an asterisk for all of them, or a valid pattern. The default is all account types.

**acct\_member | \***

The name of the members' accounting records and/or build maps on which the report will occur. The maximum parameter length is 18 characters. You can specify an individual acct\_member, an asterisk for all of them, or a valid pattern. The default is all account members.

**authcode | \***

The current authorization code for the member. The maximum parameter length is 18 characters. You can specify an individual authcode, an asterisk for all of them, or a valid pattern. The default is all authorization codes.

**change\_code | \***

A value previously assigned by a user for reference purposes. The maximum parameter length is 18 characters. You can specify an individual change\_code, an asterisk for all of them, or a valid pattern. The default is all change codes.

**change\_group | \***

The name of the group in which the member was last updated. The maximum parameter length is 18 characters. You can specify an individual change\_group, an asterisk for all of them, or a valid pattern. The default is all change groups.

**change\_userid | \***

The user ID of the person who made the last update to the member. The maximum parameter length is 18 characters. You can specify an individual change\_userid, an asterisk for all of them, or a valid pattern. The default is all change\_user IDs.

**language | \***

The language of the member. The maximum parameter length is 18 characters. You can specify an individual language, an asterisk for all of them, or a valid pattern. The default is all languages.

**YES | NO**

If you specify YES and use more than one group pattern, a precedence system determines which members are selected. If you specify NO, SCLM selects all versions of all members. The maximum parameter length is 24 characters. The default value is YES.

**ACCT | BMAP | \***

Specify the following type of data to report on:

**ACCT** Accounting information

**BMAP**

Build map information

## DBUTIL Service

- \* Build map and accounting information.

The maximum parameter length is 24 characters. The default value is ACCT.

### IN|OUT|\*

Specify the following to select members:

**IN** Controlled by the architecture definition

**OUT** Not controlled by the architecture definition

- \* Without using an architecture definition to identify them.

The maximum parameter length is 24 characters. The default is an asterisk, which indicates that members will be selected without an architecture definition. If you specify either IN or OUT, you must specify arch\_group, arch\_type, and arch\_member.

### arch\_group

The group used to identify the lowest group in the hierarchy where the architecture begins. The maximum parameter length is 8 characters.

### arch\_type

The type containing the architecture definition that controls the selected members. The maximum parameter length is 8 characters.

### arch\_member

The member containing the architecture definition that controls the selected members. The maximum parameter length is 8 characters.

### EXTENDED|NORMAL|SUBUNIT

Specify the following architecture scope to select:

#### NORMAL

Members that do or do not have compilation unit dependencies.

#### EXTENDED|SUBUNIT

Members that do have compilation unit dependencies.

The maximum parameter length is 24 characters. The default value is NORMAL.

### YES|NO

Specify YES to include page header information in the tailored output. In addition to suppressing the page header information, NO positions the data in column 1 of the tailored output. No carriage returns appear in the output. The maximum parameter length is 24 characters. The default value is YES.

### YES|NO

Specify YES to sum numeric data fields and to show the sum totals in the tailored output. The maximum parameter length is 24 characters. The default value is YES.

### report\_name

The title of the report to be written in the tailored output. The maximum parameter length is 35 characters. Commas are not allowed in the report name.

### dd\_msgs

The ddname indicating the destination of the DBUTIL service messages. If you specify a blank ddname, SCLM routes the DBUTIL service messages to the default output device, such as your terminal. Otherwise, before you call the DBUTIL service, you must allocate the ddname. The following attributes should be used: RECFM=F, LRECL=80, BLKSIZE=80. The maximum parameter length is 8 characters.

**dd\_rept**

The ddname indicating the destination of the report. If you specify a blank ddname, SCLM routes the report to the default output device, such as your terminal. Otherwise, before you call the DBUTIL service, you must allocate the ddname. The following attributes should be used: RECFM=FBA, LRECL=80, BLKSIZE=3120. The maximum parameter length is 8 characters.

**dd\_tailor**

The ddname indicating the destination of the tailored data set. If you specify a blank ddname, SCLM does not generate the tailored output. Otherwise, before you call the DBUTIL service, you must allocate the ddname. The following attributes should be used: RECFM=F, V, FB, or VB; LRECL=80 (minimum); and LRECL=2048 (maximum). If the LRECL value is less than 80, you receive an error message. The report continues to be generated, but it is wrapped using the LRECL value you specify. The maximum parameter length is 8 characters.

**report\_line**

A line of data input that determines the content of the tailored output. Note that you can include commas in the report\_line. If you specify all other parameters or if they default correctly, SCLM does not parse the report\_line for commas. The maximum parameter length is 160 characters, but the report line will be wrapped if it is more than 80 characters long.

If you use SCLM variables with data lengths greater than 8, keep in mind that their values can exceed 8 characters. Place these variables at the end of the report line to ensure that the columns in the report line up evenly. See “Chapter 6. SCLM Variables and Metavariables” on page 269 for more information.

The default value for the report\_line is the following:

```
@@FLMMBR @@FLMLAN @@FLMCML @@FLMNCL @@FLMBLL @@FLMTLS @@FLMCMS @@FLMNCS
```

## Return Codes

Additional special services messages are written to the FLMMMSG ddname. See “SCLM Service Messages” on page 22 for more information.

Other return codes might be produced by the FLMCMD processor. See “SCLM Service Return Codes” on page 20 for more information.

Possible return codes are:

- 0 Normal completion. See the SCLM messages for more information.
- 4 Warning condition. See the SCLM messages for more information.
- 8 Error condition. See the SCLM messages for more information.
- > 8 Severe error condition and SCLM does not produce messages. See “Return Codes” on page 88 for a description of the return code.

## Example

This example calls the DBUTIL service.

### Command Invocation

```
FLMCMD DBUTIL,PROJ1,,USER1,,,,,
*,*,*,*,*,N,ACCT,*,*,*,*,N,N,NAME,,,
UTILTAIL,DELETE,@@FLMPRJ,PROJ1,@@FLMGRP,@@FLMTYP,@@FLMMBR
```

## DBUTIL Service

This service command retrieves accounting information in the USER1 architecture group. SCLM selects all versions of the member without using an architecture definition to identify them. SCLM also selects all accounting types and accounting members that match the pattern.

The `dd_tailor` parameter, `UTILTAIL`, indicates the destination of the tailored output called `NAME`. The `report_line` parameter passes SCLM variables to produce a cleanup report, which you can use to delete all of the members in a group. The cleanup report does not have header information and does not total numeric data fields.

---

## DELETE—Delete Database Components

The DELETE service deletes database components. You can delete an entire member plus its associated accounting record and build map, a member's accounting record and build map, or a member's build map.

If you delete a member from a development group, and the next higher group is non-key, you should also delete the same member from the non-key group if it exists there.

**Note:** The DELETE function requires update authority for the member in order to delete the build map and accounting information.

### Command Invocation Format

```
FLMCMDBDELETE,project
      ,[prj_def]
      ,group
      ,type
      ,member
      ,access_key
      ,[ACCT|BMAP|TEXT]
```

### Call Invocation Format

```
l astrc := FLMLNK('DELETE ',sclm_id
      ,group
      ,type
      ,member
      ,access_key
      ,{ACCT|BMAP|TEXT}
      ,$msg_array);
```

## Parameters

### **project**

The project name. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

### **prj\_def**

The project definition name to be used for the delete. It defaults to the project. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

### **sclm\_id**

An SCLM ID associated with a given project and project definition. The INIT service generates the SCLM ID. The maximum parameter length is 8 characters. This parameter is used for FLMLNK only.

### **group**

The group in which the delete is to occur. The maximum parameter length is 8 characters.

### **type**

The type containing the member, accounting record, and/or build map to be deleted. The maximum parameter length is 8 characters.

### **member**

The name of the member, accounting record, and/or build map to be deleted. The maximum parameter length is 8 characters.

### **access\_key**

The access key assigned to the member with the LOCK service. If you supply the incorrect access key, the delete fails. The maximum parameter length is 16 characters.

### **ACCT|BMAP|TEXT**

Indicates which types of data SCLM is to delete for the member. If you specify BMAP, SCLM deletes only the member's build map. If you specify ACCT, SCLM deletes the member's build map and accounting record. If you specify TEXT, SCLM deletes the member's build map, the member's accounting record, and the member. The maximum parameter length is 24 characters. For FLMCMD, the default value is TEXT. There is no default value for this parameter for FLMLNK.

### **\$msg\_array**

An output parameter pointing to the message array. See "\$msg\_array" on page 13 for more details. This parameter is used for FLMLNK only.

## Return Codes

Additional special services messages are written to the FLMMMSGs ddname. See "SCLM Service Messages" on page 22 for more information.

Other return codes might be produced by the FLMCMD or the FLMLNK processor. See "SCLM Service Return Codes" on page 20 for more information.

Possible return codes are:

- 0** Normal completion.
- 4** Warning condition. The member, accounting record, and/or build map were not found. This return code is set whenever any of the data is missing, regardless of whether the request was for ACCT, BMAP, or TEXT.

## DELETE Service

- 8 Error condition. The \$msg\_array parameter contains the error message associated with this condition.

### Examples

These examples call the DELETE service.

#### Command Invocation

```
FLMCMO DELETE,PROJ1,,USER1,SOURCE,FLM01MD2,XXX#04,ACCT
```

This service command deletes the build map and accounting record for the FLM01MD2 member of the SOURCE type in the USER1 group. The project name is PROJ1. The access key for the member is XXX#04.

If the text for the member FLM01MD2 is missing, then the service returns a return code of 4, even though deletion of the text member was requested.

#### Call Invocation

**Note:** This example shows general syntax. Call invocations are language-specific. See “Chapter 3. Sample Programs Using SCLM Services” on page 105 for specific examples.

```
lastrc := FLMLNK('DELETE ', (* service *)
                 sclm_id, (* SCLM ID *)
                 'USER1 ', (* group *)
                 'SOURCE ', (* type *)
                 'FLM01MD2', (* member *)
                 'XXX#04 ', (* access key *)
                 'ACCT ', (* type of data to delete *)
                 $msg_array); (* message array pointer *)
```

This service call deletes the accounting record and the build map for the FLM01MD2 member of the SOURCE type in the USER1 group. The sclm\_id parameter contains a valid SCLM ID returned from the INIT service and the access key is XXX#04. SCLM returns all messages in the \$msg\_array.

---

## DELGROUP—Delete Group Database Components

The DELGROUP service deletes SCLM-controlled database components associated with a specified group or groups matching a specified pattern. You can delete a member or members and all associated SCLM accounting information and build map records whose names match the selection criteria. 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 can also specify that nothing actually be deleted, but that a deletion report be generated.

## Command Invocation Format

```
FLMCMD DELGROUP,project
    ,[prj_def]
    ,{group|*}
    ,{type|*}
    ,{member|*}
    ,{ACCT|BMAP|TEXT|OUTPUT}
    ,[EXECUTE|REPORT]
    ,[dd_list]
    ,[dd_msgs]
    ,[dd_rept]
    ,[dd_exit]
```

## Call Invocation Format

```
lstrc := FLMLNK('DELGROUP',sclm_id
    ,{group|*}
    ,{type|*}
    ,{member|*}
    ,{ACCT|BMAP|TEXT|OUTPUT}
    ,{EXECUTE|REPORT}
    [,dd_list
    [,dd_msgs
    [,dd_rept
    [,dd_exit]]]);
```

## Parameters

### project

The project name. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

### prj\_def

The project definition name used for the delete. It defaults to the project parameter. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

### sclm\_id

An SCLM ID associated with a given project and project definition. The SCLM ID is generated by the INIT service. The maximum parameter length is 8 characters. This parameter is used for FLMLNK only.

## DELGROUP Service

### **group | \***

The group to be deleted. Only groups that are defined in the project definition will have members deleted. Records in the VSAM data sets for groups that match the pattern but are not in the project definition are not deleted. You can specify an individual group, an asterisk (\*) for all groups, or a valid pattern. If you specify an asterisk, all groups are deleted, so use extreme caution when using the asterisk. The maximum parameter length is 17 characters.

**Note:** If you use the Delete Group Utility panel to invoke Delete Group, you cannot specify a pattern for the group field. Pattern matches in this field are restricted because of the possible hazards of using a pattern in this field.

### **type | \***

The type containing the members, accounting records, and/or build maps to be deleted. The maximum parameter length is 17 characters. You can specify an individual type, an asterisk (\*) for all types, or a valid pattern. You must specify a type. Only members with types defined in the project definition will be deleted.

The member pattern must also match.

### **member | \***

The name of the members, accounting records, and/or build maps to be deleted. The maximum parameter length is 17 characters. You can specify an individual member, an asterisk (\*) for all members, or a valid pattern. See "Selection Parameters" on page 12 for more information on specifying wildcard characters.

The type pattern must also match.

### **ACCT | BMAP | TEXT | OUTPUT**

Indicates which types of data SCLM is to delete.

If you specify BMAP, SCLM deletes only the group's build maps.

If you specify ACCT, SCLM deletes the group's build maps and accounting records.

If you specify TEXT, SCLM deletes the group's build maps, accounting records, and the PDS members associated with those records. If there is no build map or accounting information for a PDS member, the member is not deleted even if you specify the TEXT option.

If you specify OUTPUT, SCLM deletes the group's build outputs that match the selection criteria.

The maximum parameter length is 24 characters.

Because this service deletes information and there is no "Undelete" service, there is no default for this parameter.

**Note:** SCLM can continue to search for deleted data sets that were once active in the project. SCLM issues warning messages if references to deleted data sets are found.

### **EXECUTE | REPORT**

If you specify EXECUTE, any members that match the selection criteria for the specified delete flag are deleted. A report indicating which members were deleted is produced.

If you specify REPORT, no members are deleted. Instead, SCLM produces a report indicating which members are eligible for deletion. SCLM sends this report to the default output device. Specifying REPORT is a good way to identify the outcome of the delete process before deleting any members. The maximum parameter length is 24 characters. For FLMCMD, the default value is REPORT. There is no default value for FLMLNK. You are required to have update authority to the hierarchy data sets to use the DELGROUP service in either REPORT or EXECUTE mode.

**dd\_list**

The ddname indicating the destination of the purge listing for deletion of intermediate code. You must also specify TEXT or OUTPUT and intermediate code must be deleted to produce this report. If you specify a blank ddname, no listing is produced. Otherwise, before you call the DELGROUP service, you must allocate the ddname. The following attributes should be used: RECFM=VBA, LRECL=137, BLKSIZE=3120. The maximum parameter length is 8 characters.

**dd\_msgs**

The ddname indicating the destination of the DELGROUP messages. If you specify a blank ddname, SCLM routes the messages to the default output device. Otherwise, before you call the DELGROUP service, you must allocate the ddname. The following attributes should be used: RECFM=F, LRECL=80, BLKSIZE=80. The maximum parameter length is 8 characters.

**dd\_rept**

The ddname indicating the destination of the DELGROUP report. If you specify a blank ddname, SCLM sends the DELGROUP report to the default output device, such as your terminal. Otherwise, before you call the DELGROUP service, you must allocate the ddname; the following attributes should be used: RECFM=F, LRECL=80, BLKSIZE=80. The maximum parameter length is 8 characters.

**dd\_exit**

The ddname indicating the destination of the delete user exit data. Specify this parameter only if your project definition defines a notify delete user exit routine. Ask your project manager if your project is using a notify delete user exit routine. If you specify a blank ddname, SCLM routes the delete user exit data to NULLFILE. Otherwise, before you call the DELGROUP service, you must allocate the ddname. The following attributes should be used: RECFM=FB, LRECL=160, BLKSIZE=3200. The maximum parameter length is 8 characters.

**Return Codes**

Additional special services messages are written to the FLMMMSGs ddname. See "SCLM Service Messages" on page 22 for more information.

Other return codes might be produced by the FLMCMD or the FLMLNK processor. See "SCLM Service Return Codes" on page 20 for more information.

Possible return codes are:

- 0 Normal completion. See the SCLM messages for more information.
- 4 Warning condition. See the SCLM messages for more information.
- 8 Error condition. See the SCLM messages for more information.
- 12 Severe error condition. SCLM does not produce messages because there was an error invoking the DELGROUP module.

## DELGROUP Service

- 16 Severe error condition. SCLM does not produce messages because it was unable to retrieve SCLM ID information.

## Examples

These examples call the DELGROUP service.

### Command Invocation

```
FLMCMD DELGROUP,PROJ1,,USER1,**,ACCT,EXECUTE
```

This service command deletes the build map and accounting records for all types and members that are associated with the USER1 group in the PROJ1 project. SCLM sends messages to the terminal.

### Call Invocation

**Note:** This example shows general syntax. Call invocations are language-specific. See “Chapter 3. Sample Programs Using SCLM Services” on page 105 for specific examples.

```
lastrc := FLMLNK('DELGROUP',          (* service      *)
                sclm_id,                (* SCLM ID     *)
                'USER1 ',               (* group       *)
                '* ',                  (* type        *)
                '* ',                  (* member      *)
                'ACCT ',                (* types of data *)
                'EXECUTE ',            (* delete members *)
                ,dd_list                (* listing     *)
                ,dd_msgs                (* messages    *)
                ,dd_rept);              (* report      *)
```

This service call deletes the build maps and accounting records for all types and members associated with the USER1 group in the PROJ1 project. The sclm\_id parameter contains a valid SCLM ID returned from the INIT service. SCLM sends messages to the terminal.

---

## DSALLOC—Allocate Data Sets for Group/Type

The DSALLOC service allocates a ddname that corresponds to a hierarchy view specified by the user. The hierarchy view is a concatenation of the PDS data sets, beginning with the PDS data set for the first\_group and adding the PDS for each group above it in the hierarchy. If the ddname already exists, the old ddname is replaced with the new ddname. If unallocated data sets are contained in the hierarchy view, then only the allocated data sets are associated with the ddname. The list of data sets allocated to the ddname does not include extended types.

### Command Invocation Format

```
FLMCMD DSALLOC,project
                ,[prj_def]
                ,first_group
                ,[A|P]
                ,total_groups
                ,type
                ,ddname
```

## Call Invocation Format

```
lastrc := FLMLNK('DSALLOC ',sclm_id
                ,first_group
                ,{A|P}
                ,total_groups
                ,type
                ,ddname
                , $msg_array);
```

## Parameters

### project

The project name. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

### prj\_def

The project definition name used for the delete. It defaults to the project parameter. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

### sclm\_id

The SCLM ID associated with a given project. The INIT service generates the SCLM ID. Maximum parameter length is 8 characters. This parameter is used for FLMLNK only.

### first\_group

The first group in the hierarchy to be allocated to the ddname. Maximum parameter length is 8 characters. This group defines the desired view of the hierarchy. DSALLOC allocates the data sets SCLM uses to search the hierarchy from the group specified.

### A|P

A one-character value indicating the type of hierarchy to be allocated to the ddname. Acceptable values are:

- A** All groups
- P** Primary groups only.

For FLMCMD, the default value is P. There is no default value for FLMLNK.

### total\_groups

The numeric value corresponding to the number of groups for which the allocation is performed. This number includes the first\_group. Specify a zero (0) if the entire hierarchy view is wanted. The default value is zero. If this value is greater than the number of groups in the view, all groups in the view are allocated and a warning occurs. The maximum parameter length is 3 characters.

### type

The name of the type for which the allocation is performed. Maximum parameter length is 8 characters.

### ddname

The ddname for the allocated physical data sets corresponding to the desired hierarchy view. The physical data set names are dynamically allocated to the ddname. You can specify the ddname to be used or leave it blank for the

## DSALLOC Service

FLMLNK interface. If the ddname already exists, the old ddname is replaced with the new ddname. A blank value is not allowed for FLMCMD. If the ddname is blank, SCLM creates a ddname and uses it to allocate data sets; this name is returned to the user. Maximum length of this field is 8 characters.

### **\$msg\_array**

A value that points to the message array. See “Pointer Parameters” on page 13 for more details on \$msg\_array. This parameter is used for FLMLNK only.

## Return Codes

Additional special services messages are written to the FLMMMSGs ddname. See “SCLM Service Messages” on page 22 for more information.

Other return codes might be produced by the FLMCMD or the FLMLNK processor. See “SCLM Service Return Codes” on page 20 for more information.

Possible return codes are:

- 0 Normal completion.
- 4 Warning condition. The \$msg\_array parameter contains the warning message associated with this condition. A warning occurs if the number of data sets allocated to the ddname is less than the number requested in the total\_groups parameter.
- 8 Error condition. The \$msg\_array parameter contains the error message associated with this condition.

## Examples

These examples call the DSALLOC service.

### Command Invocation

```
FLMCMD DSALLOC,PROJ1,,USER1,P,4,SOURCE,APPL
```

This service invocation returns the ddname APPL with the physical data set names corresponding to the hierarchy view specified by the first\_group and the total number of groups. If the hierarchy consisted of 4 groups (USER1, INT, TEST, and RELEASE), these 4 physical data set names would be allocated to ddname APPL. A user wanting a ddname corresponding to a single group would specify the same group for the first\_group and 1 for the total number of groups.

### Call Invocation

**Note:** This example shows general syntax. Call invocations are language-specific. See Chapter 3. Sample Programs Using SCLM Services, for specific examples.

```
lastrc := FLMLNK ('DSALLOC ', (* service *)
                  sclm_id, (* SCLM ID *)
                  'USER1 ', (* first group *)
                  'P', (* hierarchy *)
                  4, (* total groups *)
                  'SOURCE ', (* type *)
                  ddname, (* ddname to allocate *)
                  $msg_array); (* message array pointer *)
```

Assume that the ddname for the preceding example is APPL. This service invocation returns the ddname APPL with the physical data set names

corresponding to the hierarchy view specified by the `first_group` and the total number of groups. If the hierarchy consisted of 4 groups (USER1, INT, TEST, and RELEASE), these 4 physical data set names would be allocated to ddname APPL. Note the project is determined by the `sclm_id` that is obtained by the INIT service call. A user wanting a ddname corresponding to a single group would specify the same group for the `first_group` and 1 for the total number of groups.

---

## EDIT— Edit a Member of a Controlled Library

The EDIT service brings up an SCLM edit session for the requested member. All of the functions of the SCLM edit panel are available from the edit service, including locking, parsing, and storing SCLM accounting data. The SCLM edit commands, such as SPROF, SCREATE, SREPLACE, and SMOVE, are the same as from the SCLM edit dialog.

### Command Invocation Format

```
FLMCMD EDIT,project
           ,[prj_def]
           ,group1
           ,[group2]
           ,[group3]
           ,[group4]
           ,type
           ,member
           ,[Y|N]
           ,[imac]
           ,[prof]
           ,[Y|N]
           ,[Y|N]
           ,[Y|N]
           ,[Y|N]
           ,[authcode]
           ,[chgcode]
           ,[volser]
           ,[dd_editmsgs];
```

## Call Invocation Format

```

1astrc := FLMLNK('EDIT', sclm_id
                ,group1
                ,group2
                ,group3
                ,group4
                ,type
                ,member
                ,Y|N
                ,imac
                ,prof
                ,Y|N
                ,Y|N
                ,Y|N
                ,Y|N
                [,authcode
                [,chgcode)
                [,volser
                [,dd_editmsgs]]]);

```

### Parameters

**project**

The project name. The maximum parameter length is 8 characters.

**prj\_def**

The project definition name to be used for edit. It defaults to project. The maximum parameter length is 8 characters.

**sclm\_id**

An SCLM ID associated with a given project and project definition. The INIT service generates the SCLM ID. The maximum parameter length is 8 characters.

**group1**

The development group at which the member is to be edited. The maximum parameter length is 8 characters.

**group2**

Name of the second group in the concatenation. The maximum parameter length is 8 characters.

**group3**

Name of the third group in the concatenation. The maximum parameter length is 8 characters.

**group4**

Name of the fourth group in the concatenation. The maximum parameter length is 8 characters.

**type**

The type containing the member to be edited. The maximum parameter length is 8 characters.

**member**

The member to be edited. The maximum length for this parameter is 8 characters.

**Y|N**

Y indicates that SCLM will allocate the entire hierarchy, beginning with group1. If Y is selected, group2, group3 and group4 must be blank. N indicates that SCLM will allocate only group1, group2, group3, and group4 (default).

**imac**

The name of an initial macro to be run. The maximum parameter length is 8 characters.

**prof**

The edit profile name to use for the edit session. The maximum parameter length is 8 characters.

**Y|N**

Y indicates that you will have an opportunity to confirm cancel, move, and replace (default). N indicates that cancel, move, and replace commands will execute without confirmation. The maximum parameter length is 24 characters.

**Y|N**

Y indicates that mixed edit mode is to be used. N indicates that mixed edit mode is not to be used (default). The maximum parameter length is 24 characters.

**Y|N**

Y indicates that the data will be edited on the workstation. N indicates that the data will be edited on the host (default). The maximum parameter length is 24 characters.

**Y|N**

Y indicates that the length of variable blocked data will be preserved. N indicates that blanks at the end of the data are retained (default). The maximum parameter length is 24 characters.

**authcode (optional)**

The authorization code to be used for the edit session. SCLM uses the authorization code for the verification just like the SCLM edit dialog. If you do not enter a blank or do not supply an authcode SCLM uses one of the following default values:

- The authorization code from the existing member, if the member being edited exists in the hierarchy.
- The default authorization code for the group if the member does not exist in the hierarchy.

The maximum parameter length is 8 characters.

**chgcode (optional)**

The default change code for the edit session. If the member's accounting record lists the change code, SCLM updates the date and time stamps for the existing change code entry. The maximum parameter length is 8 characters.

## EDIT Service

### **volser (optional)**

Volume serial for parser listings data set. The maximum parameter length is 8 characters.

### **dd\_editmsgs (optional)**

DDNAME of the destination of the edit messages. The maximum parameter length is 8 characters.

## Return Codes

Additional special services messages are written to the FLMMMSGs ddname. See “SCLM Service Messages” on page 22 for more information.

Other return codes might be produced by the FLMCMD or the FLMLNK processor. See “SCLM Service Return Codes” on page 20 for more information.

Possible return codes are:

- 0 Normal completion; data was saved.
- 4 Normal completion; data was **not** saved.
- 8 Error condition. See the dd\_editmsgs for details.
- 12 Severe error condition. SCLM does not produce messages because there was an error invoking the edit module.
- 14 Member in use.

## Example

These examples call the EDIT service.

### **Command Invocation Format**

```
FLMCMD EDIT,sclm70,sclm7044,user,,,,SOURCE,A,Y
```

This service command edits SCLM70.USER.SOURCE(A), drawing down member a from the hierarchy, if it is not in group USER. Alternate SCLM7044 is used to determine the hierarchy.

### **Program Invocation Format**

```
CALL FLMLNK('EDIT ',SLMID,'USER ',BLNK8,BLNK8,BLNK8,  
  'SOURCE ','A ',Y,BLNK8,BLNK8,  
  Y,N,N,N,'PRIVATE ','R8EDS ',BLNK8,  
  DDEDIT)  
RETCODE(R15);
```

This service call edits member A in group USER with type SOURCE, drawing down member EDIT service. The example assumes that the START and INIT services have already completed successfully, so that the SLMID value is valid.

The ddname DDEDIT has been allocated to a data set with valid characteristics.

The authcode is set to 'PRIVATE' and the change code is set to 'R8EDS'.

---

## END— End an SCLM Services Session

The END service stops an SCLM services session. It frees an application ID generated by the START service. Each START service invocation needs a matching END service invocation. This service also calls the FREE service to free any SCLM IDs associated with the given application ID that have not been explicitly freed.

## Command Invocation Format

You cannot use command procedures to call this service.

## Call Invocation Format

```
lastrc := FLMLNK('END      ',appl_id,msg_line);
```

## Parameters

### appl\_id

The application ID associated with the SCLM services session you want to stop. You must generate the application ID using the START service. The maximum parameter length is 8 characters.

### msg\_line

An output parameter that has a buffer containing any END service error message. The maximum parameter length is 80 characters.

## Return Codes

Additional special services messages are written to the FLMMMSGs ddname. See “SCLM Service Messages” on page 22 for more information.

Other return codes might be produced by the FLMLNK processor.

Possible return codes are:

- 0 Normal completion.
- 4 Warning condition. SCLM cannot free an SCLM ID associated with the application ID.
- 8 Error condition. See the msg\_line parameter description for more details.

## Example

This example calls the END service.

### Call Invocation

**Note:** This example shows general syntax. Call invocations are language-specific. See “Chapter 3. Sample Programs Using SCLM Services” on page 105 for specific examples.

```
lastrc := FLMLNK('END      ',      (* service      *)
                appl_id,      (* application ID *)
                msg_line);      (* error messages *)
```

This service call ends the SCLM services session identified by the appl\_id parameter. The appl\_id parameter contains a valid application ID returned from the START service. SCLM returns messages in the msg\_line parameter.

---

## EXPORT—Extract SCLM Accounting Information for a Group

The export service captures all SCLM accounting and build map information associated with a specified group. You can use this service with the IMPORT service to create a consistent set of data that can be archived or used to create a new release, rename a group, or transport software from one hierarchy to another.

## EXPORT Service

Although the SCLM Migration Utility provides a similar function, using the EXPORT and IMPORT services allows you to save build maps. Data presently residing in the group specified is not changed by this service.

### Command Invocation Format

```
FLMCMD EXPORT,project
           ,[prj_def]
           ,group
           ,[Y|N]
           ,[dd_msgs]
           ,[dd_rept]
```

### Call Invocation Format

```
lastrc := FLMLNK('EXPORT ',sclm_id
                 ,group
                 ,{Y|N}
                 [,dd_msgs
                 [,dd_rept]]);
```

### Parameters

#### project

The project name. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

#### prj\_def

The project definition name used for the export. It defaults to the project definition. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

#### sclm\_id

An SCLM ID associated with a given project and project definition. The SCLM ID is generated by the INIT service. The maximum parameter length is 8 characters. This parameter is used for FLMLNK only.

#### group

The group to be exported. The maximum parameter length is 8 characters. The group must be defined in the project definition. The group must have export VSAM data sets defined in the project definition.

#### Y|N

Indicates whether to purge previously exported data from the export data sets for the group. The export data sets must be empty before new export data can be stored in them. If you specify Y, SCLM attempts to purge the data in the data sets. If you specify Y and the purge fails, the export does not occur. If you specify N, SCLM assumes that the export data sets are empty and does not attempt to purge the data sets. If the export data sets are not empty, the export does not occur. The maximum parameter length is 24 characters. For FLMCMD, the default value is N. There is no default value for FLMLNK.

**dd\_msgs (optional)**

The ddname indicating the destination of the export messages. If you specify a blank ddname, SCLM routes the export messages to the default output device, such as your terminal. Otherwise, before you call the EXPORT service, you must allocate the ddname; the following attributes should be used: RECFM=F, LRECL=80, BLKSIZE=80. The maximum parameter length is 8 characters.

**dd\_rept (optional)**

The ddname indicating the destination of the export report. If you specify a blank ddname, SCLM routes the export report to the default output device, such as your terminal. Otherwise, before you call the EXPORT service, you must allocate the ddname; the following attributes should be used: RECFM=F, LRECL=80, BLKSIZE=80. The maximum parameter length is 8 characters.

**Return Codes**

Additional special services messages are written to the FLMMMSGs ddname. See “SCLM Service Messages” on page 22 for more information.

Other return codes might be produced by the FLMCMD or the FLMLNK processor. See “SCLM Service Return Codes” on page 20 for more information.

Possible return codes are:

- 0 Normal completion. See the SCLM messages for more information.
- 4 Warning condition. See the SCLM messages for more information.
- 8 Error condition. See the SCLM messages for more information.
- 12 Severe error condition. SCLM does not produce messages because there was an error invoking the EXPORT module.
- 16 Severe error condition. SCLM does not produce messages because it was unable to retrieve SCLM ID information.

**Examples****Command Invocation**

```
FLMCMD EXPORT,PROJ1,,USER1,Y
```

This service command exports the USER1 group of the PROJ1 project. The export data sets are purged of any existing information before the SCLM accounting information is exported. SCLM sends messages and the report to the terminal.

**Call Invocation**

**Note:** This example shows general syntax. Call invocations are language-specific. See “Chapter 3. Sample Programs Using SCLM Services” on page 105 for specific examples.

```
lastrc := FLMLNK('EXPORT ', (* service *)
                 sclm_id, (* SCLM ID *)
                 'USER1 ', (* group *)
                 'Y ', (* purge exported data *)
                 'EXPMSGs ', (* messages *)
                 'EXPREPT '); (* report *)
```

## EXPORT Service

This service call exports the USER1 group. The `sclm_id` parameter contains a valid SCLM ID returned from the INIT service. The export data sets are purged of any existing information before the SCLM accounting information is exported. SCLM sends messages and the report to the terminal.

---

## FREE—Free an SCLM ID

The FREE service frees an SCLM ID generated by the INIT service. Each INIT service invocation needs a matching FREE service invocation. After freeing the SCLM ID, SCLM closes all project data sets and frees the project definition specified on the INIT service.

### Command Invocation Format

You cannot use command procedures to call this service.

### Call Invocation Format

```
lastrc := FLMLNK('FREE      ',sclm_id
                 ,msg_line);
```

### Parameters

#### `sclm_id`

The SCLM ID to be freed. The INIT service must generate the SCLM ID. The maximum parameter length is 8 characters.

#### `msg_line`

An output parameter that is a buffer containing any FREE service error message. The maximum parameter length is 80 characters.

### Return Codes

Additional special services messages are written to the FLMMMSGs ddname. See “SCLM Service Messages” on page 22 for more information.

Other return codes might be produced by the FLMLNK processor.

Possible return codes are:

0 Normal completion.

8 Error condition. See the `msg_line` parameter description for more details.

### Example

This example calls the FREE service.

#### Call Invocation

**Note:** This example shows general syntax. Call invocations are language-specific. See “Chapter 3. Sample Programs Using SCLM Services” on page 105 for specific examples.

```
lastrc := FLMLNK('FREE      ',      (* service      *)
                 sclm_id,          (* SCLM ID      *)
                 msg_line);        (* error messages *)
```

This service call frees the SCLM ID identified by the `sclm_id` parameter. The `sclm_id` parameter contains a valid SCLM ID returned from the INIT service. SCLM returns messages in the `msg_line` parameter.

---

## IMPORT—Import SCLM Accounting Information to Current Project

The IMPORT service reintroduces the exported SCLM accounting information into the context of the current project, after first verifying that this data corresponds to the current contents of the SCLM controlled data sets.

Like the SCLM editor, the IMPORT service verifies authorization codes and prohibits simultaneous updates of members. The group specified to receive the import must be a development group. The IMPORT service also ensures that all the software components to be imported are available and have correct accounting information. Finally, the IMPORT service verifies that each software component is either new or based directly on the version that exists in the higher group.

**Note:** Upon completion, the IMPORT service purges the EXPORT database of all records that were successfully imported.

### Command Invocation Format

```
FLMCMD IMPORT,project
    ,[prj_def]
    ,group
    ,[authcode|'_']
    ,[change_code|'_']
    ,[userid|'_']
    ,[C|U|R]
    ,[dd_msgs]
    ,[dd_rept]
```

### Call Invocation Format

```
lastrc := FLMLNK('IMPORT ',sclm_id
    ,group
    ,{authcode}
    ,{change_code}
    ,{userid}
    ,{C|U|R}
    [,dd_msgs
    [,dd_rept]]);
```

### Parameters

#### **project**

The project name. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

#### **prj\_def**

The project definition name used for the import. It defaults to the project parameter. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

#### **sclm\_id**

An SCLM ID associated with a given project and project definition. The SCLM ID is generated by the INIT service. The maximum parameter length is 8 characters. This parameter is used for FLMLNK only.

#### **group**

The group into which data is being imported. The maximum parameter length is 8 characters. The group must be defined in the project definition as a development group. Furthermore, the group must have export accounting data sets defined for it in order for import to work.

#### **authcode**

The authorization code to be used for the lock. SCLM uses the authorization code for the verification steps described in “LOCK—Lock a Member or Assign an Access Key” on page 63. If you do not supply an authcode for FLMCMD, or if you specify a blank for either FLMCMD or FLMLNK, SCLM uses the authorization code from the exported accounting information. The maximum parameter length is 8 characters.

#### **change\_code**

If you have a change code verification routine, when you specify this parameter, you must ensure that the change code is valid. When you specify a valid change code, the IMPORT service adds the change code to each editable member’s accounting record and updates the change code date and time to the change date and time from the exported accounting record. If you specify a change code that is already listed in a member’s exported accounting record, the IMPORT service does not add a duplicate change code to the accounting record. It uses the one from the exported accounting record. For FLMCMD, the default value is blank; unless a change code is specified, the IMPORT service will not perform verification. There is no default value for FLMLNK.

#### **userid**

If you supply a value for this parameter, SCLM replaces the USERID field in each exported accounting record with the value supplied. If you do not specify a value, SCLM uses the user ID from the exported accounting information. The maximum parameter length is 8 characters. If no value is specified for FLMCMD or blank is specified for either FLMCMD or FLMLNK, the user ID from the exported accounting information will be used.

#### **C|U|R**

Indicates the import mode, where C=conditional, U=unconditional, and R=report. The maximum parameter length is 24 characters.

When you specify C, the IMPORT service attempts to import the specified group only when each accounting record and build map record successfully passes all the necessary verifications. The IMPORT service fails if any one of these records cannot pass verification. Thus, when you specify conditional

mode, the IMPORT service imports all records or none. The IMPORT service deletes the record from the export database once it has been imported successfully into the specified group.

When you specify U, the IMPORT service performs the same set of verifications, but attempts to import the group even if one or more records do not pass verification. In this case, the IMPORT service imports only those records that passed verification and leaves the records that failed verification in the export database. In addition, IMPORT attempts to store an accounting record with a predecessor baseline date/time verification error. The IMPORT service deletes the record from the export database once it has been imported successfully into the specified group.

When you specify R, the IMPORT service performs the verification and reports the eligibility of members for import. For FLMCMD, the default value is C. There is no default value for FLMLNK.

### **dd\_msgs (optional)**

The ddname indicating the destination of the import messages. If you specify a blank ddname, SCLM routes the import messages to the default output device, such as your terminal. Otherwise, before you call the IMPORT service, you must allocate the ddname; the following attributes should be used: RECFM=F, LRECL=80, BLKSIZE=80. The maximum parameter length is 8 characters.

### **dd\_rept (optional)**

The ddname indicating the destination of the import report. If you specify a blank ddname, SCLM routes the import messages to the default output device, such as your terminal. Otherwise, before you call the IMPORT service, you must allocate the ddname; the following attributes should be used: RECFM=F, LRECL=80, BLKSIZE=80. The maximum parameter length is 8 characters.

## Return Codes

Additional special services messages are written to the FLMMSGSGS ddname. See “SCLM Service Messages” on page 22 for more information.

Other return codes might be produced by the FLMCMD or the FLMLNK processor. See “SCLM Service Return Codes” on page 20 for more information.

Possible return codes are:

- 0 Normal completion. See the SCLM messages for more information.
- 4 Warning condition. See the SCLM messages for more information.
- 8 Error condition. See the SCLM messages for more information.
- 12 Severe error condition. SCLM does not produce messages because there was an error invoking the IMPORT module.
- 16 Severe error condition. SCLM does not produce messages because it was unable to retrieve SCLM ID information.

## Examples

### **Command Invocation**

```
FLMCMD IMPORT,PROJ1,,USER1,,,C
```

This service command imports data into the USER1 group in the PROJ1 project in conditional mode. SCLM sends messages and listings to the terminal.

## IMPORT Service

### Call Invocation

**Note:** This example shows general syntax. Call invocations are language-specific. See “Chapter 3. Sample Programs Using SCLM Services” on page 105 for specific examples.

```
l astrc := FLMLNK('IMPORT ', (* service *)
                    sclm_id, (* SCLM ID *)
                    'USER1 ', (* group *)
                    ' ', (* authorization code *)
                    ' ', (* change code *)
                    ' ', (* user ID *)
                    'C ', (* mode *)
                    'MESSAGES', (* messages *)
                    'REPORT '); (* report *)
```

This service call imports the USER1 group in conditional mode. The sclm\_id parameter contains a valid SCLM ID returned from the INIT service. The ddnames for the messages and report (MESSAGES and REPORT respectively) must be allocated before calling FLMLNK.

---

## INIT—Generate an SCLM ID

The INIT service initializes an SCLM ID. During this process, it also initializes the specified project definition. The INIT service also checks to make sure that the project definition is current. The project definition macros must be reassembled after installing SCLM 3.5. If the macros have not been reassembled, SCLM issues an error message. After the INIT service generates an SCLM ID, it can be passed to other SCLM services, such as DELETE and LOCK. Each INIT service invocation needs a matching FREE service invocation.

**Note:** SCLM maintains allocations of data sets in the hierarchy between uses of SCLM services. This enhances the performance of SCLM; however, if data sets in the hierarchy are created or deleted, the FREE service will need to be invoked to release the existing allocations and a new INIT service invoked to regain access to the project definition.

### Command Invocation Format

You cannot use command procedures to call this service.

### Call Invocation Format

```
l astrc := FLMLNK('INIT ',appl_id
                    ,project
                    ,prj_def
                    ,sclm_id
                    ,msg_line);
```

### Parameters

#### appl\_id

The application ID with which the generated SCLM ID is to be associated. The application ID must be generated by the START service. The maximum parameter length is 8 characters.

**project**

The project name. The maximum parameter length is 8 characters.

**prj\_def**

The project definition name to be initialized for the SCLM ID. The maximum parameter length is 8 characters.

**sclm\_id**

The generated SCLM ID. Each time you invoke the INIT service, it generates a unique SCLM ID. The maximum parameter length is 8 characters.

**msg\_line**

An output parameter that is a buffer containing any INIT service error message. The maximum parameter length is 80 characters.

## Return Codes

Additional special services messages are written to the FLMMMSG ddname. See “SCLM Service Messages” on page 22 for more information.

Other return codes might be produced by the FLMLNK processor. See “SCLM Service Return Codes” on page 20 for more information.

Possible return codes are:

- 0 Normal completion.
- 8 Error condition. See the msg\_line parameter description for more details.

## Example

This example calls the INIT service.

### Call Invocation

**Note:** This example shows general syntax. Call invocations are language-specific. See “Chapter 3. Sample Programs Using SCLM Services” on page 105 for specific examples.

```
lastrc := FLMLNK('INIT      ',      (* service          *)
                 appl_id,      (* application ID    *)
                 'PROJ1     ',      (* project name      *)
                 'PROJ1     ',      (* project definition name *)
                 sclm_id,      (* SCLM ID          *)
                 msg_line);      (* error messages    *)
```

This service call initializes an SCLM ID for the PROJ1 project using the PROJ1 project definition. The appl\_id parameter contains a valid application ID returned from the START service. SCLM returns messages in the msg\_line parameter.

---

## LOCK—Lock a Member or Assign an Access Key

The LOCK service locks a member in a development library, assigns the member an access key, or both. In most cases, LOCK allows one member to be modified by only one user at a time. Locking a member also ensures that updates to the member can occur only in the specified development library until you unlock or promote the member. The member to be locked does not have to exist in a development library or anywhere in the SCLM project hierarchy.

Suppose you are creating a new member on your programmable workstation. You can use LOCK to reserve the member name for future use.

## LOCK Service

You can assign an access key to the member to make the member even more secure than just locking it does. If you assign an access key to a member, you must, thereafter, provide that access key to further modify the member. For an explanation on using access keys, refer to *ISPF Software Configuration and Library Manager (SCLM) Developer's and Project Manager's Guide*. When using access keys, remember:

- Access keys have no effect on the BUILD, DBACCT, DBUTIL, PARSE, and RPTARCH services.
- You must supply the correct member access key when you call the DELETE, SAVE, STORE, and UNLOCK services.
- Before you can promote a member, you must call the UNLOCK service to remove a member's access key. The PROMOTE service promotes any member that has a blank access key.
- If you have successfully completed the SAVE or STORE service for a member, the member remains locked. You can still use the LOCK service to assign an access key to the member.

In most cases, LOCK allows one member to be modified by only one user at a time (see Note). When you edit a member in one development library, LOCK prohibits others from editing the same member in their development libraries. Another user cannot edit the member until you delete the member and its accounting information from your group or you promote the member to a common group.

**Note:** Depending upon the software configuration management plan for a project, a temporary copy of a member could exist in two development libraries at the same time. Refer to *ISPF Software Configuration and Library Manager (SCLM) Developer's and Project Manager's Guide* for more information, or see the project manager for the project.

The LOCK service provides the following capabilities:

- Verifying a group  
LOCK verifies that the group specified is valid. Group verification allows SCLM to control all source modifications to the higher groups of the hierarchy through the promote function.
- Verifying an authorization code  
The project administrator defines a list of *authorization codes* to each group in the project's database. An authorization code is an identifier that SCLM uses to control authority to update and promote members within a hierarchy.  
The LOCK service can only lock those members in the group that are assigned one of the authorization codes defined to the group. See "FLMGROUP Macro" on page 174 for more information.
- Verifying predecessors  
The LOCK service guarantees that the member to be locked in the development library is the most current version of the member within the hierarchical view. *Predecessors* of the member are previous versions of a member existing within the same hierarchical view.  
The LOCK service ensures that the member to be locked does not overlay changes to a predecessor. LOCK does this by verifying that the predecessor of each version of the member within the hierarchical view has not been modified.
- Verifying build output

You cannot lock members that are outputs of a build. This verification prevents accidental modification of a build output member, such as text files and compiler listings. (These members are referred to as “noneditable” elsewhere in this book.)

- Verifying access keys

The LOCK service also prevents you from accidentally modifying or deleting a member you do not control. The access key that you store with the accounting information for a member provides this verification. Locking a member with an access key allows you to prevent others from accidentally modifying or promoting the member if they make changes while working outside of SCLM.

Use the access key as a signal to other developers, not as a security measure. For example, you can use the access key to indicate the location of the member or the reason it was locked.

## Command Invocation Format

```
FLMCMD LOCK,project
        ,[prj_def]
        ,group
        ,type
        ,member
        ,[authcode]
        ,[access_key]
        ,[userid]
```

## Call Invocation Format

```
l astrc := FLMLNK('LOCK      ',sclm_id
                ,group
                ,type
                ,member
                ,{authcode|' '})
                ,{access_key|' '})
                ,{userid|' '})
                ,found_group
                ,max_prom_group
                , $acct_info
                , $list_info
                , $msg_array);
```

### Parameters

**project**

The project name. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

**prj\_def**

The project definition name to be used for the lock. It defaults to project. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

**sclm\_id**

An SCLM ID associated with a given project and project definition. The INIT service generates the SCLM ID. The maximum parameter length is 8 characters. This parameter is used for FLMLNK only.

**group**

The group in which the member is to be locked. The specified group must be a development library. The maximum parameter length is 8 characters.

**type**

The type containing the member to be locked. The maximum parameter length is 8 characters.

**member**

The member to be locked. The maximum parameter length is 8 characters.

**authcode**

The authorization code to be used for the lock. If you do not supply an authcode, SCLM uses one of the following default values:

- The authorization code from the existing member if the member being locked exists in the hierarchy
- The default authorization code for the group if the member does not exist in the hierarchy.

The maximum parameter length is 8 characters.

**access\_key**

The access key to be assigned to the member. It defaults to blank. The maximum parameter length is 16 characters. You must use the access key for any further manipulation of the member until you use the UNLOCK service to remove the access key.

**userid**

User ID of the person requesting the lock. It defaults to the current system user ID. The maximum parameter length is 8 characters.

**found\_group**

An output parameter that indicates the group in which the first occurrence of the member exists within the hierarchy. The maximum parameter length is 8 characters. This parameter is used for FLMLNK only.

**max\_prom\_group**

An output parameter that indicates the highest group in the hierarchy to which the member can be promoted. This member's maximum promotable group is based on the authorization code you use for the lock. The maximum parameter length is 8 characters. This parameter is used for FLMLNK only.

**\$acct\_info**

An output parameter pointing to a record containing the static portion of the member's accounting record. See "\$acct\_info" on page 14 for more details. This parameter is used for FLMLNK only.

**\$list\_info**

An output parameter pointing to an array of records that contains the dynamic portion of the member's accounting record. See "\$list\_info" on page 15 for more details. This parameter is used for FLMLNK only.

**\$msg\_array**

An output parameter pointing to the message array. See "\$msg\_array" on page 13 for more details. This parameter is used for FLMLNK only.

**Return Codes**

Additional special services messages are written to the FLMMMSGs ddname. See "SCLM Service Messages" on page 22 for more information.

Other return codes might be produced by the FLMCMD or the FLMLNK processor. See "SCLM Service Return Codes" on page 20 for more information.

Possible return codes are:

- 0 Normal completion. If a member is already locked, and no information concerning the lock has changed (the change code, or language, for example), then no action will be taken, but the return code will still be 0. No audit or versioning records will be written in this case.
- 8 Error condition. The \$msg\_array parameter contains the error message associated with this condition.

**Examples**

These examples call the LOCK service.

**Command Invocation**

```
FLMCMD LOCK,PROJ1,,USER1,SOURCE,FLM01MD2,,XXX#04
```

This service command locks the FLM01MD2 member of the SOURCE type in the USER1 group. The project name is PROJ1. The access key to be assigned to the member is XXX#04. The authcode and user ID parameters are defaults.

**Call Invocation**

**Note:** This example shows general syntax. Call invocations are language-specific. See "Chapter 3. Sample Programs Using SCLM Services" on page 105 for specific examples.

```
lastrc := FLMLNK('LOCK ', (* service *)
                 sclm_id, (* SCLM ID *)
                 'USER1 ', (* group *)
                 'SOURCE ', (* type *)
                 'FLM01MD2', (* member *)
                 'TESTAC ', (* authorization code *)
                 'XXX#04 ', (* access key *)
                 ' ', (* user ID *)
                 found_group, (* found group *)
                 max_prom_group, (* maximum promotable group *)
                 $acct_info, (* accounting information pointer *)
                 $list_info, (* list information pointer *)
                 $msg_array); (* message array pointer *)
```

This service call locks the FLM01MD2 member of the SOURCE type in the USER1 group. The sclm\_id parameter contains a valid SCLM ID returned from the INIT

## LOCK Service

service. The authorization code to be used for the lock verification is TESTAC and the access key is XXX#04. USERID is the user requesting the lock. SCLM returns all messages in the \$msg\_array parameter.

---

## MIGRATE—Create Accounting for Selected Members

The MIGRATE service creates or updates SCLM accounting information for members in a development library that match a given pattern.

MIGRATE checks each member whose name matches the pattern for valid SCLM accounting information. If a selected member does not have valid accounting information or if *forced* mode is specified, MIGRATE invokes the SAVE service to lock, parse, and store the member. All of the rules and restrictions that apply to the SAVE service also apply to the MIGRATE service.

**Note:** The MIGRATE service does not parse a member correctly if the member is packed. Make sure that the pack mode is off in the member's profile.

For more information on the SAVE, LOCK, PARSE, and STORE services, see their service descriptions in this chapter.

### Command Invocation Format

```
FLMCMD MIGRATE,project,[prj_def]
      ,group,type,member
      ,[authcode]
      ,[language]
      ,[change_code]
      ,[C|U|F]
      ,[dd_migmsgs]
      ,[dd_miglist]
      ,[dd_migrept]
      ,[date]
      ,[time]
```

### Call Invocation Format

```
lastrc:=FLMLNK('MIGRATE ',sclm_id
      ,group
      ,type
      ,member
      ,authcode
      ,language
      ,change_code
      ,C|U|F
      ,[dd_migmsgs]
```

```

, [dd_miglist]
, [dd_migrept]
, [date]
, [time]);

```

## Parameters

### project

The project name. The maximum parameter length is 8 characters.

### prj\_def

The project definition name to be used for the lock, parse, and store of migrated members. It defaults to the project parameter. The maximum parameter length is 8 characters.

### group

The group in which the migration is to occur. The specified group must be a development group. The maximum parameter length is 8 characters.

### type

The type containing the members. The maximum parameter length is 8 characters.

### member

A pattern used to select the members to be migrated. The maximum parameter length is 10 characters. You must specify a valid member name or valid pattern, or an error message appears.

### authcode

The authorization code to be used for locking selected members. If you do not supply an authcode or the authcode is blank, SCLM uses default values as follows:

- The authorization code from the existing member if the member being migrated exists in the hierarchy
- The default authorization code for the group if the member does not exist in the hierarchy.

The maximum parameter length is 8 characters.

### language

The language of the member. The maximum parameter length is 8 characters. You must specify the language the first time you save a member.

### change\_code

A change\_code to be added to the information obtained by parsing the member. If the member's accounting record lists the change\_code, SCLM updates the date and time stamps for the existing change\_code entry. The maximum parameter length is 8 characters.

### CIUIF

Indicates the migrate mode (C=Conditional; U=Unconditional; F=Forced). The maximum parameter length is 24 characters. The default value for FLMCMD is C. There is no default value for FLMLNK.

### dd\_migmsgs

The ddname indicating the destination of the messages generated by the MIGRATE service. If you specify a blank ddname, SCLM routes the MIGRATE service messages to the default output device, such as your terminal.

## MIGRATE Service

Otherwise, before you call the MIGRATE service, you must allocate the ddname. The following attributes should be used: RECFM=F, LRECL=80, BLKSIZE=80. The maximum parameter length is 8 characters.

### dd\_miglist

The ddname indicating the destination of the parser listings. If you specify a blank ddname, SCLM does not generate the parser listings. The maximum parameter length is 8 characters.

If the parser for the specified language does not produce a listing, specify a blank ddname. The language parsers supplied by SCLM do not produce a listing. If the parser for the specified language does produce a listing and you specified a ddname, allocate the ddname with the attributes required by the parser. Project-specific parsers can produce a listing. See "FLMTRNSL Macro" on page 190 for more information on project-defined parsers.

### dd\_migrept

The ddname indicating the destination of the migrate report. If you specify a blank ddname, SCLM routes the migrate report to the default output device, such as your terminal. Otherwise, before you call the MIGRATE service, you must allocate the ddname; the following attributes should be used: RECFM=FBA, LRECL=80, BLKSIZE=3120. The maximum parameter length is 8 characters.

### date

The date to assign to the accounting record and member statistics. Use this field if you want to keep audit records and versions from another library system. The date defaults to the current date. This parameter is required if the "time" parameter is entered. The parameter length is 10 characters. The date, with a 4-character year, must be specified in the national language format.

### time

The time to assign to the accounting record and member statistics. This parameter is required if the "date" parameter is entered. The time must be specified in the national language format. The parameter length is 8 characters.

## Return Codes

Additional special services messages are written to the FLMMMSGs ddname. See "SCLM Service Messages" on page 22 for more information.

Other return codes might be produced by the FLMCMD or the FLMLNK processor. See "SCLM Service Return Codes" on page 20 for more information.

Possible return codes are:

- 0 Normal completion.
- 4 Warning condition. See the SCLM messages for more information.
- 8 Error condition. See the SCLM messages for more information.

## Examples

These examples call the MIGRATE service.

### Command Invocation

```
FLMCMD MIGRATE,PROJ1,,USER1,SOURCE,MOD*,TESTAC,COBOL,CC001234,,PARSEDD
```

This service command migrates (locks, parses, and stores accounting information) members with names beginning MOD (such as MOD1, MOD2, or MODULE) of

the type SOURCE in the USER1 group. The project name is PROJ1 and the authorization code is TESTAC. Change code CC001234 is to be added to the information obtained by parsing the member with the COBOL parser.

SCLM copies parser listings to the PARSEDD ddname only if errors occur. You must allocate the PARSEDD ddname before you call the service.

Messages generated by the MIGRATE service appear on the default output device - this is probably the terminal if you are running under a foreground TSO session.

## Call Invocation

**Note:** This example shows general syntax. Call invocations are language-specific. See “Chapter 3. Sample Programs Using SCLM Services” on page 105 for specific examples.

```
lastrc := FLMLNK('MIGRATE ', (* service *)
                 scln_id, (* application ID *)
                 'RELEASE ', (* group *)
                 'SOURCE ', (* type *)
                 '* ', (* all members *)
                 ' ', (* default authcode *)
                 'HLASM ', (* language *)
                 'INIT ', (* change code *)
                 'C ', (* conditional *)
                 'MSGSDD ', (* messages ddname *)
                 'LISTDD ', (* list ddname *)
                 'REPTDD '; (* report ddname *)
```

This service call migrates all members of the SOURCE type and RELEASE group in the project. Each member’s accounting record has the default authcode, as defined in the project definition. All members have a language of HLASM and a change code of INIT. Any messages are written to the data set allocated to MSGSDD, and the migrate report appears in the data set allocated to REPTDD. Any parser errors are written to the data set allocated to LISTDD.

This service call initializes an SCLM ID for the PROJ1 project using the PROJ1 definition. The appl\_id parameter contains a valid application ID returned from the START service. SCLM returns messages in the msg\_line parameter.

---

## NEXTGRP— Retrieve Next Group in SCLM Hierarchy

The NEXTGRP service returns the name of the next group in a given hierarchy. For a given group, the next group is returned in the SHARED pool variable ZSNXTGRP. An indicator whether the group is key or nonkey is returned in SHARED pool variable ZSNGPKEY. The possible values for ZSNGPKEY are KEY for key groups, and NONKEY for nonkey groups.

## Command Invocation Format

```
FLMCMD NEXTGRP,project
                ,[prj_def]
                ,group
                ,[dd_msgs]
```

## Call Invocation Format

```
l astrc:=FLMLNK('NEXTGRP ',sclm_id
               ,group
               ,dd_msgs);
```

## Parameters

### project

The project name. The maximum parameter length is 8 characters.

### prj\_def

The project definition name to be used for NEXTGRP. It defaults to the project parameter. The maximum parameter length is 8 characters.

### sclm\_id

An SCLM ID associated with a given project and project definition. The INIT service generates the SCLM ID. The maximum parameter length is 8 characters.

### group

The group for which the "next" group is to be found. The maximum parameter length is 8 characters.

### dd\_msgs

The ddname indicating the destination of the messages generated by the NEXTGRP service. The maximum parameter length is 8 characters.

## Return Codes

Other return codes might be produced by the FLMCMD or the FLMLNK processor. See "SCLM Service Return Codes" on page 20 for more information.

Possible return codes are:

- 0 Normal completion. NEXTGRP completed successfully. Variables are set.
- 4 Warning condition. The group is already the top group. No variables are set.
- 8 Error condition. Invalid project, prj\_def, or group name.
- 12 Severe error condition. SCLM might not produce messages because there was an error invoking the NEXTGRP module. For some conditions, messages are available.

## Examples

### Command Invocation

The following REXX exec begins at group USER and finds each successive group in the hierarchy defined by the SCLM7010 alternate of the SCLM70 project.

```
/* REXX exec to find the next groups in a hierarchy */
TRACE off
address ispxexec
group = 'USER'
done = 'false'
address 'TSO' 'alloc fi(ddm) da(sclm.msgs) shr mod'
do until done = 'true'
  'select cmd(FLMCMD NEXTGRP,SCLM70,SCLM7010,'group',ddm)'
  if rc > 0 then
    do
      done = 'true'
    end
end
```

```

else
  do
    'vget (zsnxtgrp,zsngpkey) shared'
    say 'For group' group 'the next group is' zsnxtgrp zsngpkey
    group = zsnxtgrp
  end
end
address 'TS0' 'free fi(ddm)'

```

Executing this example produces this output:

```

For group USER the next group is STGE KEY
For group STGE the next group is DEV KEY
For group DEV the next group is INT KEY
For group INT the next group is REL KEY
For group REL the next group is BASE KEY

```

### Call Invocation

**Note:** This example shows general syntax. Call invocations are language-specific. See “Chapter 3. Sample Programs Using SCLM Services” on page 105 for specific examples.

This program fragment uses the NEXTGRP service to find the group that USER promotes into. The variables ZSNXTGRP and ZSNGPKEY are vdefined to local program variables, and the values set by the NEXTGRP service are retrieved from the shared pool by the VGET service. The example assumes that the START and INIT services have already completed successfully, so that the SLMID value is valid. The ddname DDMSGs has been allocated to a data set with valid characteristics.

```

CALL FLMLNK('NEXTGRP ',SLMID,'USER ',DDMSGs)
  RETCODE(R15);
  EVAL(8),' ',' ');
CALL ISPLINK ('VDEFINE ', 'ZSNXTGRP', ZSNXTGRP, 'CHAR ',
  EVAL(8),' ',' ');
CALL ISPLINK ('VDEFINE ', 'ZSNGPKEY', ZSNGPKEY, 'CHAR ',
  EVAL(8),' ',' ');
CALL ISPLINK ('VGET ', 'ZSNGPKEY', 'SHARED ');
CALL ISPLINK ('VGET ', 'ZSNXTGRP', 'SHARED ');

```

---

## PARSE—Parse a Member for Statistical and Dependency Information

The PARSE service parses a member for statistical and dependency information. SCLM returns two buffers containing the member’s vital information that you can pass on to the STORE service. When the STORE service receives this information, it places it in the member’s accounting record.

### Command Invocation Format

You cannot use command procedures to call this service.

## Call Invocation Format

```

lastrc := FLMLNK('PARSE  ',sclm_id
                ,group
                ,type
                ,member
                ,language
                ,{Y|N}
                ,ddname
                , $stats_info
                , $list_info
                , $msg_array);

```

### Parameters

#### **sclm\_id**

An SCLM ID associated with a given project and project definition. The INIT service generates the SCLM ID. The maximum parameter length is 8 characters.

#### **group**

The group in which the member is to be parsed. The maximum parameter length is 8 characters. Note that a member can be parsed in any group; the specified group does not have to be a development library.

#### **type**

The type containing the member to be parsed. The maximum parameter length is 8 characters.

#### **member**

The member to be parsed. The maximum parameter length is 8 characters.

#### **language**

The language used to identify the parser that will be invoked for the member. The maximum parameter length is 8 characters.

#### **Y|N**

Y indicates that parser listings are to be copied to the ddname parameter only if parser errors occur. N indicates that all parser listings are to be copied to the ddname. The maximum parameter length is 24 characters.

If the parser for the specified language does not produce a listing, specify Y. (The language parsers supplied by SCLM do not produce a listing.) If the parser for the specified language does produce a listing, specify either value. For more efficient performance, specify Y. Project-specific parsers can produce a listing.

#### **ddname**

The ddname indicating the destination of the parser listings. If you specify a blank ddname, SCLM does not generate parser listings. The maximum parameter length is 8 characters.

If the parser for the specified language does not produce a listing, you should specify a blank ddname. The parsers supplied by SCLM do not produce a

listing. If the parser for the specified language does produce a listing and you specify a ddname, allocate the ddname with the attributes the parser requires. Project-specific parsers can produce a listing.

**\$stats\_info**

An output parameter pointing to a record containing the member’s statistical information derived from parsing the member. See “\$stats\_info” on page 15 for more details.

**\$list\_info**

An output parameter pointing to an array of records that contains the member’s include, change code, and user entry information derived from parsing the member. See “\$list\_info” on page 15 for more details.

**\$msg\_array**

An output parameter pointing to the message array. See “\$msg\_array” on page 13 for more details.

**Return Codes**

Additional special services messages are written to the FLMMMSG ddname. See “SCLM Service Messages” on page 22 for more information.

Other return codes might be produced by the FLMCMD processor. See “SCLM Service Return Codes” on page 20 for more information.

Possible return codes are:

- 0 Normal completion.
- 4 Warning condition. A parser error occurred.
- 8 Error condition. The \$msg\_array parameter contains the error message associated with this condition.

**Example**

This example calls the PARSE service.

**Call Invocation**

**Note:** This example shows general syntax. Call invocations are language-specific. See “Chapter 3. Sample Programs Using SCLM Services” on page 105 for specific examples.

```

lastrc := FLMLNK('PARSE ', (* service *)
                  sclm_id, (* SCLM ID *)
                  'USER1 ', (* group *)
                  'SOURCE ', (* type *)
                  'FLM01MD2', (* member *)
                  'PASCAL ', (* language *)
                  'Y', (* listings *)
                  'PARSEDD ', (* ddname of listings *)
                  $stats_info, (* statistical information pointer *)
                  $list_info, (* list information pointer *)
                  $msg_array); (* message array pointer *)

```

This service call parses the FLM01MD2 member of the SOURCE type in the USER1 group. The sclm\_id contains a valid SCLM ID returned from the INIT service. SCLM uses the PASCAL parser and copies the parser listings to the PARSEDD ddname only if errors occur. You must allocate the PARSEDD ddname before you

## PARSE Service

call FLMLNK. SCLM returns the parse results in the \$stats\_info and \$list\_info parameters and all messages in the \$msg\_array parameter.

---

## PROMOTE—Promote a Member from One Library to Another

The PROMOTE service moves data, that is, promotes data through the project database according to a project's architecture definition and project definition. Before SCLM can promote a member, it must have a blank access key and must have successfully completed the BUILD service. If a member has an access key, you must call the UNLOCK service to reset the access key before you can promote the member.

### Command Invocation Format

```
FLMCMDBUILD PROMOTE,project
                ,[prj_def]
                ,group
                ,type
                ,member
                ,[userid]
                ,[E|N|S]
                ,[C|U|R]
                ,[dd_prommsgs]
                ,[dd_promrept]
                ,[dd_promexit]
                ,[dd_copyerr]
                ,[error_list]
                ,[create_rept]
                ,[prefix_userid]
                ,[dd_bldmsgs]
                ,[dd_bldlist]
                ,[dd_bldrept]
                ,[dd_bldexitr]
```

### Call Invocation Format

```
lastrc := FLMLNK('PROMOTE ',sclm_id
                ,group,type,member
                ,{userid|' '})
                ,{E|N|S}
```

```
,{C|U|R}
[,dd_prommsgs[,dd_promrept
[,dd_promexit[,dd_copyerr,
[, {Y|N}
[, {Y|N}
[, {prefix_userid|' '}
[,dd_bldmsgs
[,dd_bldrept
[,dd_bldlist
[,dd_bldexit]]]]]]]]]]];
```

## Parameters

### project

The project name. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

### prj\_def

The project definition name to be used for the promote. It defaults to project. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

### sclm\_id

An SCLM ID associated with a given project and project definition. The INIT service generates the SCLM ID. The maximum parameter length is 8 characters. This parameter is used for FLMLNK only.

### group

The group the promote occurs from. The maximum parameter length is 8 characters.

### type

The type containing the member to be promoted. The maximum parameter length is 8 characters.

### member

The name of the architecture member or source member to be promoted. The maximum parameter length is 8 characters.

### userid

The user ID of the person requesting the promote. If no value is specified for FLMCMD or a blank ( ' ') is specified for FLMLNK, it defaults to your TSO prefix or user ID if no TSO prefix has been created. The maximum parameter length is 8 characters.

### E|N|S

Indicates the promote scope (E=extended, N=normal, S=subunit). The maximum parameter length is 24 characters. The default value for FLMCMD is N. There is no default value for FLMLNK.

### C|R|U

Indicates the promote mode (C=conditional, R=Report, U=Unconditional). The maximum parameter length is 24 characters. The default value for FLMCMD is C. There is no default value for FLMLNK.

## PROMOTE Service

### **dd\_prommsgsgs (optional)**

The ddname indicating the destination of the promote messages. If you specify a blank ddname, SCLM routes the promote messages to the default output device, such as your terminal. Otherwise, before you call the PROMOTE service, you must allocate the ddname. The following attributes should be used: DISP=MOD, RECFM=F, LRECL=80, BLKSIZE=80. The maximum parameter length is 8 characters.

### **dd\_promrept (optional)**

The ddname indicating the destination of the promote report. If you specify a blank ddname, SCLM routes the promote report to the default output device, such as your terminal. Otherwise, before you call the PROMOTE service, you must allocate the ddname. The following attributes should be used: RECFM=FBA, LRECL=80, BLKSIZE=3120. The maximum parameter length is 8 characters.

### **dd\_promexit (optional)**

The ddname indicating the destination of the promote user exit data. Specify this parameter only if your project administrator defined a promote user exit routine in your project definition. Ask your project manager if your project is using a promote user exit routine. If you specify a blank ddname, SCLM routes the promote user exit data to NULLFILE. Otherwise, before you call the PROMOTE service, you must allocate the ddname. The following attributes should be used: RECFM=FB, LRECL=160, BLKSIZE=3200. The maximum parameter length is 8 characters.

### **dd\_copyerr (optional)**

The ddname indicating the destination of the promote copy error information. The promote copy error information consists of system messages indicating the cause of copy errors during promote processing.

If you specify a blank ddname, SCLM routes the promote copy error information to the default output device, such as your terminal. Otherwise, before you call the PROMOTE service, you must allocate the ddname. The maximum parameter length is 8 characters.

**Note:** The remaining parameters are applicable only if the project has a language with rebuild on promote specified (an FLMLRBLD statement).

### **Y|N (optional)**

Y indicates that build translator listings are to be copied to the dd\_bldlist ddname only if errors occur. N indicates that all translator listings are to be copied to the dd\_bldlist ddname. For FLMCMD, the default is Y. There is no default for FLMLNK. The maximum parameter length is 24 characters. This parameter only applies if the project definition requests automatic rebuild when a member is promoted into the 'to group'.

### **Y|N (optional)**

Y indicates that a build report is to be produced and routed to the bldrept ddname. N indicates that a build report is not to be produced. For FLMCMD, the default is Y. There is no default for FLMLNK. The maximum parameter length is 24 characters. This parameter only applies if the project definition requests automatic rebuild when a member is promoted into the 'to group'.

### **prefix\_userid (optional)**

This is the data set name prefix to be used when locating and cataloging temporary data sets. If no value is specified for FLMCMD or a blank ( ' ') is specified for FLMLNK, it defaults to the user Id parameter. The maximum

parameter length is 17 characters. This parameter only applies if the project definition requests automatic rebuild when a member is promoted into the 'to group'.

**dd\_bldmsgs (optional)**

This is the ddname indicating the destination of the build messages. If you specify a blank ddname, SCLM routes the build messages to the default output device, such as your terminal. Otherwise, before you call the BUILD service, you must allocate the ddname. The following attributes should be used: RECFM=F, LRECL=80, BLKSIZE=80. You cannot specify a blank ddname for FLMLNK. This parameter only applies if the project definition requests automatic rebuild when a member is promoted into the 'to group'. The maximum parameter length is 8 characters.

**dd\_bldrept (optional)**

This is the ddname indicating the destination of the build report. If you specify a blank ddname, SCLM routes the build report to the default output device, such as your terminal. Otherwise, before you call the BUILD service, you must allocate the ddname. The following attributes should be used: RECFM=FBA, LRECL=80, BLKSIZE=3120. The maximum parameter length is 8 characters. This parameter only applies if the project definition requests automatic rebuild when a member is promoted into the 'to group'.

**dd\_bldlist (optional)**

This is the ddname indicating the destination of the build listings. If you specify a blank ddname, SCLM does not generate the build listings. Otherwise, before you call the BUILD service, you must allocate the ddname. The following attributes should be used: DISP=MOD, RECFM=VBA, LRECL=137, BLKSIZE=3120. The maximum parameter length is 8 characters. This parameter only applies if the project definition requests automatic rebuild when a member is promoted into the 'to group'.

**dd\_bldexit (optional)**

This is the ddname indicating the destination of the build user exit data. Specify this parameter only if your project definition defines a build user exit routine. Ask your project manager if your project is using a build user exit routine. If you specify a blank ddname, SCLM routes the build user exit data to NULLFILE. Otherwise, before you call the BUILD service you must allocate the ddname. The following attributes should be used: RECFM=FB, LRECL=160, BLKSIZE=3200. The maximum parameter length is 8 characters. This parameter only applies if the project definition requests automatic rebuild when a member is promoted into the 'to group'.

**Return Codes**

Additional special services messages are written to the FLMMMSGs ddname. See "SCLM Service Messages" on page 22 for more information.

Other return codes might be produced by the FLMCMD or the FLMLNK processor. See "SCLM Service Return Codes" on page 20 for more information.

Possible return codes are:

- 0 Normal completion. See the SCLM messages for more information.
- 4 Warning condition. See the SCLM messages for more information. The location of the messages file is determined by the dd\_prommsgs parameter.
- 8 Error condition. See the SCLM messages for more information.

## PROMOTE Service

- | 10 Promote completed successfully. Build was requested in the project  
| definition, but the build failed. See the build messages file allocated to the  
| dd\_bldmsgs parameter for more information.
- 12 Severe error condition. SCLM does not produce messages because there  
was an error invoking the promote module.
- 16 Severe error condition. SCLM does not produce messages because SCLM  
cannot retrieve SCLM ID information.

## Examples

These examples call the PROMOTE service.

### Command Invocation

```
FLMCMO PROMOTE,PROJ1,,USER1,ARCHDEF,FLM01CMD,,U
```

This service command promotes the FLM01CMD member of the ARCHDEF type and all of its dependent members from the USER1 group to the next group in the hierarchy. The project name is PROJ1. The promote scope is normal (by default) and the promote mode is unconditional. SCLM sends messages, reports, and listings to the terminal.

### Call Invocation

**Note:** This example shows general syntax. Call invocations are language-specific. See “Chapter 3. Sample Programs Using SCLM Services” on page 105 for specific examples.

```
lastrc := FLMLNK('PROMOTE ', (* service *)
                  sclm_id, (* SCLM ID *)
                  'USER1 ', (* group *)
                  'ARCHDEF ', (* type *)
                  'FLM01CMD', (* member *)
                  ' ', (* user ID *)
                  'E ', (* scope *)
                  'R ', (* mode *)
                  'PROMMSGS', (* messages *)
                  'PROMREPT', (* report *)
                  'PROMEXIT', (* user exit data *)
                  'COPYDD '); (* copy errors *)
```

This service call performs a report-only promote on the FLM01CMD member of the ARCHDEF type in the USER1 group. The sclm\_id parameter contains a valid SCLM ID returned from the INIT service and USERID identifies who is requesting the promote. The promote scope is extended. You must allocate the ddnames (PROMMSGS, PROMREPT, PROMEXIT, and COPYDD, respectively) before you call FLMLNK.

---

## RPTARCH—Generate an SCLM Architecture Report

The RPTARCH service provides a list of all the components in a given application. The report generator examines the requested architecture and all of its references, and then constructs an indented report of the architecture. The report lists software components in each type referenced by the architecture to help you eliminate unnecessary code.

## Command Invocation Format

```
FLMCMD RPTARCH,project,[prj_def]
           ,group
           ,type
           ,member
           ,[HL|LEC|CC|GEN|TOP SOURCE|NONE]
           ,dd_rptmsgs
           ,dd_rptrept
```

## Call Invocation Format

You cannot use call procedures to start this service.

## Parameters

### project

The project name. The maximum parameter length is 8 characters.

### prj\_def

The project definition name to be used for generating the architecture report. It defaults to project. The maximum parameter length is 8 characters.

### group

The group the report is to be generated from. The maximum parameter length is 8 characters. If information is not found at the specified group, RPTARCH searches up the hierarchy to the next layer.

### type

The type containing the member to be reported on. The maximum parameter length is 8 characters.

### member

The member to be reported on. The maximum parameter length is 8 characters.

### HL|LEC|CC|GEN|TOP SOURCE|NONE

Indicates the cutoff (determines depth) for the architecture report.

The architecture report contains the following if you specify:

#### HL

The HL architecture members in the application represented by the architecture member you specified with the member parameter.

#### LEC

The HL and LEC architecture members in the application represented by the architecture member you specified with the member parameter.

#### CC

The HL, LEC, and CC architecture members in the application represented by the architecture member you specified with the member parameter.

#### GEN

The HL and generic architecture members in the application represented by the architecture member you specified with the member parameter.

#### TOP SOURCE

The HL, LEC, CC, and generic architecture members and top source

## RPTARCH Service

members in the application represented by the architecture member you specified with the member parameter.

### NONE

The HL, LEC, CC, and generic architecture members in each of the types and all source members down to the lowest include group in the application represented by the architecture member you specified with the member parameter.

The maximum parameter length is 24 characters. The default value is NONE.

### dd\_rptmsgs

The ddname indicating the destination of the RPTARCH service messages. If you specify a blank ddname, SCLM routes the RPTARCH service messages to the default output device, such as your terminal. Otherwise, before you call the RPTARCH service, you must allocate the ddname; the following attributes should be used: RECFM=F, LRECL=80, BLKSIZE=80. The maximum parameter length is 8 characters.

### dd\_rptrept

The ddname indicating the destination of the architecture report. If you specify a blank ddname, SCLM routes the architecture report to the default output device, such as your terminal. Otherwise, before you call the RPTARCH service, you must allocate the ddname; the following attributes should be used: RECFM=FBA, LRECL=80, BLKSIZE=3120. The maximum parameter length is 8 characters.

## Return Codes

Additional special services messages are written to the FLMMMSGs ddname. See "SCLM Service Messages" on page 22 for more information.

Other return codes might be produced by the FLMCMD processor. See "SCLM Service Return Codes" on page 20 for more information about these.

Possible return codes are:

- 0 Normal completion. See the SCLM messages for more information.
- 4 Warning condition. See the SCLM messages for more information.
- 8 Error condition. See the SCLM messages for more information.

## Example

This example calls the RPTARCH service.

### Command Invocation

```
FLMCMD RPTARCH,PROJ1,,USER1,SOURCE,FLM01MD1,NONE
```

This service command generates an architecture report for the FLM01MD1 member of the SOURCE type in the USER1 group. The project name is PROJ1. The report cutoff is NONE, and SCLM sends messages and the architecture report to your terminal.

## SAVE—Lock, Parse, and Store a Member

The SAVE service locks and parses a member, and stores that member's statistical, dependency, and historical information all in one service call. The SAVE service calls the LOCK, PARSE, and STORE services.

**Note:** The SAVE service does not parse a member correctly if the member is packed. Make sure that the pack mode is off in the member's profile.

Before you start the SAVE service, the member must exist in the development library you specify. (The LOCK, SAVE, or STORE service can be complete for the member, but this is not necessary.) Upon completion of the SAVE service, the member has been locked and its access key has been set. (You must supply the correct access key for previously locked members.) A typical development scenario follows:

1. Update or create the member.
2. Start the SAVE service to parse the member and store the member's statistical, dependency, and historical information.

For more information on the LOCK, PARSE, and STORE services, see their service descriptions in this chapter.

**Note:** Use of the SAVE service causes SCLM to delete all previously-stored \$list\_info data from the member's dependency and historical information. Each invocation of the SAVE service creates a new set of statistical, dependency, and historical information for the member.

If you need pre-existing historical information, such as user entry data, do not invoke the SAVE service. Use the LOCK, PARSE, and STORE services instead.

### Command Invocation Format

```
FLMCMD SAVE,project,[prj_def]
           ,group,type,member
           ,[authcode],[access_key]
           ,[userid],[language]
           ,[Y|N]
           ,[ddname],[C|U]
           ,[C|U]
           ,[change_code]
```

### Call Invocation Format

```
lstrc := FLMLNK('SAVE ' ,sclm_id
              ,group,type,member
              ,authcode,access_key
              ,{userid|' '},language
              ,{Y|N})
```

## SAVE Service

```
,ddname  
,{C|U}  
,{C|U}  
,{Y|N}  
,$list_info  
,max_prom_group  
,$msg_array);
```

## Parameters

### **project**

The project name. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

### **prj\_def**

The project definition name to be used for the lock, parse, and store. It defaults to the project parameter. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

### **sclm\_id**

An SCLM ID associated with a given project and project definition. The SCLM ID is generated by the INIT service. The maximum parameter length is 8 characters. This parameter is used for FLMLNK only.

### **group**

The group in which the lock, parse, and store are to occur. The specified group must be a development library. The maximum parameter length is 8 characters.

### **type**

The type containing the member. The maximum parameter length is 8 characters.

### **member**

The member to be locked and parsed, and whose accounting information is to be stored. The maximum parameter length is 8 characters.

### **authcode**

The authorization code to be used for the lock. If you do not supply an authcode, SCLM uses default values as follows:

- The authorization code from the existing member if the member being locked exists in the hierarchy
- The default authorization code for the group if the member does not exist in the hierarchy.

The maximum parameter length is 8 characters.

### **access\_key**

The access key to be assigned to the member. The access key is required for any further manipulation of the member until you use the UNLOCK service to remove the access key. It defaults to blank. The maximum parameter length is 16 characters.

### **userid**

User ID of the person requesting the SAVE service. It defaults to the current system user ID. The maximum parameter length is 8 characters.

**language**

The language of the member. The maximum parameter length is 8 characters. You must specify the language the first time you save a member; after that a language name is optional. If not specified, the language will default to the language already defined for the member. Specify a different language name if you wish to change the name of the language defined for the member. Parsers will be called based on the current value specified.

**Y|N**

Y indicates that SCLM is to copy parser listings to the ddname parameter only if parser errors occur. N indicates that SCLM is to copy all parser listings to the ddname. The maximum parameter length is 24 characters.

If the parser for the specified language does not produce a listing, specify Y. The language parsers supplied by SCLM do not produce a listing. If the parser for the specified language does produce a listing, you can specify either value. For more efficient performance, specify Y. Project-specific parsers can produce a listing. The default value for FLMCMD is Y. There is no default value for FLMLNK.

**ddname**

The ddname indicating the destination of the parser listings. If you specify a blank ddname, SCLM does not generate the parser listings. The maximum parameter length is 8 characters.

If the parser for the specified language does not produce a listing, specify a blank ddname. The language parsers supplied by SCLM do not produce a listing. If the parser for the specified language does produce a listing and you specified a ddname, allocate the ddname with the attributes required by the parser. Project-specific parsers can produce a listing.

**C|U**

Specify C to indicate that the member's statistical and dependency information is not to be saved in the event of a parser error; that is, the STORE service is not to be called if the PARSE service completes with a return code of 4. Specify U to indicate that the member's statistical and dependency information is to be saved even in the event of a parser error. The maximum parameter length is 24 characters. The default value for FLMCMD is C. There is no default value for FLMLNK.

**C|U**

Specify C to indicate that a compilation unit cannot be drawn down into a different member. Specify U to indicate that a compilation unit can be drawn down into a different member. The maximum parameter length is 24 characters. The default value for FLMCMD is C. There is no default value for FLMLNK.

**change\_code**

A change\_code to be added to the information obtained by parsing the member. If the member's accounting record lists the change\_code, SCLM updates the date and time stamps for the existing change\_code entry. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

**Y|N**

Y tells SCLM to verify change code records appearing in \$list\_info with the change code verification routine specified in the project definition. N tells SCLM not to verify change code records. The maximum parameter length is 24 characters.

## SAVE Service

This parameter is only valid for the FLMLNK call invocation. SCLM always verifies change\_code records for the FLMCMD command format.

Specify N if your project definition does not specify a change\_code verification routine. Ask your project manager if your project is using a change\_code verification routine.

### **\$list\_info**

An input or output parameter pointing to an array of records that contains change\_code information. SCLM adds any change codes appearing in the array to the information it obtains by parsing the member. If you are not adding change\_code information to the parser information, SCLM can pass a fullword zero buffer address. The array contains only change\_code records.

SCLM deletes all information associated with the member (such as user entry data) previously stored through the STORE service with the \$list\_info parameter.

SCLM ignores the **Date** and **Time Stamp** fields on all change\_code entries in the \$list\_info array. The SAVE service assigns the last change date and time from the member's accounting record to all change\_codes it finds in the array. Note that SCLM does not update the array itself.

SCLM adds all change\_code data listed in \$list\_info to the existing change\_code data in the member's accounting record. If the member's accounting record already lists the change\_code, SCLM updates the date and time stamps for the existing change\_code entry.

This parameter is used for FLMLNK only. See "Pointer Parameters" on page 13 for more details on \$list\_info.

### **max\_prom\_group**

An output parameter indicating the highest group in the hierarchy to which the member can be promoted. Based on the authorization code you used for the lock, SCLM determines the highest group that you can promote this member to. The maximum parameter length is 8 characters. This parameter is used for FLMLNK only.

### **\$msg\_array**

An output parameter pointing to the message array. See "\$msg\_array" on page 13 for more details. This parameter is used for FLMLNK only.

## Return Codes

Additional special services messages are written to the FLMMMSG ddname. See "SCLM Service Messages" on page 22 for more information.

Other return codes might be produced by the FLMCMD or the FLMLNK processor. See "SCLM Service Return Codes" on page 20 for more information.

Possible return codes are:

- 0 Normal completion.
- 4 Warning condition. The \$msg\_array parameter determines the location of this message array.
- 8 Error condition. The \$msg\_array parameter determines the location of this message array.

## Examples

These examples call the SAVE service.

### Command Invocation

```
FLMCMDB SAVE,PROJ1,,USER1,SOURCE,FLM01MD1,,XXX#05,,PASCAL,,,,,CC001234
```

This service command locks, parses, and stores the information for the member FLM01MD1 of the type SOURCE in the USER1 group. The project name is PROJ1 and the access key is XXX#05. Change code CC001234 is to be added to the information obtained by parsing the member with the PASCAL parser. All other parameters are default values.

### Call Invocation

**Note:** This example shows general syntax. Call invocations are language-specific. See “Chapter 3. Sample Programs Using SCLM Services” on page 105 for specific examples.

```
$list_info := NIL; (* Sets the buffer address to X'00000000' *)

lastrc := FLMLNK('SAVE      ', (* service          *)
                 sclm_id,    (* SCLM ID      *)
                 'USER1    ', (* group        *)
                 'SOURCE   ', (* type         *)
                 'FLM01MD1', (* member       *)
                 'TESTAC   ', (* authorization code *)
                 'XXX#05   ', (* access key   *)
                 '        ', (* user ID      *)
                 'PASCAL   ', (* language     *)
                 'Y        ', (* listings     *)
                 'PARSEDD ', (* ddname of listings *)
                 'U        ', (* statistical and dependency info *)
                 'C        ', (* compilation unit *)
                 'Y        ', (* change codes *)
                 $list_info, (* list information pointer *)
                 max_prom_group, (* maximum promotable group *)
                 $msg_array); (* message array pointer *)
```

This service call locks, parses, and stores the information for member FLM01MD1 of the SOURCE type in the USER1 group. The sclm\_id parameter contains a valid SCLM ID returned from the INIT service. The authorization code to be used for the lock verification is TESTAC and the access key is XXX#05. The PASCAL parser parses the member.

SCLM copies parser listings to the PARSEDD ddname only if errors occur. If a parser error does occur, the STORE still completes, SCLM does not draw down compilation units into a different member, and the service verifies all cs found in \$list\_info. SCLM returns all messages produced in the \$msg\_array parameter. You must allocate the PARSEDD ddname before you call FLMLNK.

---

## START—Generate an Application ID for a Services Session

The START service initializes an SCLM services session. It generates an application ID that identifies the services session. You can use the application ID to call the INIT service to initialize an SCLM ID. Each START service invocation needs a matching END service invocation.

## START Service

### Command Invocation Format

You cannot use command procedures to call this service.

### Call Invocation Format

```
lastrc := FLMLNK('START ', appl_id);
```

### Parameters

#### appl\_id

The generated application ID identifying the SCLM services session. Each time you invoke the START service, SCLM generates a unique application ID in this output parameter. The maximum parameter length is 8 characters.

### Return Codes

Additional special services messages are written to the FLMMMSG ddname. See “SCLM Service Messages” on page 22 for more information.

Other return codes might be produced by the FLMLNK processor. See “SCLM Service Return Codes” on page 20 for more information.

Possible return codes are:

- 0 Normal completion.
- 12 Severe error condition. The maximum application ID limit was exceeded.
- 16 Severe error condition. An invalid version of the SCLM table was loaded.
- 20 Severe error condition. An invalid version of the National Language Support (NLS) table was loaded.
- 24 Severe error condition. SCLM is unable to load the SCLM table.
- 28 Severe error condition. SCLM is unable to load the NLS table or the SCLM I/O load module.
- 32 Severe error condition. An invalid parameter list was passed to the requested service.
- 34 Severe error condition. An invalid service was requested.
- 36 Severe error condition. The version of the FLMLNK subroutine does not match the version of the SCLM services module.

### Example

This example calls the START service.

#### Call Invocation

**Note:** This example shows a general syntax. Call invocations are language-specific. See “Chapter 3. Sample Programs Using SCLM Services” on page 105 for specific examples.

```
lastrc := FLMLNK('START ', (* service *)  
                  appl_id); (* application ID *)
```

This service call initializes an SCLM services session.

## STORE—Store Member Information in an Accounting Record

The STORE service saves a member's statistical, dependency, and historical information in an accounting record in the project database. SCLM usually obtains statistical and dependency information by parsing the member, and it is a required input to the STORE service. SCLM retains the historical information in the project database and automatically generates it for the member.

Before you call the STORE service, you must lock the member using the LOCK service, and the member must exist in the development library you specify. After the STORE service ends, the member remains locked and the access key also remains unchanged. A typical development scenario follows:

1. Use the LOCK service to lock the member. The member may or may not yet exist.
2. Update or create the member.
3. Parse the member using the PARSE service.
4. Save the member's statistical, dependency, and historical information using the STORE service.

The STORE service removes duplicate dependency information for each member. For example, if a member is referenced as an include ten times, the STORE service records the reference only once in the accounting information.

When the STORE service receives dependency information, it replaces the existing dependency information rather than appending to it.

Change code information can relate problem report (PR) numbers, change request (CR) numbers, and other information to individual source members. The STORE service can validate change codes you input to the STORE service before it enters them into the accounting records and saves the member.

Like dependency information, all existing user data entries are replaced with the new user data the STORE service receives. User data entries are stored directly into the accounting information for the member. Duplicate entries passed to the STORE service are preserved in the accounting information.

### Command Invocation Format

You cannot use command procedures to call this service.

### Call Invocation Format

```
lastrc := FLMLNK('STORE ' ,sclm_id
                ,group,type,member
                ,access_key
                ,language
                ,{userid|' '}
                ,{C|U}
                ,{Y|N}
                ,$stats_info,$list_info
                ,$msg_array);
```

## STORE Service

### Parameters

#### **sclm\_id**

An SCLM ID associated with a given project and project definition. The INIT service generates the SCLM ID. The maximum parameter length is 8 characters.

#### **group**

The group in which the store is to occur. The specified group must be a development library. The maximum parameter length is 8 characters.

#### **type**

The type containing the member whose information is to be stored. The maximum parameter length is 8 characters.

#### **member**

The member whose information is to be stored. The maximum parameter length is 8 characters.

#### **access\_key**

The access key assigned to the member with the LOCK service. If you supply an incorrect access key, the service fails. The maximum parameter length is 16 characters.

#### **language**

The language of the member. If you used the PARSE service to parse the member, this language should be the same as the one specified as input to the PARSE service. The maximum parameter length is 8 characters. However, if the language is different, you can generate your own \$stats\_info and write an accounting record. You can also use the statistics retrieved from the PARSE service, and it will create a new accounting record with the updated information.

#### **userid**

The user ID of the person requesting the STORE service. It defaults to the current system user ID. The maximum parameter length is 8 characters.

#### **C|U**

C indicates conditional; SCLM does not draw down a compilation unit into a different member. U indicates unconditional; SCLM can draw down a compilation unit into a different member. The maximum parameter length is 24 characters.

#### **Y|N**

Y tells SCLM to verify change code records appearing in \$list\_info with the change code verification routine specified in the project definition. N tells SCLM not to verify change code records. The maximum parameter length is 24 characters.

Ask your project manager if your project is using a change code verification routine. If it is not, specify N.

#### **\$stats\_info**

A pointer to a record containing the member's statistical information. You must have a valid buffer address.

**Note:** If you used the PARSE service to generate the record, you must copy the buffer to the calling program's local storage before calling the STORE service. Failure to copy the buffer to local storage causes unpredictable results.

See “Pointer Parameters” on page 13 for more details on the \$stats\_info parameter and copying the record contents.

### **\$list\_info**

A pointer to an array of records that contains the member’s include, change code and user entry information. If the member has none of this information, you can pass a fullword zero buffer address.

All include and user entry information data listed in \$list\_info replaces existing accounting record data for the member. If you want to maintain existing information (such as user entry history) for the member, it must appear in the \$list\_info parameter.

SCLM ignores the **Date** and **Time Stamp** fields on all change code entries in the \$list\_info array. The STORE service assigns the current system date and time to all change codes it finds in the array. Note that SCLM does not update the array itself.

SCLM adds all change code data listed in \$list\_info to the existing change code information in the member’s accounting record. If the change code is already listed in the member’s accounting record, SCLM updates the date and time stamps for the existing change code entry.

The order of the include entries in \$list\_info determine the order in which the build function processes the member’s dependencies.

Note that SCLM does not permit duplicate record entries in the \$list\_info array. If it encounters duplicate records, it flags an error.

**Note:** If you used the PARSE service to generate the array, you must copy the buffer to the calling program’s local storage before you call the STORE service. Failure to copy the buffer to local storage causes unpredictable results. See “Pointer Parameters” on page 13 for more information on the \$list\_info parameter and copying the array contents.

### **\$msg\_array**

An output parameter pointing to the message array. See “Pointer Parameters” on page 13 for more information on \$msg\_array.

## **Return Codes**

Additional special services messages are written to the FLMMMSG ddname. See “SCLM Service Messages” on page 22 for more information.

Other return codes might be produced by the FLMLNK processor. See “SCLM Service Return Codes” on page 20 for more information.

Possible return codes are:

- 0 Normal completion.
- 4 Warning condition. The \$msg\_array parameter determines the location of this message array.
- 8 Error condition. The \$msg\_array parameter contains the error message associated with this condition.

## **Example**

This example calls the STORE service.

## STORE Service

### Call Invocation

**Note:** This example shows general syntax. Call invocations are language-specific. See “Chapter 3. Sample Programs Using SCLM Services” on page 105 for specific examples.

```
lastrc := FLMLNK('STORE ', (* service *)
                 sclm_id, (* SCLM ID *)
                 'USER1 ', (* group *)
                 'SOURCE ', (* type *)
                 'FLM01MD2', (* member *)
                 'XXX#04 ', (* access key *)
                 'PASCAL ', (* language *)
                 ' ', (* user ID *)
                 'C ', (* compilation unit *)
                 'Y ', (* change codes *)
                 $stats_info, (* statistical information pointer *)
                 $list_info, (* listing information pointer *)
                 $msg_array); (* message array pointer *)
```

This service call stores the statistical and dependency information (obtained from \$stats\_info and \$list\_info) in member FLM01MD2’s accounting record in the project database. The sclm\_id parameter contains a valid SCLM ID returned from the INIT service.

The member FLM01MD2 must exist in the SOURCE type in the USER1 group and must have previously been locked with an access key of XXX#04. The member is identified as a PASCAL member.

SCLM does not draw down compilation units into a different member and it verifies all change codes found in \$list\_info. SCLM returns all messages in the \$msg\_array array.

---

## UNLOCK—Unlock a Member in a Development Library

The UNLOCK service makes a locked member available for updates by another user. If an access key was assigned to the member when it was locked, the UNLOCK service resets the access key to blank.

If SAVE or STORE completes successfully for a member and that member has an access key, you can reset the access key by calling the UNLOCK service.

Before you can promote a member, you must call the UNLOCK service to remove its access key. The PROMOTE service does not promote any member that has an access key. For more information on the LOCK service and access keys, see “LOCK—Lock a Member or Assign an Access Key” on page 63.

## Command Invocation Format

```
FLMCMD UNLOCK,project
           ,[prj_def]
           ,group
           ,type
           ,member
           ,[access_key]
```

## Call Invocation Format

```
l astrc := FLMLNK('UNLOCK ',sclm_id
                ,group
                ,type
                ,member
                ,{access_key|' '})
                , $msg_array);
```

## Parameters

### project

The project name. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

### prj\_def

The project definition name to be used for the unlock. It defaults to project. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

### sclm\_id

An SCLM ID associated with a given project and project definition. The INIT service generates the SCLM ID. The maximum parameter length is 8 characters. This parameter is used for FLMLNK only.

### group

The group in which the member is to be unlocked. The specified group must be a development library. The maximum parameter length is 8 characters.

### type

The type containing the member to be unlocked. The maximum parameter length is 8 characters.

### member

The member to be unlocked. The maximum parameter length is 8 characters.

### access\_key

The access key assigned (with the LOCK or SAVE service) to the member. If you supply an incorrect access key, the unlock fails. The maximum parameter length is 16 characters.

## UNLOCK Service

For the FLMCMD format, the default is blank. For the FLMLNK format, you MUST specify an access key parameter. If you do not want to specify an access key on the FLMLNK, you must pass blanks as the parameter value.

### **\$msg\_array**

An output parameter pointing to the message array. See “\$msg\_array” on page 13 for more details on \$msg\_array. This parameter is used for FLMLNK only.

## Return Codes

Additional special services messages are written to the FLMMSGDS ddname. See “SCLM Service Messages” on page 22 for more information.

Other return codes might be produced by the FLMCMD or the FLMLNK processor. See “SCLM Service Return Codes” on page 20 for more information.

Possible return codes are:

- 0 Normal completion.
- 4 Warning condition. The \$msg\_array parameter determines the location of this message array.
- 8 Error condition. The \$msg\_array parameter contains the error message associated with this condition.

## Examples

These examples call the UNLOCK service.

### Command Invocation

```
FLMCMD UNLOCK,PROJ1,,USER1,SOURCE,FLM01MD1,XXX#05
```

This service command unlocks the FLM01MD1 member of the SOURCE type in the USER1 group. The project name is PROJ1. The access key value for the member is XXX#05.

### Call Invocation

This example shows general syntax. Call invocations are language-specific. See “Chapter 3. Sample Programs Using SCLM Services” on page 105 for language-specific examples.

```
lastrc := FLMLNK('UNLOCK ', (* service *)
                  sclm_id, (* SCLM ID *)
                  'USER1 ', (* group *)
                  'SOURCE ', (* type *)
                  'FLM01MD1', (* member *)
                  'XXX#05 ', (* access key *)
                  $msg_array); (* message array pointer *)
```

This service call unlocks the FLM01MD1 member of the SOURCE type in the USER1 group. The sclm\_id parameter contains a valid SCLM ID returned from the INIT service. The access key value for the member is XXX#05. SCLM returns all messages in the \$msg\_array parameter.

## VERDEL—Delete Version and Audit Information

The VERDEL service deletes the information about a versioned or audited member from SCLM. The information is deleted from the auditing data set defined in the project definition and from the versioning PDS associated with the audit record, if it exists. The partitioned data set used for storing the versions is also updated for deletion of the version. The date and time specified to the service must exactly match the date and time of the audit and version information to delete. Use the VERINFO service to obtain the dates and times of audit and version information.

### Command Invocation Format

```
FLMCMD VERDEL,project
           ,[prj_def]
           ,group
           ,type
           ,member
           ,date
           ,time
           ,[dd_msgs]
           ,[longdate]
```

### Call Invocation Format

```
lastrc := FLMLNK('VERDEL ',sclm_id,
                ,group
                ,type
                ,member
                ,date
                ,time
                ,$msg_array
                [,longdate]);
```

### Parameters

#### project

The project name. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

#### prj\_def

The project definition name. It defaults to the project name. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

#### sclm\_id

An SCLM ID associated with a given project and project definition. The INIT service generates the SCLM ID. The maximum parameter length is 8 characters. This parameter is used for FLMLNK only.

## VERDEL Service

- group** The group associated with the version or audit record. The maximum parameter length is 8 characters.
- type** The type associated with the version or audit record. The maximum parameter length is 8 characters.
- member**  
The member that has the version or audit record. The maximum parameter length is 8 characters.
- date** The date of the version or audit record. The date must be specified in the format given in the ZDATEF ISPF variable. Either the date or the longdate parameter is required. If both are given, the date parameter is used. The length of this parameter is 8 characters.
- time** The time of the version or audit record. The format for the time is HH:MM:SS.hh or HH:MM:SS,hh where HH is the hour from a 24 hour clock, MM is the minute, SS is the seconds and hh is the hundredths of seconds. The length of this parameter is 11 characters.
- dd\_msgs**  
The ddname indicating the destination of the messages generated by the VERDEL service. If you specify a blank ddname, SCLM routes the VERDEL messages to the default output device, such as your terminal. Otherwise, before you call the VERDEL service, you must allocate the ddname. The following attributes should be used: RECFM=F, LRECL=80, BLKSIZE=80. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.
- \$msg\_array**  
An output parameter pointing to the message array. See "Pointer Parameters" on page 13 for more information on \$msg\_array. This parameter is used for FLMLNK only.
- longdate**  
The date of the version or audit record. The longdate, with a 4-character year, must be specified in the national language format. Either the date or the longdate parameter is required. If both are given, the date parameter is used. The length of this parameter is 10 characters.

## Return Codes

Additional special services messages are written to the FLMMMSG ddname. See "SCLM Service Messages" on page 22 for more information.

Other return codes might be produced by the FLMCMD or the FLMLNK processor. See "SCLM Service Return Codes" on page 20 for more information.

Possible return codes are:

- 0 Normal completion. The audit and version information were deleted.
- 8 Error completion. No audit and version information was deleted. No audit record was found that matches the specified criteria.
- 12 Error completion. Refer to the messages for more information.

## VERINFO—Retrieve Version and Audit Information

The VERINFO service retrieves the information about a versioned or audited member into ISPF variables and tables. The service can search a group for the next or previous matching audit record, or retrieve a specific audit record. See “ISPF Variables” on page 17 for a list of the variables updated by this service.

### Command Invocation Format

```
FLMCMD VERINFO,project
    ,[prj_def]
    ,group
    ,type
    ,member
    ,[date]
    ,[time]
    ,[user_info_table]
    ,[include_table]
    ,[change_code_table]
    ,[ada_cu_table]
    ,[FORWARD|BACKWARD|MATCH]
    ,[dd_msgs]
    ,[longdate]
```

### Call Invocation Format

```
lastrc := FLMLNK('VERINFO ',sclm_id,
    ,group
    ,type
    ,member
    ,date
    ,time
    ,user_info_table
    ,include_table
    ,change_code_table
    ,ada_cu_table
    ,FORWARD|BACKWARD|MATCH
    ,$msg_array
    [,longdate]);
```

### Parameters

#### project

The project name. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

#### prj\_def

The project definition name. It defaults to the project name. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

## VERINFO Service

### **sclm\_id**

An SCLM ID associated with a given project and project definition. The INIT service generates the SCLM ID. The maximum parameter length is 8 characters. This parameter is used for FLMLNK only.

**group** The group associated with the version or audit record. The maximum parameter length is 8 characters.

**type** The type associated with the version or audit record. The maximum parameter length is 8 characters.

### **member**

The member that has the version or audit record. The maximum parameter length is 8 characters.

**date** The date of the version or audit record. If omitted or specified as blanks the longdate is used. If both the date and longdate are omitted or specified as blanks, the date will default to 00/00/00. The date must be specified in the format given in the ZDATEF ISPF variable. The length of this parameter is 8 characters.

**time** The time of the version or audit record. If omitted or specified as blanks the time will default to 00:00:00.00. The format for the time is HH:MM:SS.hh or HH:MM:SS,hh where HH is the hour from a 24 hour clock, MM is the minute, SS is the seconds and hh is the hundredths of seconds. The length of this parameter is 11 characters.

### **user\_info\_table**

The name of the ISPF table to contain the user entries from the audit record. The table must be open prior to calling the VERINFO service. A TBADD will be performed for each user entry in the audit record. The maximum parameter length is 8 characters. The following ISPF variables must be used in the table definition in order to have their value stored in the table:

- ZSUNUM - the user entry number
- ZSUENTRY - the user entry data

### **include\_table**

The name of the ISPF table to contain the includes from the audit record. The table must be open prior to calling the VERINFO service. A TBADD will be performed for each include in the audit record. The maximum parameter length is 8 characters. The following ISPF variables must be used in the table definition in order to have their value stored in the table:

- ZSIMBR - the include member name
- ZSISET - the include set name

### **change\_code\_table**

The name of the ISPF table to contain the change codes from the audit record. The table must be open prior to calling the VERINFO service. A TBADD will be performed for each change code in the audit record. The maximum parameter length is 8 characters. The following ISPF variables must be used in the table definition in order to have their value stored in the table:

- ZSCCODE - the change code
- ZSCDATE - the change code date in 2-character date format
- ZSCDAT4 - the change code date in 4-character date format
- ZSCTIME - the change code time

### **ada\_cu\_table**

The name of the ISPF table to contain the ADA compilation units from the

audit record. The table must be open prior to calling the VERINFO service. A TBADD will be performed for each ADA compilation unit in the audit record. The maximum parameter length is 8 characters. The following ISPF variables must be used in the table definition in order to have their value stored in the table:

- ZSDNAME- the ADA compilation unit name
- ZSDTYPE - the ADA compilation unit type

**FORWARD|BACKWARD|MATCH**

FORWARD indicates that if the type name, member name, date, or time do not exactly match an audit record, the information from the next audit record for the group is to be returned. This is the default.

BACKWARD indicates that if the type name, member name, date, or time do not exactly match an audit record, the information from the previous audit record for the group is to be returned.

MATCH indicates that the type name, member name, date, and time must exactly match the type name, member name, date, and time in an audit record.

To retrieve all of the audit records within a group use FORWARD and start with the type name and member name set to blanks and the date and time set to all zeros. If an audit record is found increment the last digit of the time by one before calling the VERINFO service again. Repeat this process until the service indicates that no record was found.

The maximum parameter length is 8 characters.

**dd\_msgs**

The ddname indicating the destination of the messages generated by the VERINFO service. If you specify a blank ddname, SCLM routes the VERINFO messages to the default output device, such as your terminal. Otherwise, before you call the VERINFO service, you must allocate the ddname. The following attributes should be used: RECFM=F, LRECL=80, BLKSIZE=80. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

**\$msg\_array**

An output parameter pointing to the message array. See "Pointer Parameters" on page 13 for more information on \$msg\_array. This parameter is used for FLMLNK only.

**longdate**

The date of the version or audit record. If omitted or specified as blanks the date parameter is used. If both the date and longdate are omitted or specified as blanks, the date will default to 00/00/00. The longdate must be specified in the national language format with a 4-character year. The length of this parameter is 10 characters.

## Return Codes

Additional special services messages are written to the FLMMMSG ddname. See "SCLM Service Messages" on page 22 for more information.

Other return codes might be produced by the FLMCMD or the FLMLNK processor. See "SCLM Service Return Codes" on page 20 for more information.

Possible return codes are:

|  
|

## VERINFO Service

- 0 Normal completion. An audit record exactly matching the specified criteria was found and the information was stored successfully.
- 8 Error completion. No audit record was found for the specified member.
  - If FORWARD was specified, then there are no audit records for the group which match or follow the specified type, member, date, and time.
  - If BACKWARD was specified, then there are no audit records for the group which match or precede the specified type, member, date, and time.
  - If MATCH was specified, then there is not an audit record with the specified group, type, and member name.
- 12 Error completion. Refer to the messages for more information.

---

## VERRECOV—Recover a Version

The VERRECOV service recovers a version of a member from the version data set. For retrieval of a member into the hierarchy the information is recovered from the auditing data set defined in the project definition for the group specified to the service. The date and time specified to the service must exactly match the date and time of the audit record with version information to recover. Use the VERINFO service to obtain the dates and times of audit and version information. The VERINFO service sets variable ZSVMBR, which tells the name of the version member. If ZSVMBR is blank after a VERINFO call, then there is an audit record but no version of the member with this date and time. If ZSVMBR is not blank, then there is a version to recover.

The recovery can be done to a data set outside of SCLM control by specifying the to\_dataset name parameter. To recover into the SCLM project specify the to\_group, to\_type and optionally the authcode. When recovering into SCLM, a lock is first done to lock the member at the specified group and type. If the lock fails no recovery will be performed. Either the to\_dataset must be specified or the to\_group and to\_type must be specified to indicate the location of the recovered member.

### Command Invocation Format

```
FLMCMD VERRECOV,project  
  
    ,[prj_def]  
  
    ,group  
  
    ,type  
  
    ,member  
  
    ,date  
  
    ,time  
  
    ,[to_dataset]  
  
    ,[to_group]  
  
    ,[to_type]  
  
    ,[authcode]  
  
    ,[dd_msgs]  
  
    ,[longdate]
```

## Call Invocation Format

```

lastrc := FLMLNK('VERRECOV',sclm_id,
                ,group
                ,type
                ,member
                ,date
                ,time
                ,to_dataset
                ,to_group
                ,to_type
                ,authcode
                , $msg_array
                [,longdate]);

```

## Parameters

### project

The project name. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

### prj\_def

The project definition name. It defaults to the project name. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

### sclm\_id

An SCLM ID associated with a given project and project definition. The INIT service generates the SCLM ID. The maximum parameter length is 8 characters. This parameter is used for FLMLNK only.

**group** The group associated with the version or audit record. The maximum parameter length is 8 characters.

**type** The type associated with the version or audit record. The maximum parameter length is 8 characters.

### member

The member that has the version or audit record. The maximum parameter length is 8 characters.

**date** The date of the version or audit record. The date must be specified in the format given in the ZDATEF ISPF variable. The length of this parameter is 8 characters.

**time** The time of the version or audit record. The format for the time is HH:MM:SS.hh or HH:MM:SS,hh where HH is the hour from a 24 hour clock, MM is the minute, SS is the seconds and hh is the hundredths of seconds. The length of this parameter is 11 characters.

### to\_dataset

The name of the data set to hold the recovered member. The data set must be a sequential data set or a PDS without the member name specified. The

## VERRECOV Service

data set name must be fully qualified without quotes. If the data set is a PDS the member name will be the name of the member being recovered. If this parameter is specified then the `to_group` and `to_type` parameters must not be specified. The maximum parameter length is 44 characters.

### **to\_group**

The name of the group to hold the recovered member. The group must be a development group (lowest level of the hierarchy). This parameter requires that the `to_type` also be specified. If this parameter is specified then the `to_dataset` must not be specified. The maximum parameter length is 8 characters.

### **to\_type**

The name of the type to hold the recovered member. This parameter requires that the `to_group` also be specified. If this parameter is specified then the `to_dataset` must not be specified. The maximum parameter length is 8 characters.

### **authcode**

The authorization code to be used for locking the member in the hierarchy. The authorization code must be valid for the group specified in the `to_group` parameter. If this parameter is not specified and `to_group` is specified then SCLM will attempt to lock the member with the authorization code that is in the audit record. This parameter requires that the `to_group` and `to_type` also be specified. If this parameter is specified then the `to_dataset` must not be specified. The maximum parameter length is 8 characters.

### **dd\_msgs**

The ddname indicating the destination of the messages generated by the VERRECOV service. If you specify a blank ddname, SCLM routes the VERRECOV messages to the default output device, such as your terminal. Otherwise, before you call the VERRECOV service, you must allocate the ddname. The following attributes should be used: RECFM=F, LRECL=80, BLKSIZE=80. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

### **\$msg\_array**

An output parameter pointing to the message array. See "Pointer Parameters" on page 13 for more information on `$msg_array`. This parameter is used for FLMLNK only.

### **longdate**

The date of the version or audit record. If omitted or specified as blanks, the date parameter is used. If both the date and `longdate` are omitted or specified as blanks, the date will default to 00/00/00. The `longdate`, with a 4-character year, must be specified in the national language format. The length of this parameter is 10 characters.

## Return Codes

Additional special services messages are written to the FLMMMSGs ddname. See "SCLM Service Messages" on page 22 for more information.

Other return codes might be produced by the FLMCMD or the FLMLNK processor. See "SCLM Service Return Codes" on page 20 for more information.

Possible return codes are:

0 Normal completion. The audit and version information were recovered.

- 8 Error completion. No audit and version information was recovered. No audit record was found that matches the specified criteria.
- 10 Error completion. No audit and version information was recovered. The member could not be locked with the specified authorization code.
- 12 Error completion. Refer to the messages for more information.

## VERRECOV Service

---

## Chapter 3. Sample Programs Using SCLM Services

This chapter contains the following:

- An example of Pascal program invocations that call the following SCLM services in this order:
  - START
  - INIT
  - LOCK
  - PARSE
  - STORE
  - BUILD
  - FREE
  - END
- A PL/I example that illustrates SCLM service procedures.

Command interface examples written in REXX for the Audit and Versioning Services can be found in ISP.SISPSAMP members FLMSACCT (ACCTINFO), FLMSVERI (VERINFO), FLMSVERR (VERRECOV) and FLMSVERD (VERDEL).

The source code for the Pascal sample programs is found in ISP.SISPSAMP members FLMSRV1, FLMSRV1D, and FLMSRV1S. The source code for the PL/I sample program is found in ISP.SISPSAMP member FLMPLSM.

---

### Pascal Example

You can use the following sample Pascal programs to migrate and build a component registered with SCLM. SCLM prompts you for responses as it processes the component. The program prolog contains a description of the required ddnames to be allocated before you start the program.

**Note:** All requested input parameters must be entered in upper case characters.

# Main Program FLMSRV1

```
PROGRAM FLMSRV1 ;

(*****)
(*)
(* This program allows you to call SCLM services from a *)
(* Pascal program. *)
(*)
(*****)
(***** ALL REQUESTED INPUT PARAMETERS MUST BE ENTERED *****)
(***** IN UPPER CASE. *****)
(*****)
(*****)
(*)
(* The function of this program is to register a software component *)
(* with SCLM and then build it. *)
(* The member in the SCLM controlled library (PDS) to be processed *)
(* is referenced by the variables project.group.type(member). *)
(* You must allocate the following ddnames as specified below: *)
(*)
(* PRSLIST - for parser listings (RECFM=VBA,LRECL=137,BLKSIZE=3120) *)
(* BLDMSGs - for build messages (RECFM=F, LRECL=80, BLKSIZE=80) *)
(* BLDREPT - for build report (RECFM=FBA,LRECL=80, BLKSIZE=3120) *)
(* BLDLIST - for build listings (RECFM=VBA,LRECL=137,BLKSIZE=3120) *)
(* BLDEXIT - for build user exit (RECFM=FB, LRECL=160,BLKSIZE=3200) *)
(*****)
(*****)
(*)
(*) Declare program and interface constants (*)
(*****)
CONST

    (* Declare the maximum number of records the accounting record *)
    (* list information array can hold. *)
    max_list_info_entries = 200 ;

    (* Declare the required ddnames as constants. *)
    bldmsgs = 'BLDMSGs' ;
    bldrept = 'BLDREPT' ;
    bldlist = 'BLDLIST' ;
    bldexit = 'BLDEXIT' ;

    (* Include SCLM Interface common type declarations. *)
    %INCLUDE FLMSRV1D ;

    (* Include SCLM Interface procedure definitions. *)
    %INCLUDE FLMSRV1S ;

(*****)
(*)
(*) Declare program local variables (*)
(*****)
VAR

    $acct_info          : $acct_info_type          ;
    $list_info          : $list_info_type          ;
    $list_info_copy    : $list_info_type          ;
    $stats_info         : $stats_info_type         ;
    $stats_info_copy   : $stats_info_type         ;
    $msg_array          : $msg_array_type          ;
```

```

breport_check          : char24          ;
build_scope            : char24          ;
build_mode             : char24          ;
access_key             : char16         ;
appl_id                : char8           ;
authcode               : char8           ;
ddname                 : char8           ;
error_listings_only   : char24          ;
found_group            : char8           ;
language               : char8           ;
group                  : char8           ;
listing_check         : char24          ;
max_prom_group        : char8           ;
msg_line               : char80         ;
prefix_userid         : char17          ;
project                : char8           ;
project_def            : char8           ;
retncode               : INTEGER        ;
pds_type               : char8           ;
member                 : char8           ;
SCLM_id                : char8           ;
sub_drawdown_mode     : char24          ;
userid                 : char8           ;
verify_cc              : char24          ;

(*****
*)          Define the main program          (*)
(*****

```

```

BEGIN

```

```

(* Initialize terminal I/O. *)
TERMIN (INPUT) ;
TERMOUT(OUTPUT) ;

(* Initialize some working variables. *)
$stats_info_copy := NIL ;
$list_info_copy := NIL ;

(* Get the PDS/member name of the component to process. *)
WRITELN ('Enter the name of the project to process. ');
READLN (project);
WRITELN ('Enter the name of the project definition to process. ');
READLN (project_def);
IF
  (project_def = ' ')
THEN
  project_def := project;
WRITELN ('Enter the name of the development group to process. ');
READLN (group);
WRITELN ('Enter the name of the type to process. ');
READLN (pds_type);
WRITELN ('Enter the name of the member to process. ');
READLN (member);
WRITELN ('Enter the language of the source member to register. ');
READLN (language);

```

```

(* Issue a request to begin an SCLM service session. *)
SRVSTART ( appl_id,
           retncode );

(* Continue processing only if the request succeeded. *)
IF
  retncode <> 0
THEN
  WRITELN ('SCLM service START failed, error code = ', retncode:-3 )
ELSE BEGIN
  (* Issue a request to initialize an SCLM ID. *)
  msg_line := ' ' ;
  SRVINIT ( appl_id,
            project,
            project_def,
            SCLM_id,
            msg_line,
            retncode );

  (* Continue processing only if the request succeeded. *)
  IF
    retncode <> 0
  THEN BEGIN
    WRITELN ('SCLM service INIT failed, error code = ', retncode:-3 );
    WRITELN ( msg_line );
  END

  ELSE BEGIN

    (* Issue a request to lock the component. *)
    authcode := ' ' ;
    $acct_info := NIL ;
    $list_info := NIL ;
    $msg_array := NIL ;
    SRVLOCK ( SCLM_id,
              group,
              pds_type,
              member,
              authcode,
              ' ', (* access_key *)
              userid,
              found_group,
              max_prom_group,
              $acct_info,
              $list_info,
              $msg_array,
              retncode );

    (* If the lock failed, print associated error messages. *)
    IF
      retncode <> 0
    THEN BEGIN
      WRITELN ('SCLM service LOCK failed, error code = ',
              retncode:-3);
      PUTMSG ( $msg_array );
    END
  END

```

```

ELSE BEGIN

    (* Display some of the accounting record fields *)
    WRITELN ('The component has been locked. ');
    WRITELN ('The component last changed date is: ',
             $acct_info@.change_date );
    WRITELN ('The component last changed time is: ',
             $acct_info@.change_time );
    WRITELN ('The component change-userid is: ',
             $acct_info@.change_userid );
    WRITELN ('The component version number is: ',
             $acct_info@.member_version:-3 );
END;

(* Continue processing only if the member has been locked. *)
IF
    retncode = 0
THEN BEGIN

    (* Issue a request to parse the component to obtain *)
    (* the statistical information SCLM requires. *)
    $stats_info := NIL ;

    SRVPARSE ( SCLM_id,
              group,
              pds_type,
              member,
              language,
              'Y', (* error_listings_only = yes *)
              'PRSLIST', (* ddname *)
              $stats_info,
              $list_info,
              $msg_array,
              retncode );

    (* If the parse failed, print associated error messages. *)
    IF
        retncode <> 0
    THEN BEGIN
        WRITELN ('SCLM service PARSE failed, ',
                 'error code = ',retncode:-3 );
        PUTMSG ( $msg_array );
    END
    ELSE BEGIN

        (* Copy all buffered service output into new buffers so *)
        (* subsequent service calls do not delete the information. *)
        WRITELN ('The component has been parsed. ');
        NEW ( $stats_info_copy );
        $stats_info_copy@ := $stats_info@ ;

        NEW ( $list_info_copy );
        COPYLIST ($list_info, $list_info_copy );
    END;
END;

```

```

(* Continue processing only if the member has been parsed. *)
IF
  retncode = 0
THEN BEGIN

  (* Issue a request to register the component with SCLM      *)
  $stats_info := $stats_info_copy ;
  $list_info := $list_info_copy ;

  SRVSTORE ( SCLM_id,
             group,
             pds_type,
             member,
             ' ',          (* access_key *)
             language,
             userid,
             'C',          (* sub_drawdown_mode = cond. *)
             'N',          (* verify_cc = no *)
             $stats_info,
             $list_info,
             $msg_array,
             retncode );

  (* If the store failed, print associated error messages. *)
  IF
    retncode <> 0
  THEN BEGIN
    WRITELN ('SCLM service STORE failed, ',
            'error code = ',retncode:-3);
    PUTMSG ( $msg_array );
  END;
END;

(* Continue processing only if the member has been stored. *)
IF
  retncode = 0
THEN BEGIN

  (* Issue a request to build the component *)
  (* registered with SCLM. *)
  WRITELN ('The component has been stored. ');
  prefix_userid := STR(userid) ;

  SRVBUILD ( SCLM_id,
            group,
            pds_type,
            member,
            userid,
            'N',          (* build_scope = normal      *)
            'C',          (* build_mode = conditional *)
            'N',          (* listing_check = no       *)
            'Y',          (* breport_check = yes      *)
            prefix_userid,
            bldmsgs,      (* dd_bldmsgs *)
            bldrept,      (* dd_bldrept *)
            bldlist,      (* dd_bldlist *)
            bldexit,      (* dd_bldexit *)
            retncode );

```

```

(* If the build failed, print error messages. *)
IF
  retncode <> 0
THEN BEGIN
  WRITELN ('SCLM service BUILD failed, ',
           'error code = ',retncode:-3 );
  WRITELN ('See the data set allocated to ddname=BLDMSGs ',
           'for associated error messages.' );
END
ELSE
  WRITELN ('The component has been built.' );
END;

(* Issue a request to free the SCLM ID. *)
SRVFREE ( SCLM_id,
          msg_line,
          retncode );
END;          (* INIT succeeded *)

(* Issue a request to end this SCLM service session. *)
SRVEND ( appl_id,
         msg_line,
         retncode );
END;          (* START succeeded *)

(* Free buffer memory if it is still allocated. *)
IF
  $stats_info_copy <> NIL
THEN
  DISPOSE ( $stats_info_copy );

IF
  $list_info_copy <> NIL
THEN
  DISPOSE ( $list_info_copy );

END.          (* Main Program *)

```

## Included Member FLMSRV1D

```
(*****)  
(* FLMSRV1D *)  
(* *)  
(* This member is included by program FLMSRV1 *)  
(* *)  
(*****)  
(*****)  
(* Declare Common SCLM Interface Types *)  
(*****)  
TYPE  
  
(* Declare arrays of various sizes. *)  
char2 = PACKED ARRAY (. 1.. 2 .) OF CHAR ;  
char4 = PACKED ARRAY (. 1.. 4 .) OF CHAR ;  
char6 = PACKED ARRAY (. 1.. 6 .) OF CHAR ;  
char8 = PACKED ARRAY (. 1.. 8 .) OF CHAR ;  
char12 = PACKED ARRAY (. 1.. 12 .) OF CHAR ;  
char16 = PACKED ARRAY (. 1.. 16 .) OF CHAR ; (* type = ALPHA *)  
char17 = PACKED ARRAY (. 1.. 17 .) OF CHAR ;  
char24 = PACKED ARRAY (. 1.. 24 .) OF CHAR ;  
  
char43 = PACKED ARRAY (. 1.. 43 .) OF CHAR ;  
char80 = PACKED ARRAY (. 1.. 80 .) OF CHAR ;  
char110 = PACKED ARRAY (. 1..110 .) OF CHAR ;  
char128 = PACKED ARRAY (. 1..128 .) OF CHAR ;  
  
(* Declare a pointer to an SCLM message array. *)  
$msg_array_type = @ msg_array_type ;  
msg_array_type = PACKED ARRAY (. 1 .. 9999 .) OF char80 ;  
  
(* Declare a pointer to the static portion *)  
(* of an SCLM accounting record. *)  
$acct_info_type = @ acct_info_type ;  
acct_info_type =  
RECORD  
    acct_group          : char8 ;  
    acct_type          : char8 ;  
    acct_member        : char8 ;  
    SCLM_version       : char2 ;  
    accounting_status  : CHAR ;  
    change_date        : char8 ;
```

```
change_time          : char6  ;
change_group         : char8   ;
change_userid        : char8   ;
member_version       : INTEGER ;
language             : char8   ;
authorization_code   : char8   ;
authorization_code_change : char8 ;
access_key           : char16  ;
creation_date        : char8   ;
creation_time        : char6   ;
map_date             : char8   ;
map_time             : char6   ;
predecessor_date     : char8   ;
predecessor_time     : char6   ;
promote_date         : char8   ;
promote_time         : char6   ;
promote_userid       : char8   ;
db_qual              : char8   ;
translator_version   : char8   ;
```

```

map_name          : char8 ;
  map_type        : char8  ;
  language_version : char8  ;
  total_lines     : INTEGER ;
  comment_lines   : INTEGER ;
  non_comment_lines : INTEGER ;
  blank_lines     : INTEGER ;
  prolog_lines    : INTEGER ;
  total_stmts     : INTEGER ;
  comment_stmts   : INTEGER ;
  control_stmts   : INTEGER ;
  assignment_stmts : INTEGER ;
  non_comment_stmts : INTEGER ;
  number_of_user_entries : INTEGER ;
  number_of_includes : INTEGER ;
  number_of_changecodes : INTEGER ;
  number_of_cus   : INTEGER ;
END;

(* Declare a pointer to the statistical portion *)
(* of an SCLM accounting record. *)
$stats_info_type = @ stats_info_type ;
stats_info_type =
  RECORD
    total_lines      : INTEGER ;
    comment_lines    : INTEGER ;
    non_comment_lines : INTEGER ;
    blank_lines      : INTEGER ;
    prolog_lines     : INTEGER ;
    total_stmts      : INTEGER ;
    comment_stmts    : INTEGER ;
    control_stmts    : INTEGER ;
    assignment_stmts : INTEGER ;
    non_comment_stmts : INTEGER ;
  END;

(* Declare an SCLM list-info change code entry. *)
change_code_record_type =
  RECORD
    change_code : char8 ;
    date        : char8 ;
    time        : char6 ;
  END;

(* Declare an SCLM list-info EXTD entry. *)
extd_record_type =
  RECORD
    extd_group : char8 ;
    extd_type  : char8 ;
    extd_name  : char43 ;
    date       : char8 ;
    time       : char6 ;
  END;

(* Declare an SCLM list-info compilation unit entry. *)
cu_record_type =
  RECORD
    cu_name      : char110 ;
    cu_type      : CHAR    ;
    generic_flag : CHAR    ;
  END;

```

```

        depend_cu_name      : char110 ;
        depend_cu_type     : CHAR    ;
        depend_cu_depend_type : CHAR    ;
    END;

(* Declare an SCLM accounting record list-info entry. *)
include_record_type =
    RECORD
        member      : char8 ;
        include_set : char8 ;
    END;

(* Declare an SCLM accounting record list-info entry overlay. *)
list_info_record_type =
    RECORD
        record_kind : char4 ;
        CASE INTEGER OF
            1: ( member      : char8           );
            2: ( compool    : char8           );
            3: ( change_code_record : change_code_record_type );
            4: ( user_entry  : char128        );
            5: ( cu_record   : cu_record_type );
            6: ( extd_record : extd_record_type );
            7: ( include_record : include_record_type );
        END;

END;

(* Declare a pointer to an SCLM accounting record list-info array. *)
$list_info_type = @ list_info_type ;
list_info_type = PACKED ARRAY (.1..max_list_info_entries.)
                OF list_info_record_type ;

```

## Included Member FLMSRV1S

```

(*****
*) FLMSRV1S  SCLM SERVICE INTERFACE PROCEDURE DEFINITIONS *)
(*)
(*) This member is included by program FLMSRV1 *)
(*)
(*****

(*****
*) SCLM START Service Interface *)
(*****
PROCEDURE SRVSTART ( VAR appl_id : char8 ;
                    VAR rc      : INTEGER );

    FUNCTION FLMLNK ( CONST service : char8 ;
                     VAR appl_id  : char8 ): INTEGER ;
        FORTRAN ;

BEGIN
    rc := FLMLNK ('START ', appl_id );
END;

```

```

(*****
*)          SCLM INIT Service Interface          *)
(*****
PROCEDURE SRVINIT ( CONST appl_id   : char8   ;
                    CONST project  : char8   ;
                    CONST project_def : char8   ;
                    VAR  SCLM_id   : char8   ;
                    VAR  msg_line  : char80  ;
                    VAR  rc        : INTEGER ) ;

        FUNCTION FLMLNK ( CONST service : char8   ;
                          CONST appl_id : char8   ;
                          CONST project  : char8   ;
                          CONST project_def : char8   ;
                          VAR  SCLM_id   : char8   ;
                          VAR  msg_line  : char80 ) : INTEGER ;
        FORTRAN ;

BEGIN

rc := FLMLNK ('INIT   ', appl_id, project, project_def, SCLM_id,
             msg_line );
END;

(*****
*)          SCLM FREE Service Interface          *)
(*****
PROCEDURE SRVFREE ( CONST SCLM_id : char8   ;
                   VAR  msg_line : char80  ;
                   VAR  rc       : INTEGER ) ;

        FUNCTION FLMLNK ( CONST service : char8   ;
                          CONST SCLM_id : char8   ;
                          VAR  msg_line : char80 ) : INTEGER ;
        FORTRAN ;

BEGIN
rc := FLMLNK ('FREE   ', SCLM_id, msg_line );
END;

```

```

(*****)
(*          SCLM END Service Interface          *)
(*****)
PROCEDURE SRVEND ( CONST appl_id : char8      ;
                   VAR  msg_line : char80    ;
                   VAR  rc       : INTEGER ) ;

FUNCTION FLMLNK ( CONST service : char8      ;
                  CONST appl_id : char8      ;
                  VAR  msg_line : char80    ) : INTEGER ;
FORTRAN ;

BEGIN
  rc := FLMLNK ( 'END      ', appl_id, msg_line );
END;

(*****)
(*          SCLM BUILD Service Interface        *)
(*****)
PROCEDURE SRVBUILD ( CONST SCLM_id      : char8      ;
                    CONST group       : char8      ;
                    CONST pds_type    : char8      ;
                    CONST member      : char8      ;
                    CONST userid      : char8      ;
                    CONST build_scope : char24     ;
                    CONST build_mode  : char24     ;
                    CONST listing_check : char24    ;
                    CONST breport_check : char24    ;
                    CONST prefix_userid : char17    ;
                    CONST dd_bldmsgs  : char8      ;
                    CONST dd_bldrept  : char8      ;
                    CONST dd_bldlist  : char8      ;
                    CONST dd_bldexit  : char8      ;
                    VAR  rc           : INTEGER ) ;

FUNCTION FLMLNK ( CONST service      : char8      ;
                  CONST SCLM_id     : char8      ;
                  CONST group       : char8      ;
                  CONST pds_type    : char8      ;
                  CONST member      : char8      ;
                  CONST userid      : char8      ;
                  CONST build_scope : char24     ;
                  CONST build_mode  : char24     ;
                  CONST listing_check : char24    ;
                  CONST breport_check : char24    ;
                  CONST prefix_userid : char17    ;
                  CONST dd_bldmsgs  : char8      ;
                  CONST dd_bldrept  : char8      ;
                  CONST dd_bldlist  : char8      ;
                  CONST dd_bldexit  : char8      ) : INTEGER ;
FORTRAN ;

BEGIN
  rc := FLMLNK ( 'BUILD  ', SCLM_id, group, pds_type, member, userid,
                build_scope, build_mode, listing_check, breport_check,
                prefix_userid,
                dd_bldmsgs, dd_bldrept, dd_bldlist, dd_bldexit );
END;

```

```

(*****)
(*          SCLM LOCK Service Interface          *)
(*****)
PROCEDURE SRVLOCK ( CONST SCLM_id      : char8      ;
                   CONST group       : char8      ;
                   CONST pds_type    : char8      ;
                   CONST member      : char8      ;
                   CONST authcode    : char8      ;
                   CONST access_key  : char16     ;
                   CONST userid      : char8      ;
                   VAR  found_group   : char8      ;
                   VAR  max_prom_group : char8      ;
                   VAR  $acct_info   : $acct_info_type ;
                   VAR  $list_info   : $list_info_type ;
                   VAR  $msg_array   : $msg_array_type ;
                   VAR  rc            : INTEGER ) ;

FUNCTION FLMLNK ( CONST service      : char8      ;
                 CONST SCLM_id     : char8      ;
                 CONST group       : char8      ;
                 CONST pds_type    : char8      ;
                 CONST member      : char8      ;
                 CONST authcode    : char8      ;
                 CONST access_key  : char16     ;
                 CONST userid      : char8      ;
                 VAR  found_group   : char8      ;
                 VAR  max_prom_group : char8      ;
                 VAR  $acct_info   : $acct_info_type ;
                 VAR  $list_info   : $list_info_type ;
                 VAR  $msg_array   : $msg_array_type ;
                 INTEGER ) ;

FORTRAN ;

BEGIN
  rc := FLMLNK ( 'LOCK  ', SCLM_id, group, pds_type, member, authcode,
               access_key, userid,
               found_group, max_prom_group,
               $acct_info, $list_info, $msg_array );
END;

(*****)
(*          SCLM PARSE Service Interface          *)
(*****)
PROCEDURE SRVPARSE ( CONST SCLM_id      : char8      ;
                   CONST group       : char8      ;
                   CONST pds_type    : char8      ;
                   CONST member      : char8      ;
                   CONST language    : char8      ;
                   CONST error_listings_only : char24 ;
                   CONST ddname     : char8      ;
                   VAR  $stats_info  : $stats_info_type ;
                   VAR  $list_info   : $list_info_type ;
                   VAR  $msg_array   : $msg_array_type ;
                   VAR  rc            : INTEGER ) ;

FUNCTION FLMLNK ( CONST service      : char8      ;
                 CONST SCLM_id     : char8      ;
                 CONST group       : char8      ;
                 CONST pds_type    : char8      ;
                 CONST member      : char8      ;

```

```

CONST language      : char8      ;
CONST error_listings_only : char24 ;
CONST ddname        : char8      ;
VAR $stats_info     : $stats_info_type ;
VAR $list_info      : $list_info_type ;
VAR $msg_array      : $msg_array_type ) :
                                           INTEGER ;

FORTRAN ;

BEGIN
  rc := FLMLNK ('PARSE ', SCLM_id, group, pds_type, member, language,
               error_listings_only, ddname,
               $stats_info, $list_info, $msg_array );
END;

(*****
*) SCLM STORE Service Interface (*)
(*****)
PROCEDURE SRVSTORE (CONST SCLM_id      : char8      ;
                   CONST group        : char8      ;
                   CONST pds_type     : char8      ;
                   CONST member       : char8      ;
                   CONST access_key   : char16     ;
                   CONST language     : char8      ;
                   CONST userid       : char8      ;
                   CONST sub_drawdown_mode : char24 ;
                   CONST verify_cc    : char24     ;
                   CONST $stats_info  : $stats_info_type ;
                   CONST $list_info   : $list_info_type ;
                   VAR $msg_array     : $msg_array_type ;
                   VAR rc             : INTEGER ) ;

FUNCTION FLMLNK ( CONST service      : char8      ;
                 CONST SCLM_id     : char8      ;
                 CONST group        : char8      ;
                 CONST pds_type     : char8      ;
                 CONST member       : char8      ;
                 CONST access_key   : char16     ;
                 CONST language     : char8      ;
                 CONST userid       : char8      ;
                 CONST sub_drawdown_mode : char24 ;
                 CONST verify_cc    : char24     ;
                 CONST $stats_info  : $stats_info_type ;
                 CONST $list_info   : $list_info_type ;
                 VAR $msg_array     : $msg_array_type ) :
                                           INTEGER ;

FORTRAN ;

BEGIN
  rc := FLMLNK ('STORE ', SCLM_id, group, pds_type, member,
               access_key, language, userid, sub_drawdown_mode,
               verify_cc, $stats_info, $list_info, $msg_array );
END;

```

```

(*****
*)      Procedure to print the contents of an SCLM $msg_array.      *)
(*****
PROCEDURE PUTMSG ( VAR $msg_array : $msg_array_type );

VAR
    indx : INTEGER ;

BEGIN
    (* Procedure PUTMSG *)

    (* Print message header information. *)
    WRITELN ('Message array information..');

    (* If the pointer is valid, print the information. *)
    IF
        $msg_array <> NIL
    THEN BEGIN

        (* Loop through the list information. *)
        indx := 1 ;
        WHILE
            $msg_array@(.indx.) <> 'END'
        DO BEGIN
            WRITELN ( $msg_array@(.indx.) ) ;
            indx := indx + 1
                ;
        END;
    END;
    (* if $msg_array <> nil *)

    (* Reset :q.$msg_array:eq. to NIL. *)
    $msg_array := NIL;

END;
    (* Procedure PUTMSG *)

(*****
*)      Procedure to copy an accounting record list information array. *)
(*****
PROCEDURE COPYLIST ( CONST $list_info      : $list_info_type ;
                    VAR $list_info_copy : $list_info_type ) ;

VAR
    indx : INTEGER ;

BEGIN
    (* Procedure COPYLIST *)

    (* Only perform the copy if the input list is not nil. *)
    IF
        $list_info <> NIL
    THEN BEGIN

        (* Allocate storage for the copy list if the caller *)
        (* has not yet done this. *)
        IF
            $list_info_copy = NIL
        THEN
            NEW ( $list_info_copy ) ;

        (* Loop through the list information, copying entry-by-entry. *)
        indx := 1 ;
        REPEAT
            $list_info_copy@(.indx.) := $list_info@(.indx.) ;

```

```

        indx := indx + 1                                ;
UNTIL
    ($list_info@(.indx-1.).record_kind = 'END ')
    OR
    (indx > max_list_info_entries) ;

(* Check for overflow condition. *)
IF
    indx > max_list_info_entries
THEN BEGIN
    WRITELN ('*** ERROR *** List information array overflowed!');
    WRITELN ('*** ERROR *** Increase size of program constant. ');
END;
END;                                (* if $list_info <> nil *)
END;                                (* Procedure COPYLIST *)

```

---

## PL/I Example

The following is a sample PL/I program for SCLM service procedures.

```

FLMPLSM: PROC (PARMS) OPTIONS(MAIN NOEXECOPS);
/*****/
/*
/* PL/I PROGRAM WHICH CALLS SCLM SERVICES
/*
/* PROCEDURES IN THIS PROGRAM:
/*
/* -SCLSTRT START SCLM SESSION
/* -SCLINIT INIT SCLM_ID
/* -SCLEDIT EDIT SCLM_SOURCE MEMBER
/* -SCLFREE FREE SCLM_ID
/* -SCLEND END SCLM_SESSION
/*
/*****/
/*
/* DECLARATIONS
/*
/*****/
DCL PLIRETV BUILTIN ;
DCL FLMLNK ENTRY EXTERNAL OPTIONS(ASM,INTER,RETCODE) ;
DCL ISPLINK ENTRY EXTERNAL OPTIONS(ASM,INTER,RETCODE) ;

/*****/
/* PARAMETERS USED IN THIS PROGRAM
/*****/
DCL PARMS CHAR(80) VARYING;

DCL 1 PARM,
      2 PARM1 CHAR(8) INIT('') ,
      2 DELM1 CHAR(1) INIT('') ,
      2 PARM2 CHAR(8) INIT('') ,
      2 DELM2 CHAR(1) INIT('') ,
      2 PARMX CHAR(62) INIT('') ;

/*****/
/* VARIABLES USED BY SCLM SERVICES
/*****/
DCL SERVICE CHAR(8) INIT(' ') ;
DCL APPL_ID CHAR(8) INIT(' ') ;
DCL SCLM_ID CHAR(8) INIT(' ') ;
DCL PRJ_DEF CHAR(8) INIT(' ') ;
DCL PROJECT CHAR(8) INIT(' ') ;
DCL MSG_LINE CHAR(80) INIT(' ') ;

DCL Y CHAR(24) INIT('Y '),
      C CHAR(24) INIT('C '),
      N CHAR(24) INIT('N ');

DCL SLMLIB CHAR(8),
      SLMLIB2 CHAR(8),

```

```

SLMLIB3          CHAR(8),
SLMLIB4          CHAR(8),
ALL_HIER         CHAR(24),
IMAC             CHAR(8),
PROF             CHAR(8),
CONFIRM         CHAR(24),
MIX             CHAR(24),
WS              CHAR(24),
PRESERVE        CHAR(24),
AUTHCODE        CHAR(8),
CHGCODE         CHAR(8),
VOLSER          CHAR(8),
SLMPROJ         CHAR(8),
SLMALTP         CHAR(8),
SLMTYP          CHAR(8),
SLMMEM          CHAR(8),
MSGLIST         CHAR(80);

DCL BLNK8        CHAR(8) INIT(' '),
DDNAME          CHAR(8),
DONE            BIT(1);

/*****/
/* MAIN PROGRAM LOGIC */
/*****/

PARM1           = 'PROJECT ';
PARM2           = 'ALT_PROJ';
PROJECT         = PARM1;
PRJ_DEF         = PARM2;
IF PRJ_DEF = ' ' THEN PRJ_DEF = PROJECT ;

CALL SCLSTRT ;
CALL SCLINIT ;
CALL SCLEDIT ;
CALL SCLFREE ;
CALL SCLEND ;

/*****/
/* GENERATE AN APPLICATION ID FOR SCLM SESSION */
/*****/
SCLSTRT: PROC ;
SERVICE        = 'START';
APPL_ID         = '';

CALL FLMLNK (SERVICE , APPL_ID) ;
RETCODE         = PLIRETV() ;

END SCLSTRT ;

/*****/
/* INITIALIZE SCLM ID FOR SERVICES */
/*****/
SCLINIT: PROC ;
SERVICE        = 'INIT' ;
SCLM_ID         = '';

CALL FLMLNK (SERVICE , APPL_ID
            , PROJECT

```

```

                , PRJ_DEF
                , SCLM_ID
                , MSG_LINE) ;
RETCODE      = PLIRETV() ;

END SCLINIT ;

/*****/
/* EDIT A MEMBER IN THE PROJECT HIERARCHY */
/*****/
SCLEDIT: PROC ;
SLMLIB      = 'DEV1      ' ;
SLMLIB2     = '          ' ;
SLMLIB3     = '          ' ;
SLMLIB4     = '          ' ;
SLMTYP      = 'SOURCE   ' ;
SLMMEM      = 'MEMBER1  ' ;
SERVICE    = 'EDIT     ' ;
DDNAME      = 'EDIT     ' ;
ALL_HIER    = N ;
IMAC        = '          ' ;
PROF        = '          ' ;
CONFIRM     = N ;
MIX         = N ;
WS          = N ;
PRESERVE    = 'Y' ;
AUTHCODE    = ' ' ;
CHGCODE     = ' ' ;
VOLSER      = BLNK8 ;

CALL FLMLNK(SERVICE,SCLM_ID,SLMLIB,
            SLMLIB2,SLMLIB3,SLMLIB4,
            SLMTYP,SLMMEM,ALL_HIER,
            IMAC,PROF,CONFIRM,MIX,WS,
            PRESERVE,AUTHCODE,CHGCODE,
            VOLSER,DDNAME);

RETCODE      = PLIRETV() ;
IF RETCODE > 0 THEN
    CALL ISPLINK('BROWSE ', 'SCLM.MSGS ');

END SCLEDIT ;

/*****/
/* FREE SCLM ID */
/*****/
SCLFREE: PROC ;
SERVICE     = 'FREE     ' ;

CALL FLMLNK (SERVICE, SCLM_ID
            , MSG_LINE) ;
RETCODE      = PLIRETV() ;

END SCLFREE ;

/*****/
/* END AN SCLM SERVICES SESSION */
/*****/
SCLEND: PROC ;

```

```
SERVICE      = 'END      ' ;  
CALL FLMLNK (SERVICE, APPL_ID  
              , MSG_LINE ) ;  
RETCODE      = PLIRETV() ;  
END SCLEND ;  
END;
```



---

## Chapter 4. SCLM Macros

SCLM supplies a set of macro instructions that you can use to define project definitions. This chapter describes those macro instructions, explaining the format of each. The macros described below are contained in ISP.SISPMACS, which is delivered with the product.

The macros appear in alphabetical order. For each macro, the chapter provides the command format, a description of the parameters you use, and an example. For additional information, refer to *ISPF Software Configuration and Library Manager (SCLM) Project Manager's and Developer's Guide*.

Table 3. Macros

Macro	Description	Page
FLMABEG	Define project name of the project definition	"FLMABEG Macro" on page 129
FLMAEND	Last macro in the project definition	"FLMAEND Macro" on page 130
FLMAGRP	Define a set of authorization codes	"FLMAGRP Macro" on page 130
FLMALLOC	Many uses	"FLMALLOC Macro" on page 130
FLMALTC	Specify control information	"FLMALTC Macro" on page 148
FLMATVER	Enable audit and version utility	"FLMATVER Macro" on page 152
FLMCNTRL	Specify project specific control options	"FLMCNTRL Macro" on page 155
FLMCPYLB	Identify data set name to be allocated	"FLMCPYLB Macro" on page 173
FLMGROUP	Define groups in the project definition	"FLMGROUP Macro" on page 174
FLMINCLS	Associate include sets with types in the project hierarchy	"FLMINCLS Macro" on page 176
FLMLANGL	Define a language to SCLM	"FLMLANGL Macro" on page 180
FLMLRBLD	Cause certain members to be rebuilt when promoted into particular groups.	"FLMLRBLD Macro" on page 182
FLMSYSLB	Define system macro or include data sets	"FLMSYSLB Macro" on page 183
FLMTCOND	Run or skip build translators	"FLMTCOND Macro" on page 185
FLMTOPTS	Vary options passed to a build translator	"FLMTOPTS Macro" on page 189
FLMTRNSL	Similar to JCL EXEC statements	"FLMTRNSL Macro" on page 190
FLMTYPE	Define types in the project definition	"FLMTYPE Macro" on page 196

This chapter uses the following notation conventions to describe the format of the SCLM macros:

- Uppercase** Uppercase commands or parameters must be spelled out as shown.
- Lowercase** Lowercase parameters are variables; substitute your own values.
- Underscore** Underscored parameters are the system default.
- Brackets ( [ ] )** Parameters in brackets are optional.
- Braces ( { } )** Braces show two or more parameters from which you must select one.
- OR ( | )** The OR ( | ) symbol separates two or more values (inside braces) from which you must select one.

The example below shows the macro format for the SAMPLE macro:

```
SAMPLE PARM1=parm1_input
        ,PARM1A={XXX|YYY|ZZZ}
        [,PARM2=parm2_input]
        [,PARM2A=Y|N]
```

In the sample macro:

- PARM1 is a required parameter for which a user-specified value is substituted. There is no default value.
- PARM1A is a required parameter that must have the value XXX, YYY, or ZZZ. There is no default value.
- PARM2 is an optional parameter for which a user-specified value is substituted. There is no default value.
- PARM2A is an optional parameter that must have the value Y or N. The default is Y.

---

## Notes on Using the SCLM Macros

Because these are S/370 Assembler macros, all rules pertaining to Assembler macros apply. In addition, the following SCLM guidelines apply to the use of SCLM macros:

- Assembler does not support blanks in macro parameters; if a blank is used in a parameter that is delimited by parentheses, everything on the line after the blank will be ignored. If you use single quotes to delimit parameters, be careful when continuing to a new line. All blanks between the first single quote and the continuation character are considered part of the parameter. This can result in exceeding the maximum parameter length. If you need more than 71 characters for a line of code, you must put a continuation character in column 72 and begin the remaining lines in column 16.
- If any commas are omitted from the parameter list for any of the macros, the project definition might assemble correctly, but SCLM might use different defaults than expected, resulting in errors. All parameters except the last must be followed by a comma.
- If an optional parameter is specified without a value or the parameter is not specified, the default value is used; for example, PARM2A= or not specifying PARM2A on the macro statement causes PARM2A to default to Y. If the parameter does not have a default value then no value (null) is specified for the parameter.
- SCLM handles invalid values for required and optional parameters differently. If you specify an invalid value for a required parameter, SCLM might issue an

error message and the project definition assembly ends with a return code of 8. If you specify an invalid value for an optional parameter, SCLM issues a warning message, uses the default value for the parameter, and returns a return code of 4. Limited verification of the parameters is done during the assembly of the project definition. In many cases, the error does not occur until run time. An MNOTE is added to the assembly listing to indicate the invalid parameter specifications.

- SCLM performs validation of data set names when the data sets are opened. SCLM performs validation of VSAM versioning data set names when they are required for use by SCLM.
- The messages you receive when the project definition is assembled are issued from the SCLM macros and the assembler. SCLM messages are identified as MNOTES. For more information on MNOTES, refer to *ISPF Messages and Codes*. For explanations of other messages, refer to the *Assembler H V2 Programming Guide*.

The SCLM MNOTES appear in one of two places within the listing:

- Directly after the SCLM macro statement that contains incorrect parameter values
  - Near the end of the listing where SCLM cross-checks the values of the various SCLM macro statements. Assembler error messages usually occur inline where the syntax error was made.
- Some SCLM macros accept SCLM variables as input parameters. Valid variables are indicated in the parameter descriptions for the particular macros to which they apply. If the description of a parameter does not list valid variables, no variables can be used for that parameter. For more information on any variable mentioned in the descriptions, see “Chapter 6. SCLM Variables and Metavariables” on page 269.

You can find more information on the use of macros in the *Assembler H V2 Application Programming Language Reference* or *OS/VS - DOS/VSE - VM/370 Assembler Language*.

---

## FLMABEG Macro

Use this macro to define the project name of the project definition. It is required for the project definition and must appear before the other SCLM macros in the project definition.

### Macro Format

name FLMABEG

### Parameters

**name**

An 8-character project name. For alternate project definitions, use the “main” project definition name; this is the high-level qualifier of the project definition LOAD data set.

### Example

PROJ1 is the name of the project being specified by this project definition.

```
PROJ1 FLMABEG
```

### FLMAEND Macro

Use this macro as the last macro in the project definition. All SCLM macros you use to define the project definition must appear between the FLMABEG and FLMAEND macros. It is required and must be the last macro in the project definition.

#### Macro Format

FLMAEND

#### Parameters

This macro has no parameters.

---

### FLMAGRP Macro

Use this macro to define a set of authorization codes. You can then specify the authorization group name in the **AC** field on the FLMGROUP macro to assign the set of authorization codes to that group name.

#### Macro Format

name FLMAGRP AC=(code1,code2,...)

#### Parameters

##### name

An 8-character authorization group name containing no special characters or imbedded blanks.

##### AC=(code1,code2,...)

A list of authorization codes and authorization groups you can assign to the authorization group name. If *code#* is an authorization group, then you must have previously defined it with the FLMAGRP macro. Each authorization code or group can be up to 8 characters and cannot contain commas. The maximum number of characters allowed is 255, including commas and the delimiting parentheses.

#### Example

Authorization group SET1 contains the authorization codes R3M0, R3M1, and R3M2. Authorization group SET2 contains two authorization codes, R1M0 and R2M0, and one previously defined authorization group, SET1, for a total of five authorization codes (R1M0, R2M0, R3M0, R3M1, and R3M2).

```
SET1 FLMAGRP AC=(R3M0,R3M1,R3M2)
SET2 FLMAGRP AC=(R1M0,R2M0,SET1)
```

---

### FLMALLOC Macro

This macro provides the following capabilities to SCLM:

- Allocate temporary or permanent data sets that are used by translators  
FLMALLOC provides a limited equivalency to JCL DD or TSO ALLOCATE statements in your procedure libraries. The FLMALLOC parameters that provide this capability are BLKSIZE, CATLG, DDNAME, DIRBLKS, DISP, DSNTYPE, LRECL, RECFM, RECNUM, MEMBER, DINIT, MALLOC, and ALLCDEL. DINIT, MALLOC, and ALLCDEL indicate how the data set is to be dispositioned.

When allocating permanent data sets, use IOTYPE=A or I. When allocating temporary data sets, use IOTYPE=O, P, S, or W.

IOTYPEs A and I are used to associate a ddname with data sets that already exist.

IOTYPE=S is used for input data from an SCLM-controlled library. The member in this library is copied into a temporary data set for use by the translator.

IOTYPE=O is used for output data to be stored in an SCLM-controlled data set. A temporary sequential data set is allocated for use by the translator and the output produced by the translator is copied into the member in the SCLM-controlled data set.

IOTYPE=P is used for output data to be stored in an SCLM-controlled library. A temporary partitioned data set is allocated for use by the translator and members produced by the translator are copied into an SCLM controlled data set.

In general, if the translator writes to a sequential data set, use IOTYPE=O; if the translator writes to a member of a partitioned data set, use IOTYPE=P.

IOTYPE=W is used to allocate temporary data sets for use by the translator. These data sets are discarded at the completion of a BUILD or PROMOTE.

SCLM creates temporary data sets for the translators to use rather than allocating directly to the hierarchy data sets. This protects the integrity of the project hierarchy data when the translator is producing output that will be stored in the hierarchy. The output is copied from the temporary datasets to the project hierarchy after all the translators have been invoked. If multiple translators are invoked, DDNAMEs for the temporary outputs must be unique for each translator or only the outputs from the last translator will be copied.

All of the allocations for FLMALLOC macros that follow the FLMTRNSL macro are performed just before the translator is invoked. The exception to this rule applies when MALLOC=Y; see the description of MALLOC for details. The ordering of FLMALLOC macros in relation to FLMTRNSL macros is similar to the ordering of DD statements in relation to EXEC statements in JCL.

Temporary data sets that were created by SCLM are deleted when all of the build translators have completed processing for the member being built.

- Control the contents of the ddname substitution list

The ddname substitution list is passed as a parameter to a translator that has PORORDER=2 or 3. If PORORDER=0 or 1, SCLM does not generate a ddname substitution list. Not all translators accept ddname substitution lists. If a translator does accept a ddname substitution list, the ddnames in the list are used to override the default ddnames used by the translator.

In addition to ddnames, sometimes the ddname substitution list specifies the name of a member to be created. This is true for the linkage editor used by SCLM. The linkage editor accepts the name of the member to be created as a parameter in the ddname substitution list. Valid FLMALLOC parameters for this capability are KEYREF and IOTYPE. Use IOTYPE=L to add a member name to its ddname substitution list.

Ddname substitution lists are generated through the use of FLMALLOC statements. Each FLMALLOC statement adds a ddname to the list. The order in which the ddnames appear in the list is defined by the order of the FLMALLOC statements.

For general information about ddname substitution lists, refer to “Invoking Utility Programs from Application Programs” in the *DFSMS Utilities* manual . Refer to the manuals for the specific translator being invoked for details on the substitution list contents expected. For IBM supplied compilers, this information

## FLMALLOC Macro

is located in the compiler's Programmers Guide manual under "Invoking Compiler from Application Programs" or "Dynamic Invocation of Compiler".

- Identify hierarchy data to be used or created by a translator  
FLMALLOC can be used to identify information in the hierarchy that is either the input to a translator or the destination of the output from a translator.  
The FLMALLOC parameters that are valid for this purpose are MALLOC, NOSAVRC, DFLTTYP, KEYREF, and LANG. The IOTYPE identifies whether the allocation is for input to or output from a translator. To identify the members of the hierarchy to use as input to a translator, use IOTYPE=S or A. To identify the temporary output data sets that SCLM should store in the hierarchy, use IOTYPE=O or P. If you want the information to be read from or saved in the project hierarchy, you must specify the KEYREF parameter, except for IOTYPE=S, which defaults to SINC.
- Identify the translator data sets to be copied to a listing data set  
Use the PRINT parameter to copy the contents of a temporary data set to the listing data set. A listing data set is a sequential data set that contains any list information returned from a translator. Listings can be created by the SCLM build, promote, parse, migrate, or save services and by SCLM Edit. See the PRINT parameter description for more information.
- Use the output of one translator as input to another translator  
This capability is used when one translator creates information that is required by another translator. This is only possible when multiple translators are defined for a language definition for the same function (FUNCTN=) value.  
Use IOTYPE=U to indicate that the output from a previously called translator is to be used. See "IOTYPE=U" on page 140 for more information.

## Macro Format

```
FLMALLOC IOTYPE={A|I|L|N|O|P|S|U|W}  
  
[,BLKSIZE=block_size]  
  
[,CATLG=N|Y]  
  
[,DDNAME=ddname]  
  
[,DINIT=N|Y]  
  
[,DIRBLKS=directory_blocks]  
  
[,DISP=OLD|SHR|MOD|NEW]  
  
[,DFLTMEM=default_member]  
  
[,DFLTTYP=default_type]  
  
[,DSNTYPE=PDS|Library]  
  
[,KEYREF=keyword_reference]  
  
[,LANG=language]  
  
[,MALLOC=N|Y]  
  
[,ALLCDEL=N|Y]  
  
[,LRECL=record_length]  
  
[,MEMBER=member_name]
```

```

[,NOSAVRC=no_save_rc]
[,PRINT=N|Y|I]
[,RECFM=record_format]
[,RECNUM=number_of_records]
[,VIO={Y|N}]
[,INCLS=include_set_name]

```

## Parameters

**IOTYPE={A|I|L|N|O|P|S|U|W}**

Specifies the type of data sets to be allocated and how these data sets can be used. FLMALLOC has different capabilities based on the IOTYPE assigned to it. Therefore, through the use of IOTYPES, FLMALLOC is like nine different macros.

Figure 4 on page 134 shows the language definition, FLM01ASM, which is delivered in the partitioned data set ISPSISPSAMP. This sample is available as part of the ISPF product. Refer to this sample as necessary to understand the different IOTYPES. Of course, not all IOTYPES are used by any one language definition, but this will provide some aid in understanding most IOTYPES.

## FLMALLOC Macro

```

*****
*           OS/V S ASSEMBLER LANGUAGE DEFINITION FOR SCLM
*
* POINT THE FLMSYSLB MACRO(S) AT ALL 'STATIC' COPY DATA SETS
* ADD FLMCPYLB MACROS FOR EACH FLMSYSLB, TO THE 'SYSLIB' FLMALLOC MACRO
* 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.
*****
ASM      FLMSYSLB SYS1.MACLIB
*
          FLMLANGL    LANG=ASM,VERSION=ASMV1.0
*
* PARSER TRANSLATOR
*
          FLMTRNSL   CALLNAM='SCLM ASM PARSE',
                   FUNCTN=PARSE,
                   COMPILE=FLMLPGEN,
                   PORDER=1,
                   OPTIONS=(SOURCEDD=SOURCE,
                   PARSEMEM=@@FLMMBR,
                   STATINFO=@@FLMSTP,
                   LISTINFO=@@FLMLIS,
                   LISTSIZE=@@FLMSIZ,
                   LANG=A)          *** THIS IS ASSEMBLER ONLY ***
*
          (* SOURCE      *)
          FLMALLOC   IOTYPE=A,DDNAME=SOURCE
          FLMCPYLB  @@FLMDSN(@@FLMMBR)
*
* BUILD TRANSLATOR(S)
*
*
*
          --ASSEMBLER INTERFACE--
          FLMTRNSL   CALLNAM='ASSEMBLER',
                   FUNCTN=BUILD,
                   COMPILE=IFOX00,
                   VERSION=1.0,
                   GOODRC=0,
                   PORDER=1,
                   OPTIONS=(XREF(SHORT),LINECOUNT(75),OBJECT,RENT)
*
* DDNAME ALLOCATIONS
*
          FLMALLOC   IOTYPE=S,DDNAME=SYSIN,KEYREF=SINC,RECNUM=5000
          FLMALLOC   IOTYPE=W,DDNAME=SYSUT1,RECNUM=17500
          FLMALLOC   IOTYPE=W,DDNAME=SYSUT2,RECNUM=15000
          FLMALLOC   IOTYPE=W,DDNAME=SYSUT3,RECNUM=15000
          FLMALLOC   IOTYPE=O,DDNAME=SYSGO,KEYREF=OBJ,RECNUM=7500,DFLTYP=OBJ
          FLMALLOC   IOTYPE=A,DDNAME=SYSTEM
          FLMCPYLB   NULLFILE
          FLMALLOC   IOTYPE=A,DDNAME=SYSPUNCH
          FLMCPYLB   NULLFILE
          FLMALLOC   IOTYPE=I,DDNAME=SYSLIB,KEYREF=SINC
* ADD ONE FLMCPYLB FOR EACH FLMSYSLB
          FLMCPYLB   SYS1.MACLIB
          FLMALLOC   IOTYPE=O,DDNAME=SYSRINT,KEYREF=LIST,PRINT=Y,
                   DFLTYP=SOURCLST,RECNUM=20000
*
* 5665-402 (C) COPYRIGHT IBM CORP 1980, 1989

```

Figure 4. Sample language definition for Assembler

For the purpose of explanation, assume that any source modules built by sample architecture definitions given in the following descriptions have been saved with the preceding language definition.

**IOTYPE=A** Allocate a permanent data set or set of permanent data sets for either input or output. You need the FLMCPYLB macro to identify the data sets. There is an MVS limitation to the number of data sets that can be allocated to the ddname; the maximum is 123 data sets. The data sets allocated using this IOTYPE can be either partitioned or sequential. The default disposition is SHR. The DISP parameter can be used to override the default when a single data set is to be allocated. For example, you might use the override when you want to allocate a data set to be used for output with a disposition of OLD. If more than one data set is allocated, the DISP parameter must be SHR.

**Example:**

In Figure 4 on page 134, IOTYPE=A is used to allocate the SOURCE DDNAME. This identifies the source data set that will be used by the 'SCLM ASM PARSE' translator. If you use SCLM Edit to save member FLM01MD1 in type SOURCE and group DEV1 of project PROJ1, the FLMCPYLB statement identifies 'PROJ1.DEV1.SOURCE(FLM01MD1)' as the member to allocate as input to the parser.

**IOTYPE=I** Allocate libraries in the hierarchy for an include set. The INCLS parameter indicates the name of an include set as specified on an FLMINCLS macro. If no INCLS parameter is specified, the default include set is used.

A value should be specified for the KEYREF parameter or it will default to SINC and a warning message will be issued. The data sets allocated depend on the value of the KEYREF parameters:

- For KEYREF=SREF, the hierarchy for the SREF type is allocated.
- For KEYREF=CREF, the hierarchy for the CREC type is allocated.
- For KEYREF=SINC, the INCLS parameter indicates that the types allocated are listed in the FLMINCLS macro for the include set. The FLMSYSLB data sets are allocated if ALCSYSLB=Y is specified on the FLMLANGL macro for the language, followed by any data sets specified on FLMCPYLB macros.

SCLM allocates all of the data sets for the types associated with the include set within the current view of the hierarchy. The starting group for the hierarchical view is defined by the group used as input to the function, rather than the group where the referenced member was found. The hierarchies for each type are allocated in the order specified on the FLMINCLS macro.

This allocation is typically used to resolve include dependencies when performing a compilation. FLMCPYLBs that follow this allocation should not reference SCLM-controlled data sets.

At least one data set must exist in the hierarchy for the types referenced.

**Example:**

## FLMALLOC Macro

In Figure 4 on page 134, IOTYPE=I with KEYREF=SINC is used to allocate the SYSLIB DDNAME. Because no INCLS parameter is specified for the IOTYPE=I, the default include set is used. In addition, because no FLMINCLS macro is specified for the default include set in this language definition, an FLMINCLS macro is generated with TYPES=(@@FLMTYP,@@FLMETP). If the example project hierarchy has been set up according to the steps identified in *ISPF Software Configuration and Library Manager (SCLM) Project Manager's and Developer's Guide* member FLM01CMD for 'PROJ1.RELEASE.ARCHDEF' contains the following statements:

```
*
*   Object Module 1
*
OBJ   FLM01MD1 OBJ
LIST  FLM01MD1 SOURCLST
SINC  FLM01MD1 SOURCE
PARM  NOXREF,LC(75)
```

If a build was performed on this member at the DEV1 group, the FLMALLOC macro would indicate that a hierarchy should be allocated starting at the DEV1 group for the type indicated by the SINC card. In this case, 'PROJ1.DEV1.SOURCE', 'PROJ1.TEST.SOURCE', 'PROJ1.RELEASE.SOURCE', and 'SYS1.MACLIB' would be allocated.

If you were to look at the project definition for PROJ1, you would see the following macro that defines the SOURCE type:

```
SOURCE  FLMTYPE
```

Notice that there is no extend type defined. If, however, this type had been defined as follows:

```
SOURCE  FLMTYPE EXTEND=SOURCE2
```

then building this member at the DEV1 group would have resulted in an allocation of the following: 'PROJ1.DEV1.SOURCE', 'PROJ1.TEST.SOURCE', 'PROJ1.RELEASE.SOURCE', 'PROJ1.DEV1.SOURCE2', 'PROJ1.TEST.SOURCE2', 'PROJ1.RELEASE.SOURCE2', and 'SYS1.MACLIB', in that order.

If the extended type SOURCE2 was defined as shown in the preceding macro, and a build was performed at the TEST group, the following would be allocated: 'PROJ1.TEST.SOURCE', 'PROJ1.RELEASE.SOURCE', 'PROJ1.TEST.SOURCE2', 'PROJ1.RELEASE.SOURCE2', and 'SYS1.MACLIB', in that order.

In this example, the default values have been used for the include set. If the FLMALLOC macro for IOTYPE=I had been written as follows, the include set of SYSLIB would have been used:

```
FLMALLOC  IOTYPE=I,DDNAME=SYSLIB,KEYREF=SINC,INCLS=SYSLIB
* ADD ONE FL MCPYLB FOR EACH FLMSYSLB
  FL MCPYLB  SYS1.MACLIB
```

In the previous example, the default values have been used for the include set and the FLMSYSLB data sets were not allocated. If the FLMLANGL macro had ALCSYSLB=Y and the FLMALLOC macro for IOTYPE=I had been written as follows, the include set of SYSLIB would have been used:

## FLMALLOC Macro

```
FLMALLOC IOTYPE=I,DDNAME=SYSLIB,KEYREF=SINC,INCLS=SYSLIB
* COPYLIB ALLOCATION OF FLMSYSLB DATA SETS IS DONE AUTOMATICALLY
```

The FLMSYSLB macro would need to specify the include set using the INCLS parameter.

```
ASM    FLMSYSLB SYS1.MACLIB,INCLS=SYSLIB
```

An FLMINCLS macro is required in the language definition to indicate the types to be included in the allocation. The following FLMINCLS macro first searches the MACROS type followed by the type and extended type.

```
* INDICATE THE TYPES TO SEARCH FOR THE SYSLIB INCLUDE-SET
SYSLIB    FLMINCLS TYPES=(MACROS,@@FLMTYP,@@FLMETP)
```

Using the preceding FLMINCLS macro, SCLM allocates data sets in the following order for the SYSLIB ddname when building at group DEV1:

1. 'PROJ1.DEV1.MACROS'
2. 'PROJ1.TEST.MACROS'
3. 'PROJ1.RELEASE.MACROS'
4. 'PROJ1.DEV1.SOURCE'
5. 'PROJ1.TEST.SOURCE'
6. 'PROJ1.RELEASE.SOURCE'
7. 'PROJ1.DEV1.SOURCE2'
8. 'PROJ1.TEST.SOURCE2'
9. 'PROJ1.RELEASE.SOURCE2'
10. 'SYS1.MACLIB'

### IOTYPE=L

Pass a member name in the ddname substitution list. See the PORDER parameter in "FLMTRNSL Macro" on page 190 for more information. The KEYREF parameter identifies the member name and type. This IOTYPE is commonly used to identify the load module name for the S/370 linkage editor.

**Note:** IOTYPE=L is only valid when the PORDER parameter in the FLMTRNSL macro is set to 2 or 3.

### Example:

Figure 4 on page 134 is not a linkage editor language definition; therefore, it does not contain an example of IOTYPE=L. However, FLM01370 in ISP.SISPSAMP, part of the sample project definition, contains an example of IOTYPE=L. If the example project hierarchy has been set up according to the steps identified in *ISPF Software Configuration and Library Manager (SCLM) Project Manager's and Developer's Guide* member FLM01LD1 for 'PROJ1.RELEASE.ARCHDEF' contains the following statements:

```
*
*   Load Module LMOD1
*
LOAD  FLM01LD1 LOAD
LMAP  FLM01LD1 LMAP
INCL  FLM01CMD ARCHDEF
PARAM MAP,NCAL,
PARAM LET
```

## FLMALLOC Macro

If a build was performed for this architecture definition, the FLMALLOC macro with IOTYPE=L and KEYREF=LOAD would pass "FLM01LD1" to the Linkage Editor.

**IOTYPE=N** Skip over a field during ddname substitution. This IOTYPE is valid only for PORDER=2 or 3. SCLM adds 8 bytes of hexadecimal zeros to the ddname substitution list.

**Example:**

This IOTYPE is not used by Figure 4 on page 134, but if a translator accepted a ddname substitution list, using this IOTYPE on the FLMALLOC macro would result in 8 hexadecimal zeros being placed in the ddname substitution list.

**IOTYPE=O** Allocate a sequential temporary data set that will contain output from a translator that is to be saved in the project hierarchy. A KEYREF parameter must be used to identify the output member name and type. Valid KEYREF values are OBJ, COMP, LIST, LOAD, LMAP, and OUTx.

**Note:** If the outputs of a translator are empty files then SCLM will copy the translator outputs into the hierarchy as empty members and create accounting records for these members.

**Example:**

In Figure 4 on page 134, IOTYPE=O is used to allocate the SYSPRINT DDNAME. This is a temporary data set into which the translator will write the Assembler listing. Refer to the example for IOTYPE=I for an illustration of what is contained in architecture definition member FLM01CMD. If this member were built at DEV1, the build listing would be copied into the hierarchy into the member and type specified by the LIST card, FLM01MD1 SOURCLST.

This example is building an architecture definition, so DFLTTYP will be ignored or overridden by the LIST card. If only the source were being built, the listing would go into the type specified by DFLTTYP.

**IOTYPE=P** Allocate a temporary partitioned data set that will contain output from a translator that is to be saved in the project hierarchy. The dsntype parameter is used to indicate whether the temporary data set should be allocated as PDS or PDSE.

If the output from a translator is to be saved in the project hierarchy, then a KEYREF parameter must be used to identify the target member into which the translator output will be copied. Specify any output KEYREF value, such as KEYREF=LOAD or OUTx. If the output from a translator is not to be saved in the project hierarchy, do not specify a KEYREF parameter; this is essentially like using IOTYPE=W except that a partitioned data set is allocated for use by the translator instead of a sequential data set.

If the build is to occur on a workstation, use IOTYPE=P and DFLTMEM=\* to take advantage of workstation build caching. IOTYPE=P preserves the workstation file's date and time information as it is copied to the host. If the output is needed as input for

another build step, the date and time at the host member is compared the date and time of the corresponding workstation file. If they match, the file is considered to be the same, and the file is not transferred.

IOTYPE=P would be used when a translator requires a partitioned data set. If a translator accepted a sequential data set, IOTYPE=O would be used.

SCLM determines the names of the members to be copied into the project hierarchy from the architecture definition being built or from the DFLTMEM parameter on the FLMALLOC macro. If an architecture definition member is being built, the name specified in that member is used. If a source member is being built directly or as a result of an INCLD architecture statement, the DFLTMEM parameter on the FLMALLOC macro is used.

If an asterisk is specified for the output member name in the architecture definition, or no DFLTMEM parameter is specified, then all members in the temporary data set are copied into the project hierarchy. Otherwise, only the member that matches the name on the architecture statement or DFLTMEM parameter is copied into the project hierarchy.

**Example:**

Figure 4 on page 134 does not contain an example of IOTYPE=P. However, if the Assembler, IFOX00, had required a partitioned data set for the object module instead of a sequential data set, then the FLMALLOC for the SYSGO DDNAME would have used IOTYPE=P instead of IOTYPE=O.

**IOTYPE=S**

Allocate a temporary sequential data set and create the input stream for the translator by concatenating the contents of all the members that are SINCD as well as any text specified via CMD cards. Concatenation will occur in the order specified by the architecture definition. Use the KEYREF parameter to identify the members from the project hierarchy that will be used to create the input stream.

When the following criteria are met, SCLM allocates the PDS member directly from the SCLM-controlled library, rather than copying the member first to a sequential data set. The criteria are:

- there is only one input
- the input is from a SINC statement
- the KEYREF on the FLMALLOC statement is SINC
- you are not doing input list processing.

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

**Example:**

In Figure 4 on page 134, IOTYPE=S is used to allocate the data set that will contain the input stream for the translator SYSIN. Refer to the example for IOTYPE=I for an illustration of what is contained in architecture definition member FLM01CMD. If this member were built at DEV1, the SYSIN data set would contain a copy of member FLM01MD1, type SOURCE. If more than one SINC card

## FLMALLOC Macro

had been specified, then the source referenced by subsequent SINC cards would have been appended to the end of SYSIN in the order specified in the architecture definition.

### IOTYPE=U

Any preallocated ddname that matches the DDNAME parameter value will be used. There will be no new ddname allocation. This is typically used for referring back to a preallocated ddname from a previous FLMALLOC following a previous FLMTRNSL in the same language definition. In this situation the DDNAME parameter values need to be the same.

Ddname substitution lists are useful in situations in which more than one translator is defined for a language and one translator needs to use the output from a previous translator. This latter translator would have an FLMALLOC statement with IOTYPE=U and the same DDNAME parameter value as the previous FLMALLOC for a previous FLMTRNSL in the same language definition. In order to use ddname substitution lists the translator must be programmed to handle the ddname substitution list and the FLMTRNSL must have a PORDER value of 2 or 3 to construct and pass the list to the translator.

Translators that are programmed to use ddname substitution lists include some compilers, linkage editors, and utilities. These translators will use the DDNAME parameter value for a data set. If the DDNAME parameter is not specified the system will generate a ddname for use in the ddname substitution list.

For PORDER values of 0 or 1 SCLM does nothing. There are no additional file allocations. At execution time the translator will use whatever data set has been allocated to the ddname specified by the translator program.

### Example:

Figure 4 on page 134 does not use IOTYPE=U. The sample language definition for the assembler language only calls one build translator. However, if this language definition had called a preprocessor and had PORDER=2 or 3, as shown in Figure 5 on page 141, the assembler compiler, IFOX00, would want to use the output from the preprocessor, IFPRE0. It would not be necessary for IFOX00 to create a data set that would contain the input stream because this has been prepared by IFPRE0.

```

*
* BUILD TRANSLATOR(S)
*
*
* --CREATE THE INPUT STREAM FOR THE ASSEMBLER COMPILER--
*
* --ASSEMBLER PREPROCESSOR--
*   FLMTRNSL  CALLNAM='ASM PREPROCESSOR',          C
*             FUNCTN=BUILD,                        C
*             COMPILE=IFPRE0,                      C
*             PORDER=3,                            C
*             OPTIONS=(GROUP=@@FLMGRP,             C
*             TYPE=@@FLMTYP,                       C
*             MEMBER=@@FLMMBR)
*
* DDNAME ALLOCATIONS
*
*   FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,          C
*             RECNUM=9000,DDNAME=SYSIN
* --CALL THE ASSEMBLER COMPILER TO PROCESS INPUT--
*
* --ASSEMBLER INTERFACE--
*   FLMTRNSL  CALLNAM='ASSEMBLER',                C
*             FUNCTN=BUILD,                        C
*             COMPILE=IFOX00,                      C
*             VERSION=1.0,                         C
*             GOODRC=0,                            C
*             PORDER=3,                            C
*             OPTIONS=(XREF(SHORT),LINECOUNT(75),OBJECT,RENT)
*
* DDNAME ALLOCATIONS
*
*   FLMALLOC  IOTYPE=U,DDNAME=SYSIN
*   FLMALLOC  IOTYPE=W,DDNAME=SYSUT1,RECNUM=17500
*   FLMALLOC  IOTYPE=W,DDNAME=SYSUT2,RECNUM=15000
*   FLMALLOC  IOTYPE=W,DDNAME=SYSUT3,RECNUM=15000
*   FLMALLOC  IOTYPE=O,DDNAME=SYSGO,KEYREF=OBJ,RECNUM=7500,DFLTYP=OBJ
*   FLMALLOC  IOTYPE=A,DDNAME=SYSTEM
*   FLMCPYLB  NULLFILE
*   FLMALLOC  IOTYPE=A,DDNAME=SYSPUNCH
*   FLMCPYLB  NULLFILE
*   FLMALLOC  IOTYPE=I,DDNAME=SYSLIB,KEYREF=SINC
* * ADD ONE FLMCPYLB FOR EACH FLMSYSLB
*   FLMCPYLB  SYS1.MACLIB
*   FLMALLOC  IOTYPE=O,DDNAME=SYSPRINT,KEYREF=LIST,PRINT=Y,
*             DFLTYP=SOURCLST,RECNUM=20000
*

```

Figure 5. Sample Language Definition that Calls a Preprocessor

**IOTYPE=W** Allocate a temporary sequential data set for translator use. SCLM uses the RECFM, LRECL, and RECNUM parameters for allocation of this data set. If they are not specified, SCLM uses the defaults.

**Example:**

Figure 4 on page 134 uses IOTYPE=W to allocate SYSUT1. When the Assembler, IFOX00, is invoked, a sequential data set is created and allocated to DDNAME SYSUT1. This data set is used internally by the assembler and it is not necessary to store it in the project hierarchy. This language definition does not print the contents of this data set to the build listing data set because the PRINT keyword was not specified and defaults to N.

## FLMALLOC Macro

When all build translators for this language have completed processing, SYSUT1 will be deleted. In the preceding example, when the assembler has completed and returned control to build, build will deallocate the data set associated with SYSUT1.

The position of the FLMALLOC macros is very important because SCLM can pass ddnames directly to the translator; see the **PORDER** field description. SCLM passes ddnames to the translator in the order of the FLMALLOC macros.

SCLM deallocates temporary data sets after all translators for a particular member and a particular function (for example, FUNCTN=BUILD, COPY, and PURGE specified in the FLMTRNSL macro) for a particular language have completed processing.

To use the output from one translator step as input to another translator step, add (or modify) an FLMALLOC macro for the second translator step with IOTYPE=U and DDNAME=*ddname* allocated for the first translator step. Care should be taken when adding FLMALLOC macros for the second translator step. Depending on the PORDER that is specified in the FLMTRNSL macro, it may be necessary to put the new FLMALLOC macro in a particular position in the list of FLMALLOC macros. Refer to the documentation for the particular compiler or translator you are calling to determine whether or not it accepts ddname substitution lists and, if so, what order it expects the parameters to be passed.

Table 4 indicates the valid IOTYPES for each function. Note that all IOTYPES are valid for a build, and that IOTYPES A, U, and W are valid for all functions.

Table 4. Valid IOTYPES for Each Function

IOTYPE	Build	Copy	Parse	Purge	Verify.
A	X	X	X	X	X
I	X				
L	X				
N	X				
O	X				
P	X				
S	X				
U	X	X	X	X	X
W	X	X	X	X	X

### **,BLKSIZE=***block\_size*

Block size of the data set. This parameter is valid for IOTYPE=W, O, P, and S. If this parameter is not specified or is specified as 0 (zero), then the block size used is the largest integral multiple of the LRECL values that is less than or equal to 3120. It is recommended that this value match the block size of the target data set for IOTYPE=P and RECFM=U. This parameter is ignored for IOTYPE=A, I, L, N, and U.

The IBM linkage editor requires that the DCBS option parameter be passed in order for the SYSLMOD block size to be used in creating load modules. If the DCBS option is not specified, the linkage editor creates load modules using the maximum record size for the device type. Use the OPTIONS= parameter on the FLMTRNSL macro to pass the DCBS option. Failure to do so can result in message FLM44507.

**,CATLG=N|Y**

Indicates whether a data set is to be cataloged. Valid for IOTYPE=W, O, P, and S. SCLM temporarily allocates cataloged data sets with a predefined high-level qualifier, the TSO-prefix. The data set is deleted after all translators complete their functions. The default is N.

**,DDNAME=ddname**

The ddname to be used for this allocation. If you do not specify a ddname for the allocation, SCLM generates one for you. If the PORDER parameter in FLMTRNSL has the value of 0 or 1, a nonblank value is required for DDNAME.

A special case occurs when MALLOC=Y is specified. Because MALLOC=Y implies more than one allocation, you must allow SCLM to generate ddnames for these allocations. If PORDER=2 or 3, SCLM generates a ddname if the parameter is omitted. This parameter is not used for IOTYPE=L or IOTYPE=N.

Do not reuse the same ddname in multiple-step language definitions unless you intend to pass data from one step to the next using IOTYPE=U. If the same DDNAME is used for multiple translators, only the outputs from the last translator will be copied to the hierarchy.

**,DINIT=N|Y**

Indicates whether SCLM should create a member in a temporary data set allocated with IOTYPE=P. DINIT is ignored for all IOTYPES except P. If DINIT=Y, SCLM initializes the member with a single record containing the string "DUMMY FILE" beginning in column 1. The member created will have the same name as the build map that is created if the translator is successful. If the MEMBER parameter is specified, its value will be used to determine the name of the member to initialize. If the MEMBER parameter is not specified, the member initialized will be the member to be saved in the hierarchy. If the member will not be saved in the hierarchy, the member initialized will have the same name as the source or architecture definition controlling the build.

**,DIRBLKS=directory\_blocks**

The number of directory blocks allocated to the data set if the data set is partitioned (IOTYPE=P). For IOTYPE=P, the default is 1 and for all other IOTYPES, the default is 0. SCLM will ignore nonzero values for all IOTYPES except IOTYPE=P.

**,DISP= OLD|SHR|MOD|NEW**

Optional parameter used to identify the disposition for the allocation on a DD card in JCL. Valid values are OLD, SHR, MOD, NEW. If not specified, the disposition defaults to an appropriate value for the IOTYPE parameter, as described in Table 5.

Table 5. Valid DISP values for IOTYPE values

IOTYPE	Default	Valid Values	Condition.
A	SHR	SHR,MOD,OLD	With a single copylib
A	SHR	SHR	With multiple copylibs
A	SHR	SHR,OLD	With MALLOC=Y
I	SHR	SHR	
L	n/a	n/a	
N	n/a	n/a	
O	OLD	NEW,MOD	
O	SHR	SHR,OLD	With MALLOC=Y

## FLMALLOC Macro

Table 5. Valid DISP values for IOTYPE values (continued)

IOTYPE	Default	Valid Values	Condition.
P	NEW	NEW	
S	MOD	MOD	
U	n/a	n/a	
W	NEW	NEW,MOD	

The DISP parameter applies to the temporary data set created by SCLM instead of the controlled members in the hierarchy for IOTYPE O and P. Refer to the *TSO Extensions Version 2 Command Language Reference* for more information.

This parameter is ignored for IOTYPE L, N, and U.

### **,DFLTMEM=***default\_member*

Indicates the name for the output member. Use IOTYPE=O or P to allocate data sets that will be used for translator outputs. If this parameter is not used, the output member name for IOTYPE=O will be the same as the source member; for IOTYPE=P all output members will be copied using the translator-generated member names. SCLM ignores this field during a build if you use an architecture definition member to build the source member. If you are using an architecture definition member, define the translator outputs with an output keyword such as OBJ, OUTx, or LOAD. DFLTMEM is ignored unless:

1. KEYREF is specified with a valid output keyword.
2. DFLTTYP is specified.
3. IOTYPE is either O or P.

The name of the translator output can be based on the name of the source input by using an asterisk as a special match character. The asterisk is replaced by the name of the source member. If the substitution of the source name would result in a name longer than 8 characters, the source name is truncated to produce an 8-character name. For example, if the DFLTMEM parameter is \*FM, a source member of EX00G would cause the output to be stored in name EX00GFM.

### **,DFLTTYP=***default\_type*

Indicates the name of the SCLM type for translator outputs. Use IOTYPE=O or P to allocate data sets that will be used for translator outputs. The output member name is the same as the source member. SCLM ignores this field during a build if you use an architecture member to build the source member. If you are using an architecture member, define translator outputs with an output keyword such as OBJ, OUTx, or LOAD. DFLTTYP is ignored if no KEYREF is specified.

The type for the translator output can be based on the type of the source input by using an asterisk as a special match character. The asterisk is replaced by the type of the source member. If the substitution of the source type would result in a name longer than 8 characters, the source type is truncated to produce an 8-character result. If the DFLTTYP parameter is \*LST, a source type of SRC1 would cause the output to be stored in type SRC1LST. The type specified on this parameter, or the type generated if an asterisk is used, must be defined to the project definition with the FLMTYPE macro. No verification of this parameter is performed when the project definition is generated.

**,DSNTYPE=PDS|Library**

Determines whether a temporary partitioned data set (IOTYPE=P) is allocated as a PDS or PDSE. Use DSNTYPE=LIBRARY to have the data set allocated as a PDSE. If you specify DSNTYPE=LIBRARY and your system or project specifies that temporary data sets should be allocated to VIO, then add the CATLG=Y parameter to the FLMALLOC macro. This parameter is only valid for IOTYPE=P. The default value is PDS.

**,KEYREF=keyword\_reference**

Refers to a keyword in the build map or architecture definition. The member name and type (as denoted in the build map or architecture definition) associated with the keyword are used by other parameters in this macro:

- If IOTYPE=L, *keyword\_reference* identifies the member name the macro passes in the ddname substitution list for the translator.
- If IOTYPE=S, *keyword\_reference* identifies the input members for the translator. For LEC architecture members, the contents of the temporary data set will depend on the KEYREF specified. If KEYREF is specified as INCL, an include statement in a format used by the S/370 linkage editor will be generated for each object member or load module referenced. If KEYREF is specified as SINC, the contents of each object member will be copied into the temporary data set. S/370 linkage editor include statements are generated for each load module specified as input. This is true when KEYREF=SINC or KEYREF=INCL. Although it will take longer to process KEYREF=SINC, this can be used to handle object members having large block sizes or containing linkage edit control statements.
- If IOTYPE=I, *keyword\_reference* determines the type name of the hierarchy to allocate. The keywords that can be used with IOTYPE=I are SINC, SREF, and CREF.
- If IOTYPE=O or P, *keyword\_reference* identifies the location in the hierarchy for build to copy the output created by the translator if the translator is successful. The keywords that can be used with IOTYPE=O or P are COMP, LIST, LMAP, LOAD, OBJ, and, OUTx.

**,LANG=language**

Allows a build output to be assigned a different language than the build input. If this parameter is not specified, then build outputs are assigned the same language as inputs.

This parameter is not necessary when you can create in a single build all the build outputs you want.

Use this parameter when you want the build output of one language definition to be verified, built, copied, or purged in another language definition.

**,MALLOC=N|Y**

Use MALLOC=Y when the translator generates a sequential output data set that has a specific data set name and cannot be allocated to a ddname before the translator is invoked. This condition might occur if the translator performs its own allocations and always creates a data set with a specific name. Input list translators are required to generate output data sets that can be captured with this type of allocation. For input list processing, one allocation is performed for each member processed on the input list.

When you specify the FLMALLOC macro with MALLOC=Y, you must also specify an FLMCPYLB macro that identifies the name of the data set to be allocated.

## FLMALLOC Macro

An FLMALLOC with MALLOC=Y is ignored for all iotypes except O and A. If MALLOC=Y and IOTYPE is not an A and not an O, an error message is produced. The KEYREF parameter must be specified on the FLMALLOC for the allocation to occur. If MALLOC=Y is specified, the ddname parameter must be blank.

### **,ALLCDEL=N|Y**

Indicates that all data sets referenced by this FLMALLOC macro should be deleted when SCLM has finished processing them. For example, specify ALLCDEL=Y to indicate that the output listings from the Input List translator should be deleted after they are copied into the hierarchy. The ALLCDEL parameter is ignored unless MALLOC=Y is specified.

### **,LRECL=record\_length**

Logical record length of the data set (numeric). It is valid for IOTYPE=W, O, P, and S. The default is 80. It is recommended that this value match the LRECL of the target data sets for IOTYPE=O or P.

### **,MEMBER=member\_name**

Causes a ddname to be allocated to a member of a temporary partitioned data set created by SCLM. This parameter is valid only for IOTYPE=P.

The member name can be evaluated dynamically by specifying @@FLMONM or @@FLMMBR as the parameter value. If a KEYREF OUTx parameter is specified and the architecture definition has a matching OUTx statement, then SCLM uses the output member name in the architecture definition. If no OUTx architecture statement is specified, then SCLM uses the name of the member being built. This can be the name of an architecture definition or the name of a build input.

This parameter is not necessary for most translators. However, some translators must know the name of the output member.

### **,NOSAVRC=no\_save\_rc**

A return code value set by a translator that indicates whether or not SCLM is to store a translator output in this data set. This parameter is valid for IOTYPE=O and P. SCLM provides this feature to handle translators that, by design, have missing or static outputs. If it is decided that these outputs need not be saved for some situations, then the translators can be written to recognize these situations and return an appropriate return code. Through the use of this return code and the NOSAVRC parameter, SCLM will be able to determine when the output should be saved in the hierarchy and when it should not. This helps avoid unnecessary rebuilds of some build components. This parameter, if specified, must have a nonzero positive value; if not specified, the default is zero.

**Note:** An example is a translator that can differentiate 'comment only' changes from code changes and determine which outputs are not affected. A listing is updated but not OBJECT code. SCLM can use this information to avoid unnecessary work.

### **,PRINT=N|Y|I**

Indicates whether or not the contents of a sequential data set are to be copied to the SCLM listings data set (userid.BUILD.LISTxx). The contents will only be copied in the case of an error when the *error listings only* field is selected on the build panel. This parameter is only valid for data sets allocated with IOTYPE=W, S, or O. The valid values are:

**N** indicates the contents of the temporary sequential data set are *not* to be copied to the listings data set. This is the default setting.

- Y indicates that the contents of the temporary sequential data set are to be copied to the listings data set.
- I indicates that the contents of the temporary sequential data set are to be copied to the listings data set. SCLM will open and close the temporary data set before invoking the translator.

Data sets allocated with PRINT=Y must be opened by the translator. Otherwise, an ABEND can occur when SCLM attempts to copy the contents to the build listings data set. For data sets that will not be opened by the translator, use PRINT=I. As PRINT=I adds an open and close, build performance can be slightly degraded.

**,RECFM=record\_format**

Record format of the data set. It is valid for IOTYPE=W, O, P, and S. Valid values are F, FA, FM, FB, FBA, FBM, V, VA, VM, VB, VBA, VBM, and U; the default is FB (fixed blocks). It is recommended that this value match the RECFM of the target data sets for IOTYPE=O or P.

**,RECNUM=number\_of\_records**

Number of records to be allocated (numeric). It is valid for IOTYPE=W, O, P, and S. The default is 500.

This parameter is used in the calculation of the primary and secondary space allocations required for the temporary data set. Space allocations are in blocks and the number of blocks is determined by the number of records using the following formula:

$$(((\text{number\_of\_records} * ((3120 / \text{record\_length}) + 1)) + 1) / 16) + 1$$

**,VIO=Y|N**

Overrides the selection for use of VIO. Y causes the data set to always be allocated using VIO; N causes the allocation to never use VIO. The default is to determine use of VIO by comparing the RECNUM specification to the value for MAXVIO on the FLMCNTRL macro.

**Note:** The Automatic Class Selection (ACS) routines defined for a DFSMS installation can override the selection requested by SCLM. Contact your site's system programmer for information about how these will interact on your system.

**,INCLS=FLMINCLS\_name**

Refers to an FLMINCLS macro in the language definition that lists the types to be allocated. If the FLMLANGL macro for the language has ALCSYSLB=Y, the FLMSYSLB data sets for the include set will be allocated after the data sets from the project. This parameter is only valid for IOTYPE=I. If no INCLS= parameter is specified for IOTYPE=I, the default include set is used to determine the types for allocation.

**Defining a Software Component Using the FLMALLOC Macro**

You can specify a software component either with an architecture member or with the FLMALLOC macros you specified in the language definition. For example, the language definition for member xxxxxxxx in type SOURCE contains the following FLMALLOC macros:

```
FLMALLOC IOTYPE=S,KEYREF=SINC
FLMALLOC IOTYPE=O,KEYREF=LIST,DFLTTP=LISTING
FLMALLOC IOTYPE=O,KEYREF=OBJ,DFLTTP=OBJECT
```

Building the member is the same as building the following architecture definition:

## FLMALLOC Macro

```
SINC xxxxxxxx SOURCE
LIST xxxxxxxx LISTING
OBJ xxxxxxxx OBJECT
```

Always use the SINC keyword (on the KEYREF= parameter of the FLMALLOC macro) to identify the input member. If you need multiple SINC keywords, you must use an architecture member to specify the software component. Options to override the translator options (using the PARM and PARMx keywords) also require that you use an architecture member. You can also use the fields **DFLTCRF** and **DFLTSRF** on the FLMLANGL macro to identify the types to use in resolving source dependencies.

### Example 1

Two data sets are allocated: one to contain the input stream (IOTYPE=S), the other to contain the output from the translator (IOTYPE=O). The input stream is the member you specify on the SINC statement of an architecture member. The output is copied to the member specified with the LIST statement of an architecture member. The output is also copied to the listing data set for the SCLM function.

```
FLMALLOC IOTYPE=S,KEYREF=SINC,RECNUM=5000,LRECL=80,RECFM=FB

FLMALLOC IOTYPE=O,KEYREF=LIST,RECNUM=5000,LRECL=133,RECFM=VBA, X
PRINT=Y
```

### Example 2

The hierarchy for the type specified on the SINC statement of an architecture member is allocated. Two additional data sets are allocated after the hierarchy by the FLMCPYLB macro.

```
FLMALLOC IOTYPE=I,KEYREF=SINC
FLMCPYLB SYS1.LINKLIB
FLMCPYLB SYS1.MACLIB
```

### Example 3

The temporary partitioned data set (IOTYPE=P) that will contain the translator output to be saved into the project hierarchy will be allocated as a PDSE.

```
FLMALLOC IOTYPE=P,KEYREF=LOAD,RECFM=U,LRECL=0, X
BLKSIZE=6144,RECNUM=5000,DIRBLKS=200,DDNAME=SYSLMOD, X
DSNTYPE=LIBRARY
```

---

## FLMALTC Macro

With this macro, you can specify control information that is different from that specified by FLMCNTRL. You can specify different VSAM databases or flexible data set naming conventions to associate with a group.

When the ALTC parameter of the FLMGROUP macro matches the name of the FLMALTC macro, only the control information for the VSAM databases and data set naming conventions defined in the FLMALTC macro are used for that group.

The FLMALTC macro values override the ACCT, ACCT2, DSNAME, EXPACCT, VERS, VERS2, and VERPDS values from the FLMCNTRL macro. The FLMALTC macro does *not* use these values from the FLMCNTRL macro so you must specify all the parameters you want on the FLMALTC macro statement. Any values not available to the FLMALTC macro are taken from the FLMCNTRL macro.

Any number of FLMGROUP macros can reference a single FLMALTC macro. SCLM issues a warning if an FLMALTC macro is defined that is not referenced by any FLMGROUP macro.

## Macro Format

name FLMALTC

```

ACCT=primary_accounting_data_set
[,ACCT2=secondary_accounting_data_set]
[,DSNAME=dataset_name]
[,EXPACCT=export_account_data_set]
[,VERS=primary_audit_control_data_set]
[,VERS2=secondary_audit_control_data_set]
[,VERPDS=version_pds_name]

```

## Parameters

*name*

A unique 8-character name used to identify the control information defined by the FLMALTC macro. The name must be used in conjunction with the ALTC parameter of an FLMGROUP macro to indicate which set of information should be used for that group.

*ACCT=primary\_accounting\_data\_set*

The name of the primary accounting data set to be used by any group referencing this FLMALTC macro. The data set you specify must be the name of the VSAM cluster you want to use. No SCLM variables can be used for this parameter.

*,ACCT2=secondary\_accounting\_data\_set*

The name of the secondary accounting data set to be used by any group referencing this FLMALTC macro. Allocate this secondary VSAM data set following the same criteria as the primary accounting data set. Choose a unique name for this data set. It should reside on a different volume than the primary one. If a severe problem occurs with the primary data set (for example, a head crash on that disk), you can use this backup data set to restore the primary data set. The default is no secondary accounting data set.

Because additional accounting updates take place if you use this option, the updates will degrade performance. No SCLM variables can be used for this parameter.

*,DSNAME=dataset\_name*

This parameter lets you specify the data set naming conventions for the partitioned data sets controlled by SCLM. The naming convention is specified as a pattern that can include a subset of the SCLM variables.

The only SCLM variables that can be used in the DSNAME parameter of FLMALTC are:

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

## FLMALTC Macro

The value specified in this parameter is used to resolve the SCLM variable @@FLMDSN. If this parameter is not specified, the data set name pattern defaults to @@FLMPRJ.@@FLMGRP.@@FLMTYP. You can enter up to 44 characters for this parameter, including the SCLM variables and the periods.

If a data set name is specified, it must include the SCLM variable @@FLMTYP. It is also recommended that the variable @@FLMGRP be used in the data set name pattern. This helps prevent data from one group overwriting data in another group.

**Note: SCLM does not enforce or guarantee the uniqueness of partitioned data set names.**

The variables can appear in any location within the DSNAME parameter. Any user-specified qualifiers can also be used. The preceding SCLM variables will be substituted with values that range from 1 to 8 characters. When determining the length of the final data set name, assume that the SCLM variables will contain values that are the maximum (8) number of characters.

Examples of data set name lengths are:

- APPL1.@@FLMGRP.@@FLMTYP is  $5 + 1 + 8 + 1 + 8 = 23$ .
- @@FLMPRJ.@@FLMGRP.@@FLMTYP.COMMON is  $8 + 1 + 8 + 1 + 8 + 1 + 6 = 33$ .

The data set name must meet all of the requirements specified by the MVS data set naming conventions. If the data set name is too long or it does not meet MVS data set naming conventions, errors occur during SCLM functions (for example, build or promote).

**,EXPACCT=***export\_account\_data\_set*

The name of the export accounting data set used by any group referencing this FLMALTC macro. The data set you specify must be the name of the VSAM cluster you want to use and must have a different name from any ACCT or ACCT2 parameter specified in FLMCNTRL or any FLMALTC macro. The following variables can be used in specifying the name of the export accounting data set name:

- @@FLMPRJ
- @@FLMGRP
- @@FLMUID

**,VERS=***primary\_audit\_control\_data\_set*

The name of the primary audit control data set to be used by any group referencing this FLMALTC macro. If you do not specify a VERS value, audit and versioning operations are not performed for the group. If you specify the VERS keyword and omit the primary\_audit\_control\_data\_set name, SCLM does not verify the name, and errors occur later during processing. If you do not specify a name, the value is blank.

**,VERS2=***secondary\_audit\_control\_data\_set*

The name of the secondary audit control data set to be used by any group in the project referencing this FLMALTC macro. If you specify the VERS2 keyword and omit the secondary\_audit\_control\_data\_set name, SCLM does not verify the name, and errors occur later during processing. If you do not specify a name, the value is blank.

Because additional audit record updates occur if this option is used, be aware that overall performance will degrade. Do not specify VERS2 unless you have specified VERS. If you do, an error will occur when the project definition is assembled.

**,VERPDS=***version\_pds\_name*

The name of the partitioned data set to contain the version data. The following variables can be used when specifying the name of the partitioned data set: @@FLMPRJ, @@FLMGRP, @@FLMTYP, and @@FLMDSN. For example:

- VERPDS=@@FLMPRJ.@@FLMGRP.@@FLMTYP.VERSION
- VERPDS=@@FLMDSN.VERSION
- VERPDS=@@FLMPRJ.VERSION.@@FLMGRP

This parameter is optional. If you do not specify a value, the value @@FLMDSN.VERSION is assigned to the parameter (even if versioning is not active). Refer to the description of the DSNNAME parameter for more information about the value of @@FLMDSN.

If @@FLMDSN is used, it must be specified in the first 8 characters of the VERPDS= statement to be valid. For example, VERPDS=@@FLMDSN.VERSN12 is valid, but VERPDS=@@FLMPRJ.@@FLMDSN.VERSN12 is invalid. The VERPDS parameter on the FLMALTC macro can be used to override the version data partitioned data set for a specific group or set of groups.

You can have only one VERPDS data set per group and type at a time. However, you can respecify the VERPDS data set name to control the size of the version data sets. If the VERS=primary audit control data set name remains the same, a pointer to the VERPDS that holds a particular version allows you to retrieve and delete versions of members, even if you have changed the name of the VERPDS data set.

The FLMATVER macro must be used to enable versioning for particular groups. If you specify a value of 2 or more for the VERCOUNT parameter on the FLMCNTRL macro, you must specify a separate VERPDS for each group that you intend to version.

**Note:** Failure to specify a separate VERPDS for each group can cause retrieval problems.

## Example

```

PROJXYZ  FLMABEG

          FLMCNTRL ACCT=PROJXYZ.ACCT.DATABASE

RELCNTL  FLMALTC ACCT=PROJ2.ACCT.DATABASE,          C
          DSNNAME=RELEASE.PROJ2.@@FLMGRP.@@FLMTYP

DEVCNTL  FLMALTC ACCT=PROJDEV.ACCT.DATABASE,        C
          DSNNAME=SWDEV.@@FLMPRJ.@@FLMGRP.@@FLMTYP

REL      FLMGROUP KEY=Y,ALTC=RELCNTL
INT      FLMGROUP KEY=Y,PROMOTE=REL
DEV      FLMGROUP KEY=Y,PROMOTE=INT,ALTC=DEVCNTL
    
```

The DEVCNTL FLMALTC macro defines an alternate accounting database and data set name to be used by the DEV group that references this macro. The PDS data sets associated with the DEV group have the naming convention

## FLMALTC Macro

'SWDEV.PROJXYZ.DEV.type'.

The RELCNTL FLMALTC macro defines an accounting database and data set name to be used by the REL group that references this macro. The naming convention used for the PDS data sets associated with the REL group is

'RELEASE.PROJ2.REL.type'.

---

## FLMATVER Macro

Use this macro to enable the audit and version utility and to define the group and the type of members in that group to record audit and version information for.

You must specify the name of the VSAM data sets to contain the audit information and the name of the partitioned data sets to contain the versions using the FLMCNTRL and FLMALTC macros. You can define multiple versioning partitioned data sets for a project.

Using the group and type defined in the FLMATVER macro, SCLM records information in the VSAM data set each time a member's accounting information is created, updated, or deleted within that SCLM group. This information is a record that contains the member's accounting information, the type of operation, the user ID of the user who performed the operation, and the date and time the operation occurred.

You can use the FLMATVER macro to store a version of a member. The member is stored when the particular SCLM operation (such as SAVE) has completed successfully. The version contains the information to recreate the member as it previously existed. You can disable the versioning function while maintaining the audit capabilities. Version information is captured each time an editable member or an output that is **not record format U** is created or updated, but not when it is deleted. Sequence number differences can be ignored by coding the SEQNUM parameter, otherwise, they are treated as data.

## Macro Format

```
FLMATVER  
  GROUP=group|*  
  ,TYPE=type|*  
  [,SEQNUM=STANDARD|STD|COBOL|NONE]  
  [,VERSION=YES|NO]  
  [,VERCOUNT=number_to_retain]
```

## Parameters

### GROUP|\*

The name of the group for which the audit data, version data, or both, is to be maintained. The group must be defined in the project. Use an asterisk (\*) to indicate all groups.

### ,TYPE|\*

The name of the type for which the audit data, version data, or both, is to be maintained. The type must be defined in the project. Use an asterisk (\*) to indicate all types.

Audit information can be captured for editable or noneditable types. Version information can be captured for editable types and non-editable types that are not record format U. This means that you can maintain version information for types such as "source" and "object", but not for load modules or other data that

has record format U. Therefore, if you have a project with record format U data, such as load modules, you should *not* specify TYPE=\* and VERSION=YES. If you attempt to version data that is record format U, an error message is issued during SCLM processing.

**,SEQNUM=STANDARD | STD | COBOL | NONE**

If you specify STANDARD, STD, or COBOL, SCLM ignores sequence number differences when creating a version of a member.

STANDARD or STD means ignore differences in the last eight columns of the data for fixed formats, and the first eight columns of the data for variable formats. In both cases the ignored columns are presumed to be standard sequence numbers.

COBOL means ignore differences in the first six columns of the data, which are presumed to be COBOL sequence numbers.

Omitting this parameter, or specifying NONE, indicates that all columns are to be treated as data.

**Note:** When changing the value of the SEQNUM specification for a project, also change the VERPDS specification on the FLMCNTRL or FLMALTC macros for the affected groups. Failure to do so may cause checksum verification errors when attempting to recover versions created with the previous specification (see the CHECKSUM keyword below).

**,VERSION=YES | NO**

If you specify YES, both the versioning and auditing processes are active. If you specify NO, versioning is *not* active; however, the audit process is active. If not specified, VERSION will default to NO. Version data can be captured for any editable or non-editable (output) members that are **not record format U**.

**Note:** You cannot have versioning without auditing.

**,VERCOUNT=number\_to\_retain**

The number of versions to keep in the version partitioned data set for the group or type specified. If you specify a value of zero (0), then all versions associated with a member are kept.

If you specify a value of two (2) or more, each time a member is changed the latest copy of the member is stored and the earliest copy is deleted, so that the number of versions remains constant. Any audit records that are associated with versions that have been deleted are retained, but no longer indicate that a version of the member exists. If you do specify a value of two or more, allocate a separate VERPDS for each group that has versioning enabled.

**Note:** Failure to allocate a separate VERPDS for each group can cause retrieval problems. Use the FLMALTC macro, or use the @@FLMGRP variable in the VERPDS name.

If a VERCOUNT value is not specified on the FLMATVER macro or if a value of one (1) is specified, then the value specified using the VERCOUNT parameter on the FLMCNTRL macro is used. If a VERCOUNT value is not specified on either macro, then all versions associated with a member are kept.

**CHECKSUM=YES/NO**

If you specify YES or omit this parameter, checksum verification of versions on retrieval is in effect.

## FLMATVER Macro

In the case of message FLM39220 Return Code 34, which indicates a damaged version or a version created before SEQNUM support was available in SCLM, you may do the following to override the checksum verification failure:

- Insert the CHECKSUM=NO parameter.
- Reassemble the project definition.
- Retry retrieval of the version.

Note that the validity of the of the retrieved version is not assured. *This procedure is recommended for emergency use only.*

## Example

The following statements illustrate how to capture versions of members as well as auditing information.

The first of the following statements saves versions of members with Typological and GROUP=PROD, ignoring differences in columns 1–6 where COBOL sequence numbers are expected. The second statement saves versions of your COPYBOOK members, including the non-editable members that might have been generated as a result of building a BMS member. The third statement tells SCLM to keep only the latest 2 versions of your object modules. This overrides any VERCOUNT specified on the FLMCNTRL macro for the project.

```
FLMATVER GROUP=PROD,TYPE=COBOL,VERSION=YES,SEQNUM=COBOL
FLMATVER GROUP=PROD,TYPE=COPYBOOK,VERSION=YES
FLMATVER GROUP=PROD,TYPE=OBJ,VERSION=YES,VERCOUNT=2
```

**Note:** If sequence number differences are to be ignored, full length source lines are saved in the delta file for all lines with non-sequence number differences, but lines with sequence number differences *only* are not saved in the delta file. So, when the version is retrieved, the original sequence numbers for unchanged lines are lost. Instead, the sequence numbers for the most current version are retained.

The following saves only the auditing information for members with TYPE=PASCAL and GROUP=PROD.

```
FLMATVER GROUP=PROD,TYPE=PASCAL,VERSION=NO
```

**Note:** You can omit the VERSION=NO parameter as it is the default. If omitted, versions of the member will not be saved.

The order of the FLMATVER macros is important to keep in mind. Versioning is enabled/disabled in the order specified, so after turning versioning off for GROUP=\* or TYPE=\*, any later FLMATVER macros that specify a particular GROUP or TYPE will be ignored.

In the following example, no version will be saved for PROJ.AAA.SOURCE.

```
FLMATVER GROUP=*,TYPE=*,VERSION=NO
FLMATVER GROUP=AAA,TYPE=SOURCE,VERSION=YES
```

However, if the two statements are reversed, a version of PROJ.AAA.SOURCE will be saved.

```
FLMATVER GROUP=AAA,TYPE=SOURCE,VERSION=YES
FLMATVER GROUP=*,TYPE=*,VERSION=NO
```

## FLMCNTRL Macro

Use this macro to specify project-specific control options. This macro can appear only once in any project definition. If FLMCNTRL is not specified, it will default to

```
FLMCNTRL ACCT=project.ACCOUNT.FILE
```

for any group that does not have internal data sets explicitly defined through the use of the FLMALTC macro.

### Macro Format

```
FLMCNTRL
```

```
[ACCT=primary_account_data_set|project.ACCOUNT.FILE]
```

```
[,ACCT2=secondary_account_data_set]
```

```
[,EXPACCT=export_account_data_set]
```

```
[,VERS=primary_audit_control_data_set]
```

```
[,VERS2=secondary_audit_control_data_set]
```

```
[,VSAMRLS=NO|YES]
```

```
[,VERPDS=version_pds_name]
```

```
[,VERCOUNT=number_to_retain]
```

```
[,DSNAME=dataset_name_pattern]
```

```
[,DASDUNIT=DASD_unit_name|SYSALLDA]
```

```
[,VIOUNIT=VIO_unit_name|VIO]
```

```
[,MAXLINE=max_line_count|60]
```

```
[,MAXVIO=max_vio_count|5000]
```

```
[,OPTOVER=N|Y]
```

```
[,VERCC=change_code_routine]
```

```
[,VERCCDS=change_code_dataset]
```

```
[,VERCCCM=LINK|ATTACH|TSOLNK|ISPLNK]
```

```
[,VERCCOP=change_code_options]
```

```
[,CCVFY=initial_change_code_exit_routine]
```

```
[,CCVFYDS=initial_change_code_exit_dataset]
```

```
[,CCVFYCM=LINK|ATTACH|TSOLNK|ISPLNK]
```

```
[,CCVFYOP=initial_change_code_exit_options]
```

```
[,CCSAVE=save_change_code_exit_routine]
```

```
[,CCSAVDS=save_change_code_exit_dataset]
```

```
[,CCSAVCM=LINK|ATTACH|TSOLNK|ISPLNK]
```

## FLMCNTRL Macro

```
[,CCSAVOP=save_change_code_exit_options]

[,AVDVFY=verify_audit_version_delete_exit_routine]
    [,AVDVFYDS=verify_audit_version_delete_exit_dataset]
    [,AVDVFYCM=LINK|ATTACH|TSOLNK|ISPLNK]
    [,AVDVFYOP=verify_audit_version_delete_exit_options]

[,AVDNTF=notify_audit_version_delete_exit_routine]
    [,AVDNTFDS=notify_audit_version_delete_exit_dataset]
    [,AVDNTFCM=LINK|ATTACH|TSOLNK|ISPLNK]
    [,AVDNTFOP=notify_audit_version_delete_exit_options]

[,BLDINIT=build_initial_user_exit_routine]
    [,BLDINIDS=build_initial_user_exit_dataset]
    [,BLDINICM=LINK|ATTACH|TSOLNK|ISPLNK]
    [,BLDINIOP=build_initial_user_exit_options]

[,BLDNTF=build_notify_user_exit_routine]
    [,BLDNTFDS=build_notify_user_exit_dataset]
    [,BLDNTFCM=LINK|ATTACH|TSOLNK|ISPLNK]
    [,BLDNTFOP=build_notify_user_exit_options]

[,PRMINIT=promote_initial_user_exit_routine]
    [,PRMINIDS=promote_initial_user_exit_dataset]
    [,PRMINICM=LINK|ATTACH|TSOLNK|ISPLNK]
    [,PRMINIOP=promote_initial_user_exit_options]

[,PRMVFY=promote_verify_user_exit_routine]
    [,PRMVFYDS=promote_verify_user_exit_dataset]
    [,PRMVFYCM=LINK|ATTACH|TSOLNK|ISPLNK]
    [,PRMVFYOP=promote_verify_user_exit_options]

[,PRMCPY=promote_copy_user_exit_routine]
    [,PRMCPYDS=promote_copy_user_exit_dataset]
    [,PRMCPYCM=LINK|ATTACH|TSOLNK|ISPLNK]
    [,PRMCPYOP=promote_copy_user_exit_options]
```

```
[,PRMPURGE=promote_purge_user_exit_routine]
    [,PRMPRGDS=promote_purge_user_exit_dataset]
    [,PRMPRGCM=LINK|ATTACH|TSOLNK|ISPLNK]
    [,PRMPRGOP=promote_purge_user_exit_options]

[,DELINIT=initial_delete_exit_routine]
    [,DELINIDS=initial_delete_exit_dataset]
    [,DELINICM=LINK|ATTACH|TSOLNK|ISPLNK]
    [,DELINIOP=initial_delete_exit_options]

[,DELVFY=verify_delete_exit_routine]
    [,DELVFYDS=verify_delete_exit_dataset]
    [,DELVFYCM=LINK|ATTACH|TSOLNK|ISPLNK]
    [,DELVFYOP=verify_delete_exit_options]

[,DELNTF=notify_delete_exit_routine]
    [,DELNTFDS=notify_delete_exit_dataset]
    [,DELNTFCM=LINK|ATTACH|TSOLNK|ISPLNK]
    [,DELNTFOP=notify_delete_exit_options]
```

## Parameters

**ACCT**=*primary\_account\_data\_set* | **project.ACCOUNT.FILE**

The name of the primary accounting data set for the project. The data set you specify must be the name of the VSAM cluster you want to use. The default accounting data set name is `project.ACCOUNT.FILE`, where *project* is the project name specified on the FLMABEG macro. The ACCT parameter on the FLMALTC macro can be used to override the primary accounting data set for a specific group or set of groups. No SCLM variables can be used for this parameter.

**,ACCT2**=*secondary\_account\_data\_set*

The name of a secondary accounting data set for the project. Allocate this secondary VSAM data set following the same criteria as the primary accounting data set. Choose a unique name for this data set. It should reside on a different volume than the primary one. If a severe problem occurs with the primary data set (for example, a head crash on that disk), you can use this backup data set to restore the primary data set. The default is no secondary accounting data set. The ACCT2 parameter on the FLMALTC macro can be used to override the secondary VSAM accounting data set for a specific group or set of groups.

Because additional accounting updates take place if you use this option, be aware that the updates will degrade overall performance. No SCLM variables can be used for this parameter.

## FLMCNTRL Macro

**,EXPACCT=***export\_account\_data\_set*

The name of the export accounting data set used for exporting or importing project accounting information. The data set you specify must be the name of the VSAM cluster you want to use and must have a different name from any ACCT or ACCT2 parameter specified in FLMCNTRL or any FLMALTC macro. The default is no export accounting data set. The EXPACCT parameter on the FLMALTC macro can be used to override the export accounting data set for a specific group or set of groups. The following variables can be used in specifying the name of the export accounting data set name:

- @@FLMPRJ
- @@FLMGRP
- @@FLMUID

**,VERS=***primary\_audit\_control\_data\_set*

The name of the primary audit control data set for the project. This parameter is required to perform audit and versioning for groups that do not reference an FLMALTC macro with VERS specified. If you specify the VERS keyword and omit the primary\_audit\_control\_data\_set name, errors occur later during processing. The default is no audit control data set.

**,VERS2=***secondary\_audit\_control\_data\_set*

The name of the secondary audit control data set for the project. If you specify the VERS2 keyword and omit the secondary\_audit\_control\_data\_set name, errors occur later during processing. The default is no secondary audit control data set. The VERS2 parameter on the FLMALTC macro can be used to override the secondary audit control data set for a specific group or set of groups.

Because additional audit record updates occur if this option is used, be aware that overall performance will degrade. Do not specify VERS2 unless you have specified VERS. If you do, an error will occur when the project definition is assembled.

**,VSAMRLS=**NO|YES

Indicates whether or not SCLM should allow the VSAM data sets to be shared across systems when the level of DFSMS installed is 1.3 or later. The default is NO.

SCLM uses VSAM Record Level Sharing (RLS) to allow the sharing of the VSAM data sets. To maintain the integrity of the VSAM data sets in a shared environment, the VSAM data sets must be allocated for RLS and all hardware and software to support RLS must be in place for the system. (Refer to the DFSMS documentation for hardware and software requirements.)

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 the VSAM data sets so that they cannot be accessed from multiple systems.

**,VERPDS=***version\_pds\_name*

The name of the partitioned data set to contain the version data. The following variables can be used when specifying the name of the partitioned data set: @@FLMPRJ, @@FLMGRP, and @@FLMTYP, or @@FLMDSN. For example:

- VERPDS=@@FLMPRJ.@@FLMGRP.@@FLMTYP.VERSION
- VERPDS=@@FLMDSN.VERSION
- VERPDS=@@FLMPRJ.VERSIO.@@FLMGRP

This parameter is optional. If you do not specify a value, the value @@FLMDSN.VERSION is assigned to the parameter (even if versioning is not active.) Refer to the description of the DSNAME parameter for more information about the value of @@FLMDSN.

If @@FLMDSN is used, it must be specified in the first 8 characters of the VERPDS= statement to be valid. For example, VERPDS=@@FLMDSN.VERSN12 is valid, but VERPDS=@@FLMPRJ.@@FLMDSN.VERSN12 is not valid. The VERPDS parameter on the FLMALTC macro can be used to override the version data partitioned data set for a specific group or set of groups.

You can have only one VERPDS data set per group and type at a time. However, you can respecify the VERPDS data set name to control the size of the version data sets. If the VERS=primary audit control data set name remains the same, a pointer to the VERPDS that holds a particular version allows you to retrieve and delete versions of members, even if you have changed the name of the VERPDS data set.

The FLMATVER macro must be used to enable versioning for particular groups. If you specify a value of 2 or more for the VERCOUNT parameter on the FLMCNTRL macro, you must specify a separate VERPDS for each group that you intend to version.

**Note:** Failure to specify a separate VERPDS for each group can cause retrieval problems.

**,VERCOUNT=***number\_to\_retain*

The number of versions to keep in the version partitioned data set. If you specify a value of 0 (the default), all versions associated with a member will be kept. If you specify a value of 2 or more, each time a member is saved or promoted, the latest copy of the version is stored and the earliest copy is disposed. Any audit records that were associated with the version are retained but will no longer indicate that a version of the member exists. If you do specify a value of 2 or more, allocate a separate VERPDS for each group that has versioning enabled.

**Note:** Failure to allocate a separate VERPDS for each group can cause retrieval problems. Use the FLMALTC macro, or use the @@FLMGRP variable in the VERPDS name.

If you specify a value of 1, an error will occur when the project definition is assembled. The only version maintained in this case would be a full source copy of the member that exists in the project hierarchy.

**,DSNAME=***dataset\_name*

This parameter lets you specify the data set naming conventions for the project partitioned data sets controlled by SCLM. The naming convention is specified as a pattern that can include a subset of the SCLM variables.

The only SCLM variables that can be used in the DSNAME parameter of FLMCNTRL are:

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

The value specified in this parameter is used to resolve the SCLM variable @@FLMDSN. If this parameter is not specified, the data set name pattern

## FLMCNTRL Macro

defaults to @@FLMPRJ.@@FLMGRP.@@FLMTYP. You can enter up to 44 characters for this parameter, including the SCLM variables and the periods.

If a data set name is specified, it must include the SCLM variable @@FLMTYP. It is also recommended that the variable @@FLMGRP be used in the data set name pattern. This helps prevent data from one group overwriting data in another group.

**Note: SCLM does not enforce or verify the uniqueness of partitioned data set names.**

The DSNAMES parameter on the FLMALTC macro can be used to override the data set naming conventions for a specific group or set of groups.

The variables can appear in any location within the DSNAMES parameter. Any user-specified qualifiers can also be used. The preceding SCLM variables can contain values up to 8 characters.

Examples of data set name lengths are:

- APPL4.@@FLMGRP.@@FLMTYP is  $5 + 1 + 8 + 1 + 8 = 23$ .
- REL30.COMMON2A.@@FLMGRP.@@FLMTYP is  $5 + 1 + 8 + 1 + 8 + 1 + 8 = 32$ .

The resulting data set name must meet all of the requirements specified by the MVS data set naming conventions. If the data set name is too long or it does not meet MVS data set naming conventions, then errors occur during SCLM functions (for example, Build or Promote).

**,DASDUNIT=***dasd\_unit\_name* | **SYSALLDA**

The name of the unit where DASD data sets will reside. The maximum DASD unit name length is 8 characters. The default is SYSALLDA.

**,VIOUNIT=***VIO\_unit\_name* | **VIO**

The name of the unit where a temporary VIO data set will reside. The maximum VIO unit name length is 8 characters. The default is VIO. For more information on MAXVIO, see MAXVIO on page 160.

**,MAXLINE=***max\_line\_count* | **60**

An integer value indicating the maximum number of lines per page for all SCLM reports. The minimum value you can specify is 35, and the default is 60.

**,MAXVIO=***max\_vio\_count* | **5000**

An integer value indicating the maximum number of records permitted for VIO allocation. The default is 5000. The maximum value is 2147483647.

**,OPTOVER=N** | **Y**

Indicates whether translator option overrides are allowed or disallowed. If OPTOVER=Y, developers can override the translator options by specifying the keyword PARMx in the architecture member followed by the new options. The default is Y. See *ISPF Software Configuration and Library Manager (SCLM) Developer's and Project Manager's Guide* for more information about PARMx.

**,VERCC=***change\_code\_routine*

The member name of the change code verification routine. Specify the data set containing the member in the VERCCDS parameter. If you do not specify the VERCC parameter, then SCLM does not invoke the exit routine.

**,VERCCDS=***change\_code\_dataset*

The name of the data set containing the translator load module, REXX exec, or

CLIST specified by the VERCC parameter. The data set name is not required when the translator resides in one of the system concatenation libraries. The data set name can be up to 44 characters long.

**,VERCCM=LINK | ATTACH | TSOLNK | ISPLNK**

Indicates whether the translator is to be linked, attached, or invoked by the TSO service facility routine or called through ISPF services. Use ATTACH for load modules unless you need access to ISPF variables or services. In that case, use LINK. Using LINK can result in loops or out-of-space abends because storage is not freed between calls to the translators.

TSOLNK is for translators written as REXX execs. TSOLNK results in the translator being invoked from IKJEFTSR (TSO service facility routine) with parameter 1 of x'00010001'. This parameter indicates that the TSO service facility should invoke the requested translator from an unauthorized environment and that the translator can be a TSO command, REXX exec, or CLIST.

ISPLNK is for translators that must have access to ISPF variables or services. The value specified on the VERCC parameter is the ISPF service that is used to call the translator. The only supported value is SELECT. The keywords, including the command to run, are specified in the VERCCOP parameter. The name of the load module, CLIST, REXX exec, or other command is also specified as part of the VERCCOP parameter.

The default is LINK.

**,VERCCOP=change\_code\_options**

Option list to be passed to the VERCC user exit routine. You can specify a maximum of 255 characters for the options, including delimiters. Enclose the option string in parentheses or single quotes. The options string precedes the list of parameters passed to the exit routine by SCLM. SCLM removes any trailing blanks and does not add a delimiter between the option string and the SCLM parameters. End the options string with a non-blank delimiter so that the options and parameters can be identified by the exit routine.

When the call method for the exit routine, VERCCCM, is ISPLNK, the options string must contain the keywords and parameters for the ISPF SELECT service. The options must be in the format expected by the service. For more information about the ISPF SELECT service, refer to the *ISPF Services Guide and Reference*.

**,CCVFY=verify\_change\_code\_exit\_routine**

The name of the verify change code exit routine. If you do not specify the CCVFY parameter, SCLM does not invoke the exit routine.

**,CCVFYDS=verify\_change\_code\_exit\_dataset**

The name of the data set containing the translator load module, REXX exec or CLIST specified by the CCVFY parameter. The data set name is not required when the translator resides in one of the system concatenation libraries. The data set name can be up to 44 characters.

**,CCVFYCM=LINK | ATTACH | TSOLNK | ISPLNK**

Indicates whether the exit routine is to be linked, attached, invoked by the TSO service facility routine or called through ISPF services. Use ATTACH for load modules unless you need access to ISPF variables or services; in that case, use LINK. Using LINK can result in loops or out-of-space abends because storage is not freed between calls to the translators.

TSOLNK is for translators written as REXX execs. TSOLNK results in the translator being invoked from IKJEFTSR (TSO service facility routine) with

## FLMCNTRL Macro

parameter 1 of x'00010001'. This parameter indicates that the TSO service facility should invoke the requested translator from an unauthorized environment and that the translator can be a TSO command, REXX exec, or CLIST.

ISPLNK is for translators that must have access to ISPF variables or services. The value specified on the CCVfy parameter is the ISPF service that is used to call the translator. The only supported value is SELECT. The keywords, including the command to run, are specified in the CCVfyOP parameter. The name of the load module, CLIST, REXX exec or other command is also specified as part of the CCVfyOP parameter.

The default is LINK.

### **,CCVfyOP=***verify\_change\_code\_exit\_options*

Option list to be passed to the CCVfy user exit routine. You can specify a maximum of 255 characters for the options, including delimiters. Enclose the option string in parentheses or single quotes. The options string precedes the list of parameters passed to the exit routine by SCLM. SCLM removes any trailing blanks and does not add a delimiter between the option string and the SCLM parameters. End the options string with a non-blank delimiter so that the options and parameters can be identified by the exit routine.

When the call method for the exit routine, CCVfyCM, is ISPLNK, the options string must contain the keywords and parameters for the ISPF SELECT service. The options must be in the format expected by the service. For a description of the ISPF SELECT service, refer to the 'ISPF Services Guide'.

### **,CCSAVE=***save\_change\_code\_exit\_routine*

The name of the save change code exit routine. If you do not specify the CCSAVE parameter, SCLM does not invoke the exit routine.

### **,CCSAVDS=***save\_change\_code\_exit\_dataset*

The name of the data set containing the translator load module, REXX exec or CLIST specified by the CCSAVE parameter. The data set name is not required when the translator resides in one of the system concatenation libraries. The data set name can be up to 44 characters.

### **,CCSAVCM=**LINK | ATTACH | TSOLNK | ISPLNK

Indicates whether the exit routine is to be linked, attached, invoked by the TSO service facility routine or called through ISPF services. Use ATTACH for load modules unless you need access to ISPF variables or services; in that case, use LINK. Using LINK can result in loops or out-of-space abends because storage is not freed between calls to the translators.

TSOLNK is for translators written as REXX execs. TSOLNK results in the translator being invoked from IKJEFTSR (TSO service facility routine) with parameter 1 of x'00010001'. This parameter indicates that the TSO service facility should invoke the requested translator from an unauthorized environment and that the translator can be a TSO command, REXX exec, or CLIST.

ISPLNK is for translators that must have access to ISPF variables or services. The value specified on the CCSAVE parameter is the ISPF service that is used to call the translator. The only supported value is SELECT. The keywords, including the command to run, are specified in the CCSAVOP parameter. The name of the load module, CLIST, REXX exec or other command is also specified as part of the CCSAVOP parameter.

The default is LINK.

**,CCSAVOP=save\_change\_code\_exit\_options**

Option list to be passed to the CCSAVE user exit routine. You can specify a maximum of 255 characters for the options, including delimiters. Enclose the option string in parentheses or single quotes. The options string precedes the list of parameters passed to the exit routine by SCLM. SCLM removes any trailing blanks and does not add a delimiter between the option string and the SCLM parameters. End the options string with a non-blank delimiter so that the options and parameters can be identified by the exit routine.

When the call method for the exit routine, CCSAVCM, is ISPLNK, the options string must contain the keywords and parameters for the ISPF SELECT service. The options must be in the format expected by the service. For a description of the ISPF SELECT service, refer to the 'ISPF Services Guide'.

**,AVDVFY=verify\_audit\_version\_delete\_exit\_routine**

The name of the audit version delete verification exit routine. If you do not specify the DELVFY parameter, SCLM does not invoke the exit routine.

**,AVDVFYDS=verify\_audit\_version\_delete\_exit\_dataset**

The name of the data set containing the translator load module, REXX exec, or CLIST specified by the AVDVFY parameter. The data set name is not required when the translator resides in one of the system concatenation libraries. The data set name can be up to 44 characters long.

**,AVDVFYCM=LINK|ATTACH|TSOLNK|ISPLNK**

Indicates whether the exit routine is to be linked, attached, invoked by the TSO service facility routine, or called through ISPF services. Use ATTACH for load modules unless you need access to ISPF variables or services. In that case, use LINK. Using LINK can result in loops or out-of-space abends because storage is not freed between calls to the translators.

TSOLNK is for translators written as REXX execs. TSOLNK results in the translator being invoked from IKJEFTSR (TSO service facility routine) with parameter 1 of x'00010001'. This parameter indicates that the TSO service facility should invoke the requested translator from an unauthorized environment and that the translator can be a TSO command, REXX exec, or CLIST.

ISPLNK is for translators that must have access to ISPF variables or services. The value specified on the AVDVFY parameter is the ISPF service that is used to call the translator. The only supported value is SELECT. The keywords, including the command to run, are specified in the AVDVFYOP parameter. The name of the load module, CLIST, REXX exec, or other command is also specified as part of the AVDVFYOP parameter.

The default is LINK.

**,AVDVFYOP=verify\_audit\_version\_delete\_exit\_options**

Option list to be passed to the AVDVFY user exit routine. You can specify a maximum of 255 characters for the options, including delimiters. Enclose the option string in parentheses or single quotes. The options string precedes the list of parameters passed to the exit routine by SCLM. SCLM removes any trailing blanks and does not add a delimiter between the option string and the SCLM parameters. End the options string with a non-blank delimiter so that the options and parameters can be identified by the exit routine.

When the call method for the exit routine, AVDVFYCM, is ISPLNK, the options string must contain the keywords and parameters for the ISPF SELECT service.

## FLMCNTRL Macro

The options must be in the format expected by the service. For more information about the ISPF SELECT service, refer to the *ISPF Services Guide and Reference*.

**,AVDNTF=***notify\_audit\_version\_delete\_exit\_routine*

The name of the audit version delete notification exit routine. If you do not specify the AVDNTF parameter, SCLM does not invoke the exit routine.

**,AVDNTFDS=***notify\_audit\_version\_delete\_exit\_dataset*

The name of the data set containing the translator load module, REXX exec, or CLIST specified by the AVDNTF parameter. The data set name is not required when the translator resides in one of the system concatenation libraries. The data set name can be up to 44 characters long.

**,AVDNTFCM=**LINK | ATTACH | TSOLNK | ISPLNK

Indicates whether the exit routine is to be linked, attached, invoked by the TSO service facility routine, or called through ISPF services. Use ATTACH for load modules unless you need access to ISPF variables or services. In that case, use LINK. Using LINK can result in loops or out-of-space abends because storage is not freed between calls to the translators.

TSOLNK is for translators written as REXX execs. TSOLNK results in the translator being invoked from IKJEFTSR (TSO service facility routine) with parameter 1 of x'00010001'. This parameter indicates that the TSO service facility should invoke the requested translator from an unauthorized environment and that the translator can be a TSO command, REXX exec, or CLIST.

ISPLNK is for translators that must have access to ISPF variables or services. The value specified on the AVDVFY parameter is the ISPF service that is used to call the translator. The only supported value is SELECT. The keywords, including the command to run, are specified in the AVDNTFOP parameter. The name of the load module, CLIST, REXX exec, or other command is also specified as part of the AVDNTFOP parameter.

The default is LINK.

**,AVDNTFOP=***verify\_audit\_version\_delete\_exit\_options*

Option list to be passed to the AVDNTF user exit routine. You can specify a maximum of 255 characters for the options, including delimiters. Enclose the option string in parentheses or single quotes. The options string precedes the list of parameters passed to the exit routine by SCLM. SCLM removes any trailing blanks and does not add a delimiter between the option string and the SCLM parameters. End the options string with a non-blank delimiter so that the options and parameters can be identified by the exit routine.

When the call method for the exit routine, AVDNTFCM, is ISPLNK, the options string must contain the keywords and parameters for the ISPF SELECT service. The options must be in the format expected by the service. For more information about the ISPF SELECT service, refer to the *ISPF Services Guide and Reference*.

**,BLDINIT=***build\_initial\_user\_exit\_routine*

The member name of the initial build user exit routine. SCLM invokes the routine at the beginning of the build process during initialization. Specify the data set containing the member using the BLDINIDS parameter. If you do not specify the BLDINIT parameter, then SCLM does not invoke the exit routine.

**,BLDINIDS=***build\_initial\_user\_exit\_dataset*

The name of the data set containing the translator load module, REXX exec, or CLIST specified by the BLDINIT parameter. The data set name is not required

when the translator resides in one of the system concatenation libraries. The data set name can be up to 44 characters long.

**,BLDINICM=LINK|ATTACH|TSOLNK|ISPLNK**

Indicates whether the translator is to be linked, attached, or invoked by the TSO service facility routine or called through ISPF services. Use ATTACH for load modules unless you need access to ISPF variables or services. In that case, use LINK. Using LINK can result in loops or out-of-space abends because storage is not freed between calls to the translators.

TSOLNK is for translators written as REXX execs. TSOLNK results in the translator being invoked from IKJEFTSR (TSO service facility routine) with parameter 1 of x'00010001'. This parameter indicates that the TSO service facility should invoke the requested translator from an unauthorized environment and that the translator can be a TSO command, REXX exec, or CLIST.

ISPLNK is for translators that must have access to ISPF variables or services. The value specified on the BLDINIT parameter is the ISPF service that is used to call the translator. The only supported value is SELECT. The keywords, including the command to run, are specified in the BLDINIOP parameter. The name of the load module, CLIST, REXX exec, or other command is also specified as part of the BLDINIOP parameter.

The default is LINK.

**,BLDINIOP=build\_initial\_user\_exit\_options**

Option list to be passed to the BLDINIT user exit routine. You can specify a maximum of 255 characters for the options, including delimiters. Enclose the option string in parentheses or single quotes. The options string precedes the list of parameters passed to the exit routine by SCLM. SCLM removes any trailing blanks and does not add a delimiter between the option string and the SCLM parameters. End the options string with a non-blank delimiter so that the options and parameters can be identified by the exit routine.

When the call method for the exit routine, BLDINICM, is ISPLNK, the options string must contain the keywords and parameters for the ISPF SELECT service. The options must be in the format expected by the service. For more information about the ISPF SELECT service, refer to the *ISPF Services Guide and Reference*.

**,BLDNTF=build\_notify\_user\_exit\_routine**

The member name of the build notification user exit routine. SCLM invokes the routine at the end of the build process after the build has taken place. Specify the data set containing the member using the BLDNTFDS parameter. If you do not specify the BLDNTF parameter, then SCLM does not invoke the exit routine.

**Note:** The original format for this user exit, using the BLDEXT1 parameter, is still supported by SCLM. However, parameters used with this exit routine must be either *ALL* old format or *ALL* new format. Specifying the user exit routine in both the old and new formats, or mixing old and new format parameters for the same exit causes errors when the project definition is assembled.

**,BLDNTFDS=build\_notify\_user\_exit\_dataset**

The name of the data set containing the translator load module, REXX exec, or CLIST specified by the BLDNTF parameter. The data set name is not required when the translator resides in one of the system concatenation libraries. The data set name can be up to 44 characters long.

### **,BLDNTFCM=LINK | ATTACH | TSOLNK | ISPLNK**

Indicates whether the translator is to be linked, attached, or invoked by the TSO service facility routine or called through ISPF services. Use ATTACH for load modules unless you need access to ISPF variables or services. In that case, use LINK. Using LINK can result in loops or out-of-space abends because storage is not freed between calls to the translators.

TSOLNK is for translators written as REXX execs. TSOLNK results in the translator being invoked from IKJEFTSR (TSO service facility routine) with parameter 1 of x'00010001'. This parameter indicates that the TSO service facility should invoke the requested translator from an unauthorized environment and that the translator can be a TSO command, REXX exec, or CLIST.

ISPLNK is for translators that must have access to ISPF variables or services. The value specified on the BLDNTF parameter is the ISPF service that is used to call the translator. The only supported value is SELECT. The keywords, including the command to run, are specified in the BLDNTFOP parameter. The name of the load module, CLIST, REXX exec, or other command is also specified as part of the BLDNTFOP parameter.

The default is LINK.

### **,BLDNTFOP=*build\_notify\_user\_exit\_options***

Option list to be passed to the BLDNTF user exit routine. You can specify a maximum of 255 characters for the options, including delimiters. Enclose the option string in parentheses or single quotes. The options string precedes the list of parameters passed to the exit routine by SCLM. SCLM removes any trailing blanks and does not add a delimiter between the option string and the SCLM parameters. End the options string with a non-blank delimiter so that the options and parameters can be identified by the exit routine.

When the call method for the exit routine, BLDNTFCM, is ISPLNK, the options string must contain the keywords and parameters for the ISPF SELECT service. The options must be in the format expected by the service. For more information about the ISPF SELECT service, refer to the *ISPF Services Guide and Reference*.

### **,PRMINIT=*promote\_initial\_user\_exit\_routine***

The member name of the initial promote user exit routine. SCLM invokes this routine at the beginning of the promote process during initialization. Specify the data set containing the member in the PRMINIDS parameter. If you do not specify the PRMINIT parameter, then SCLM does not invoke the exit routine.

### **,PRMINIDS=*promote\_initial\_user\_exit\_dataset***

The name of the data set containing the translator load module, REXX exec, or CLIST specified by the PRMINIT parameter. The data set name is not required when the translator resides in one of the system concatenation libraries. The data set name can be up to 44 characters long.

### **,PRMINICM=LINK | ATTACH | TSOLNK | ISPLNK**

Indicates whether the translator is to be linked, attached, or invoked by the TSO service facility routine or called through ISPF services. Use ATTACH for load modules unless you need access to ISPF variables or services. In that case, use LINK. Using LINK can result in loops or out\_of\_space abends because storage is not freed between calls to the translators.

TSOLNK is for translators written as REXX execs. TSOLNK results in the translator being invoked from IKJEFTSR (TSO service facility routine) with parameter 1 of x'00010001'. This parameter indicates that the TSO service facility

should invoke the requested translator from an unauthorized environment and that the translator can be a TSO command, REXX exec, or CLIST.

ISPLNK is for translators that must have access to ISPF variables or services. The value specified on the PRMINIT parameter is the ISPF service that is used to call the translator. The only supported value is SELECT. The keywords, including the command to run, are specified in the PRMINIOP parameter. The name of the load module, CLIST, REXX exec, or other command is also specified as part of the PRMINIOP parameter.

The default is LINK.

**,PRMINIOP**=*promote\_initial\_user\_exit\_options*

Option list to be passed to the PRMINIT user exit routine. You can specify a maximum of 255 characters for the options, including delimiters. Enclose the option string in parentheses or single quotes. The options string precedes the list of parameters passed to the exit routine by SCLM. SCLM removes any trailing blanks and does not add a delimiter between the option string and the SCLM parameters. End the options string with a non-blank delimiter so that the options and parameters can be identified by the exit routine.

When the call method for the exit routine, PRMINICM, is ISPLNK, the options string must contain the keywords and parameters for the ISPF SELECT service. The options must be in the format expected by the service. For more information about the ISPF SELECT service, refer to the *ISPF Services Guide and Reference*.

**,PRMVFY**=*promote\_verify\_user\_exit\_routine*

The member name of the promote verification user exit routine. SCLM invokes this routine at the end of the verification phase of the promote process. Specify the data set containing the member in the PRMVFYDS parameter. If you do not specify the PRMVFY parameter, then SCLM does not invoke the exit routine.

**Note:** The original format for this user exit, using the PRMEXT1 parameter, is still supported by SCLM. However, parameters used with this exit routine must be either *ALL* old format or *ALL* new format. Specifying the user exit routine in both the old and new formats, or mixing old and new format parameters for the same exit causes errors when the project definition is assembled.

**,PRMVFYDS**=*promote\_verify\_user\_exit\_dataset*

The name of the data set containing the translator load module, REXX exec, or CLIST specified by the PRMVFY parameter. The data set name is not required when the translator resides in one of the system concatenation libraries. The data set name can be up to 44 characters long.

**,PRMVFYCM**=LINK|ATTACH|TSOLNK|ISPLNK

Indicates whether the translator is to be linked, attached, or invoked by the TSO service facility routine or called through ISPF services. Use ATTACH for load modules unless you need access to ISPF variables or services. In that case, use LINK. Using LINK can result in loops or out-of-space abends because storage is not freed between calls to the translators.

TSOLNK is for translators written as REXX execs. TSOLNK results in the translator being invoked from IKJEFTSR (TSO service facility routine) with parameter 1 of x'00010001'. This parameter indicates that the TSO service facility should invoke the requested translator from an unauthorized environment and that the translator can be a TSO command, REXX exec, or CLIST.

## FLMCNTRL Macro

ISPLNK is for translators that must have access to ISPF variables or services. The value specified on the PRMVFY parameter is the ISPF service that is used to call the translator. The only supported value is SELECT. The keywords, including the command to run, are specified in the PRMVFYOP parameter. The name of the load module, CLIST, REXX exec, or other command is also specified as part of the PRMVFYOP parameter.

The default is LINK.

### **,PRMVFYOP**=*promote\_verify\_user\_exit\_options*

Option list to be passed to the PRMVFY user exit routine. You can specify a maximum of 255 characters for the options, including delimiters. Enclose the option string in parentheses or single quotes. The options string precedes the list of parameters passed to the exit routine by SCLM. SCLM removes any trailing blanks and does not add a delimiter between the option string and the SCLM parameters. End the options string with a non-blank delimiter so that the options and parameters can be identified by the exit routine.

When the call method for the exit routine, PRMVFYCM, is ISPLNK, the options string must contain the keywords and parameters for the ISPF SELECT service. The options must be in the format expected by the service. For more information about the ISPF SELECT service, refer to the *ISPF Services Guide and Reference*.

### **,PRMCOPY**=*promote\_copy\_user\_exit\_routine*

The member name of the promote copy user exit routine. SCLM invokes this routine at the end of the copy phase of the promote process. Specify the data set containing the member in the PRMCOPYDS parameter. If you do not specify the PRMCOPY parameter, then SCLM does not invoke the exit routine.

**Note:** The original format for this user exit, using the PRMEXT2 parameter, is still supported by SCLM. However, parameters used with this exit routine must be either *ALL* old format or *ALL* new format. Specifying the user exit routine in both the old and new formats, or mixing old and new format parameters for the same exit causes errors when the project definition is assembled.

### **,PRMCPYDS**=*promote\_copy\_user\_exit\_dataset*

The name of the data set containing the translator load module, REXX exec, or CLIST specified by the PRMCOPY parameter. The data set name is not required when the translator resides in one of the system concatenation libraries. The data set name can be up to 44 characters long.

### **,PRMCPYCM**=LINK|ATTACH|TSOLNK|ISPLNK

Indicates whether the translator is to be linked, attached, or invoked by the TSO service facility routine or called through ISPF services. Use ATTACH for load modules unless you need access to ISPF variables or services. In that case, use LINK. Using LINK can result in loops or out-of-space abends because storage is not freed between calls to the translators.

TSOLNK is for translators written as REXX execs. TSOLNK results in the translator being invoked from IKJEFTSR (TSO service facility routine) with parameter 1 of x'00010001'. This parameter indicates that the TSO service facility should invoke the requested translator from an unauthorized environment and that the translator can be a TSO command, REXX exec, or CLIST.

ISPLNK is for translators that must have access to ISPF variables or services. The value specified on the PRMCOPY parameter is the ISPF service that is used to call the translator. The only supported value is SELECT. The keywords,

including the command to run, are specified in the PRMCOPYOP parameter. The name of the load module, CLIST, REXX exec, or other command is also specified as part of the PRMCOPYOP parameter.

The default is LINK.

**,PRMCOPYOP**=*promote\_copy\_user\_exit\_options*

Option list to be passed to the PRMCOPY user exit routine. You can specify a maximum of 255 characters for the options, including delimiters. Enclose the option string in parentheses or single quotes. The options string precedes the list of parameters passed to the exit routine by SCLM. SCLM removes any trailing blanks and does not add a delimiter between the option string and the SCLM parameters. End the options string with a non-blank delimiter so that the options and parameters can be identified by the exit routine.

When the call method for the exit routine, PRMCOPYCM, is ISPLNK, the options string must contain the keywords and parameters for the ISPF SELECT service. The options must be in the format expected by the service. For more information about the ISPF SELECT service, refer to the *ISPF Services Guide and Reference*.

**,PRMPURGE**=*promote\_purge\_user\_exit\_routine*

The member name of the promote purge user exit routine. SCLM invokes this routine at the end of the copy phase of the promote process. Specify the data set containing the member in the PRMPRGDS parameter. If you do not specify the PRMPURGE parameter, then SCLM does not invoke the exit routine.

**Note:** The original format for this user exit, using the PRMEXT3 parameter, is still supported by SCLM. However, parameters used with this exit routine must be either *ALL* old format or *ALL* new format. Specifying the user exit routine in both the old and new formats, or mixing old and new format parameters for the same exit causes errors when the project definition is assembled.

**,PRMPRGDS**=*promote\_purge\_user\_exit\_dataset*

The name of the data set containing the translator load module, REXX exec, or CLIST specified by the PRMPURGE parameter. The data set name is not required when the translator resides in one of the system concatenation libraries. The data set name can be up to 44 characters long.

**,PRMPRGCM**=LINK | ATTACH | TSOLNK | ISPLNK

Indicates whether the translator is to be linked, attached, or invoked by the TSO service facility routine or called through ISPF services. Use ATTACH for load modules unless you need access to ISPF variables or services. In that case, use LINK. Using LINK can result in loops or out-of-space abends because storage is not freed between calls to the translators.

TSOLNK is for translators written as REXX execs. TSOLNK results in the translator being invoked from IKJEFTSR (TSO service facility routine) with parameter 1 of x'00010001'. This parameter indicates that the TSO service facility should invoke the requested translator from an unauthorized environment and that the translator can be a TSO command, REXX exec, or CLIST.

ISPLNK is for translators that must have access to ISPF variables or services. The value specified on the PRMPURGE parameter is the ISPF service that is used to call the translator. The only supported value is SELECT. The keywords, including the command to run, are specified in the PRMPRGOP parameter. The name of the load module, CLIST, REXX exec, or other command is also specified as part of the PRMPRGOP parameter.

## FLMCNTRL Macro

The default is LINK.

### **,PRMPRGOP**=*promote\_purge\_user\_exit\_options*

Option list to be passed to the PRMPURGE user exit routine. You can specify a maximum of 255 characters for the options, including delimiters. Enclose the option string in parentheses or single quotes. The options string precedes the list of parameters passed to the exit routine by SCLM. SCLM removes any trailing blanks and does not add a delimiter between the option string and the SCLM parameters. End the options string with a non-blank delimiter so that the options and parameters can be identified by the exit routine.

When the call method for the exit routine, PRMPRGCM, is ISPLNK, the options string must contain the keywords and parameters for the ISPF SELECT service. The options must be in the format expected by the service. For more information about the ISPF SELECT service, refer to the *ISPF Services Guide and Reference*.

### **,DELINIT**=*initial\_delete\_exit\_routine*

The name of the initial delete exit routine. If you do not specify the DELINIT parameter, then SCLM does not invoke the exit routine. This routine is only invoked for the Delete Group (DELGROU) service or dialog (ISPF Option 10.3.9).

### **,DELINIDS**=*initial\_delete\_exit\_dataset*

The name of the data set containing the translator load module, REXX exec, or CLIST specified by the DELINIT parameter. The data set name is not required when the translator resides in one of the system concatenation libraries. The data set name can be up to 44 characters long.

### **,DELINICM**=LINK | ATTACH | TSOLNK | ISPLNK

Indicates whether the translator is to be linked, attached, or invoked by the TSO service facility routine or called through ISPF services. Use ATTACH for load modules unless you need access to ISPF variables or services. In that case, use LINK. Using LINK can result in loops or out\_of\_space abends because storage is not freed between calls to the translators.

TSOLNK is for translators written as REXX execs. TSOLNK results in the translator being invoked from IKJEFTSR (TSO service facility routine) with parameter 1 of x'00010001'. This parameter indicates that the TSO service facility should invoke the requested translator from an unauthorized environment and that the translator can be a TSO command, REXX exec, or CLIST.

ISPLNK is for translators that must have access to ISPF variables or services. The value specified on the DELINIT parameter is the ISPF service that is used to call the translator. The only supported value is SELECT. The keywords, including the command to run, are specified in the DELINIOP parameter. The name of the load module, CLIST, REXX exec, or other command is also specified as part of the DELINIOP parameter.

The default is LINK.

### **,DELINIOP**=*initial\_delete\_exit\_options*

Option list to be passed to the DELINIT user exit routine. You can specify a maximum of 255 characters for the options, including delimiters. Enclose the option string in parentheses or single quotes. The options string precedes the list of parameters passed to the exit routine by SCLM. SCLM removes any trailing blanks and does not add a delimiter between the option string and the SCLM parameters. End the options string with a non-blank delimiter so that the options and parameters can be identified by the exit routine.

When the call method for the exit routine, DELINICM, is ISPLNK, the options string must contain the keywords and parameters for the ISPF SELECT service. The options must be in the format expected by the service. For more information about the ISPF SELECT service, refer to the *ISPF Services Guide and Reference*.

**,DELVFY=verify\_delete\_exit\_routine**

The name of the delete verification exit routine. If you do not specify the DELVFY parameter, then SCLM does not invoke the exit routine. This exit routine is invoked for Library Utility Delete (ISPF Option 10.3.1) or the Delete service.

**,DELVFYDS=verify\_delete\_exit\_dataset**

The name of the data set containing the translator load module, REXX exec, or CLIST specified by the DELVFY parameter. The data set name is not required when the translator resides in one of the system concatenation libraries. The data set name can be up to 44 characters long.

**,DELVFYCM=LINK | ATTACH | TSOLNK | ISPLNK**

Indicates whether the translator is to be linked, attached, or invoked by the TSO service facility routine or called through ISPF services. Use ATTACH for load modules unless you need access to ISPF variables or services. In that case, use LINK. Using LINK can result in loops or out-of-space abends because storage is not freed between calls to the translators.

TSOLNK is for translators written as REXX execs. TSOLNK results in the translator being invoked from IKJEFTSR (TSO service facility routine) with parameter 1 of x'00010001'. This parameter indicates that the TSO service facility should invoke the requested translator from an unauthorized environment and that the translator can be a TSO command, REXX exec, or CLIST.

ISPLNK is for translators that must have access to ISPF variables or services. The value specified on the DELVFY parameter is the ISPF service that is used to call the translator. The only supported value is SELECT. The keywords, including the command to run, are specified in the DELVFYOP parameter. The name of the load module, CLIST, REXX exec, or other command is also specified as part of the DELVFYOP parameter.

The default is LINK.

**,DELVFYOP=verify\_delete\_exit\_options**

Option list to be passed to the DELVFY user exit routine. You can specify a maximum of 255 characters for the options, including delimiters. Enclose the option string in parentheses or single quotes. The options string precedes the list of parameters passed to the exit routine by SCLM. SCLM removes any trailing blanks and does not add a delimiter between the option string and the SCLM parameters. End the options string with a non-blank delimiter so that the options and parameters can be identified by the exit routine.

When the call method for the exit routine, DELVFYCM, is ISPLNK, the options string must contain the keywords and parameters for the ISPF SELECT service. The options must be in the format expected by the service. For more information about the ISPF SELECT service, refer to the *ISPF Services Guide and Reference*.

**,DELNTF=notify\_delete\_exit\_routine**

The name of the delete notification exit routine. If you do not specify the DELNTF parameter, then SCLM does not invoke the exit routine. This exit is

## FLMCNTRL Macro

invoked for Library Utility Delete (ISPF Option 10.3.1), the Delete Group (DELGROU) service or dialog (ISPF Option 10.3.9) or Delete service.

**,DELNTFDS=notify\_delete\_exit\_dataset**

The name of the data set containing the translator load module, REXX exec, or CLIST specified by the DELNTF parameter. The data set name is not required when the translator resides in one of the system concatenation libraries. The data set name can be up to 44 characters long.

**,DELNTFCM=LINK|ATTACH|TSOLNK|ISPLNK**

Indicates whether the translator is to be linked, attached, or invoked by the TSO service facility routine or called through ISPF services. Use ATTACH for load modules unless you need access to ISPF variables or services. In that case, use LINK. Using LINK can result in loops or out-of-space abends because storage is not freed between calls to the translators.

TSOLNK is for translators written as REXX execs. TSOLNK results in the translator being invoked from IKJEFTSR (TSO service facility routine) with parameter 1 of x'00010001'. This parameter indicates that the TSO service facility should invoke the requested translator from an unauthorized environment and that the translator can be a TSO command, REXX exec, or CLIST.

ISPLNK is for translators that must have access to ISPF variables or services. The value specified on the DELNTF parameter is the ISPF service that is used to call the translator. The only supported value is SELECT. The keywords, including the command to run, are specified in the DELNTFOP parameter. The name of the load module, CLIST, REXX exec, or other command is also specified as part of the DELNTFOP parameter.

The default is LINK.

**,DELNTFOP=notify\_delete\_exit\_options**

Option list to be passed to the DELNTF user exit routine. You can specify a maximum of 255 characters for the options, including delimiters. Enclose the option string in parentheses or single quotes. The options string precedes the list of parameters passed to the exit routine by SCLM. SCLM removes any trailing blanks and does not add a delimiter between the option string and the SCLM parameters. End the options string with a non-blank delimiter so that the options and parameters can be identified by the exit routine.

When the call method for the exit routine, DELNTFCM, is ISPLNK, the options string must contain the keywords and parameters for the ISPF SELECT service. The options must be in the format expected by the service. For more information about the ISPF SELECT service, refer to the *ISPF Services Guide and Reference*.

## Example

The following information has been specified: the accounting data set, allocation of data sets using VIO if the FLMALLOC macro RECNUM parameter is not greater than 10000 and a verify change code exit routine that is called using ISPLNK.

```
FLMCNTRL ACCT=PROJ1.ACCOUNT.FILE,           X
        CCVFY=SELECT,                         X
        CCVFYCM=ISPLNK,                       X
        CCVFYOP='CMD(SAYPARM ,',             X
        MAXVIO=10000
```

## FLMCPYLB Macro

The FLMCPYLB macro identifies the name of a data set to be allocated by an occurrence of the FLMALLOC macro. FLMCPYLB macros that do not immediately follow either an FLMALLOC macro or another FLMCPYLB macro are ignored.

FLMCPYLB macros for build translators support only a limited set of the variables that are discussed in Chapter 6. SCLM Variables and Metavariables. For Build translators, the FLMCPYLB macro must be associated with an FLMALLOC macro that has its IOTYPE set to A or I. When the FLMCPYLB macro meets these conditions, the following variables are supported:

- @@FLMDBQ
- @@FLMSRF
- @@FLMPRJ
- @@FLMALT
- @@FLMUID
- @@FLMGRP. For build translators: The value for group in @@FLMGRP will be the group where the member referenced on the first SINC statement is found if the architecture definition being built is a CC or Generic architecture definition. If the architecture definition is an HL or LEC architecture definition, the value for @@FLMGRP will be the group where the build is taking place.
- @@FLMMBR. The @@FLMMBR variable is replaced with the name of the member being built. For a CC or a generic architecture definition, it is the name of the architecture definition.
- @@FLMTYP. The @@FLMTYP variable is replaced with the name of the type of the member being built. For a CC or generic architecture definition, it is the type of the architecture definition.
- @@FLMDSN

## Macro Format

```
FLMCPYLB dataset_name|NULLFILE
          [,VOL=volser]
```

## Parameters

*dataset\_name* | NULLFILE

Use the FLMCPYLB macro to allocate a data set to a ddname. Place the FLMCPYLB after an FLMALLOC macro with IOTYPE=I or A. See the IOTYPE parameter on the FLMALLOC macro for more information. For all other IOTYPES, SCLM ignores the data sets, unless MALLOC=Y. When MALLOC=Y, the IOTYPE can be either O or A. If you specify more than one FLMCPYLB, SCLM concatenates the data sets in the order they are specified. When you use them with IOTYPE=I, SCLM allocates the data sets after the type hierarchy libraries. SCLM can concatenate up to 123 data sets. Thus, when you use IOTYPE=I, ensure that the number of groups in the hierarchy (primary groups), plus the number of FLMCPYLB macros you specify does not exceed 123. If you concatenate more than 123 data sets, the project definition assembles without errors but using it produces unpredictable results.

You can specify partitioned data sets, sequential data sets, or members of fully-qualified partitioned data sets. Specify NULLFILE for the data set name for allocation of a dummy data set.

FLMCPYLB data set names can contain the following SCLM variables:

## FLMCPYLB Macro

- @@FLMDBQ
- @@FLMSRF
- @@FLMPRJ
- @@FLMALT
- @@FLMUID
- @@FLMGRP
- @@FLMGRB
- @@FLMMBR
- @@FLMTYP
- @@FLMDSN

The specified data set name, or the resulting data set name when SCLM variables are used, must meet all of the requirements of MVS data set names. The project definition allows up to 54 characters, including periods and parentheses, to support a data set with member name specification. A data set name containing SCLM variables that is longer than this will cause errors when the project definition is assembled, even if the substituted value meets all MVS naming conventions. A data set name that is allowed by the project definition, but does not meet MVS naming convention restrictions (for example, a data set name without the member specified that is more than 44 characters long), causes errors to occur during SCLM functions like Build.

### **,VOL=volser**

Specifies the serial number of an eligible direct access volume on which the data set is located. This allows reference to a data set that is either uncatalogued or that is located on a different volume than the catalog specifies. The default action, if not specified, is to use the volume in the dataset's catalog entry.

**Note:** If an SMS managed volume is specified, the system will override this specification with the volume in the catalog entry.

Use of the VOL keyword is mutually exclusive with specification of the NULLFILE dataset or with MALLOC=Y on the FLMALLOC macro.

## Example

The three data sets specified by the FLMCPYLB macro are allocated to the DDNAME ISPLoad.

```
FLMALLOC IOTYPE=A,DDNAME=ISPLoad
FLMCPYLB PROJ1.INTERNAL.LOAD
FLMCPYLB SYS2.ISPF.LOAD
FLMCPYLB SYS1.LINKLIB
```

The number of concatenated data sets and the names of the data sets are verified at run time.

If some includes are coming from system libraries instead of from the hierarchy, FLMCPYLB macros might be needed to allow the compiler or other build processor(s) to find those includes. The FLMCPYLB macros are needed if FLMSYSLB macros are used for the language and the language definition macro (FLMLANGL) has ALCSYSLB=N. In this case, an FLMCPYLB macro must be specified for each FLMSYSLB macro.

---

## FLMGROUP Macro

Use this macro to define each group in the project definition. This macro is required and can be used multiple times.

## Macro Format

```
name FLMGROUP
    [AC=(code1,code2,...)]
    [,ALTC=group_control_options]
    [,KEY=N|Y]
    [,PROMOTE=next_group]
```

## Parameters

### name

An 8-character group name.

### AC=(code1,code2,...)

A list of authorization codes and authorization groups that defines the authorization codes for the given group. If any item in the list is an authorization group, you must have previously defined it with the FLMAGRP macro.

The first authorization code you specify is the default authorization code used when a member is introduced to SCLM in this group. Each authorization code can be up to 8 characters and cannot contain commas. The maximum number of characters allowed for the authorization code list is 255, including commas and the delimiting parentheses.

If you omit this parameter, you cannot edit any members in this group. In addition, no editable members can be promoted into or out of this group.

### ,ALTC=group\_control\_options

Specifies an alternate set of control options to be used for this group. The name must match the name of an FLMALTC macro in the project definition. The data sets defined on the referenced FLMALTC macro are used to store the information for this group instead of the data sets specified on the FLMCNTRL macro. If this parameter is not specified, the group uses the data sets specified on the FLMCNTRL macro.

### ,KEY=N|Y

Defines whether the group is a key group or a non-key group. The default is Y. The KEY parameter does not apply to groups specified with the EXLIBID parameter.

### ,PROMOTE=next\_group

Defines the next higher group within the hierarchy for this group. If you do not specify it, SCLM does not allow any promotions out of this group.

## Example 1

Seven groups are defined for this project definition. The hierarchy consists of five layers. Groups DEV1 and DEV2 are defined as development groups because no groups promote to them. All groups except for the TEST group are defined as key groups. A list of authorization codes are assigned to each group. Group RELEASE is defined as the highest group in the hierarchy because it does not specify the PROMOTE parameter.

```
DEV1    FLMGROUP AC=(R6M0),KEY=Y,PROMOTE=STAGE1
DEV2    FLMGROUP AC=(R7M0),KEY=Y,PROMOTE=STAGE2
STAGE1  FLMGROUP AC=(R6M0,R7M0),KEY=Y,PROMOTE=INT
```

## FLMGROUP Macro

```
STAGE2  FLMGROUP AC=(R6M0,R7M0),KEY=Y,PROMOTE=INT
INT      FLMGROUP AC=(R6M0,R7M0),KEY=Y,PROMOTE=TEST
TEST     FLMGROUP AC=(R6M0,R7M0),KEY=N,PROMOTE=RELEASE
RELEASE  FLMGROUP AC=(R6M0),KEY=Y
```

## Example 2

In the following example:

- The ALTC parameter of the DEV group specifies that the control information defined by the FLMALTC macro DEVCNTL is used instead of the control information defined by the FLMCNTRL macro. The PDS data sets associated with this group have the naming convention SWDEV.PROJXYZ.DEV.type.
- The INT group uses the control information defined by the FLMCNTRL macro. The data set name used for SCLM-controlled PDS data sets defaults to @@FLMPRJ.@@FLMGRP.@@FLMTYP, resulting in a naming convention of PROJXYZ.INT.type for these data sets.
- The accounting database used by the REL group is PROJ2.ACCT.DATABASE as defined by the RELCNTL FLMALTC macro. The naming convention used for the PDS data sets is RELEASE.PROJ2.REL.type.

```
PROJXYZ  FLMABEG

          FLMCNTRL ACCT=PROJXYZ.ACCT.DATABASE

RELCNTL  FLMALTC ACCT=PROJ2.ACCT.DATABASE,                C
          DSNAME=RELEASE.PROJ2.@@FLMGRP.@@FLMTYP

DEVCNTL  FLMALTC ACCT=PROJDEV.ACCT.DATABASE,              C
          DSNAME=SWDEV.@@FLMPRJ.@@FLMGRP.@@FLMTYP

REL       FLMGROUP KEY=Y,ALTC=RELCNTL
INT       FLMGROUP KEY=Y,PROMOTE=REL
DEV       FLMGROUP KEY=Y,PROMOTE=INT,ALTC=DEVCNTL

          FLMAEND
```

---

## FLMINCLS Macro

Use this macro to associate include sets with types in the project hierarchy. This association is used to determine the location of include members within the project. Parsers may be written to associate an include set with each include found in a source member. This macro indicates to the build function where include sets can be found and which data sets to allocate for input to build translators. This macro is part of a language definition. The FLMINCLS macro must follow the FLMLANG macro for the language definition.

The FLMSYSLB macro is used to specify data sets outside the project that contain includes. The INCLS parameter value on the FLMSYSLB macro associates the name of an include set with the FLMSYSLB libraries. The search for an include member will first take place in the include set and then in the associated FLMSYSLB.

The default include set is associated with an FLMSYSLB that has no INCLS parameter. The default include set is specified by an FLMINCLS that has no name parameter. Only one default FLMINCLS may be specified for each language definition. SCLM will generate a default include set if one is not specified.

Use the INCLS parameter on an FLMALLOC macro of IOTYPE=I to associate an FLMINCLS macro with an FLMALLOC macro. If no name is specified on the

FLMINCLS macro, the macro is for the default include set. The default include set is associated with FLMALLOC macros of IOTYPE=I where no INCLS parameter is specified.

If an FLMINCLS macro is specified for the default include set, at least one FLMALLOC macro must reference the default include set (by specifying IOTYPE=I and no INCLS parameter). If there is no FLMINCLS macro in the language, an FLMALLOC macro for the default include set is optional.

SCLM ensures that each language definition includes a default include set and a COMPOOL include set. If the language definition does not include macros to define these two include sets, the following definitions are generated:

```

FLMINCLS TYPES=(@@FLMTYP,@@FLMETP)
COMPOOL FLMINCLS TYPES=(@@FLMCRF @@FLMECR)

```

## Macro Format

name FLMINCLS

[SAMEAS=*flmincls\_name* | TYPES=(*list\_of\_types*)]

[CROSLANG=Y|N]

## Parameters

### name

The name of the include set that is being defined in this macro. An include set is associated with FLMSYSLB or FLMALLOC when the name matches the value of an INCLS parameter. In addition, the name may be the name of an include set returned by a parser for the language that includes this FLMINCLS macro. Each include set name can only be used once per language definition.

To specify the default include set, leave this parameter blank.

### SAMEAS=*flmincls\_name*

The name of another FLMINCLS macro that contains the list of types to search. If you use this parameter, the include set defined by this macro has the same list of types as the include set listed on the SAMEAS parameter. You cannot reference the default include set by specifying SAMEAS= with a blank.

### TYPES=(*list\_of\_types*)

A list of the types that contain the includes for the include set. Build searches these types in the order given on this parameter. The hierarchies for each type are concatenated for use by all FLMALLOC macros that reference this FLMINCLS macro.

Duplicate types are not removed from the list.

Two SCLM variables can be used on this parameter: @@FLMTYP and @@FLMETP. The value of @@FLMTYP is the type of the member on the first SINC statement in an architecture definition or the type of the member if a single member is being built. The value of @@FLMETP is the extended type of that member. (See the EXTEND parameter on the FLMTYPE macro).

The value that will be substituted into the @@FLMCRF variable will be the DFLTCRF type. The value that will be substituted into the @@FLMECR variable for include set definitions will be the extended type of the DFLTCRF type. If there is no extended type for the DFLTCRF type, the @@FLMECR variable will be ignored.

## FLMINCLS Macro

### CROSLANG=Y|N

The CROSLANG parameter indicates whether SCLM processes the includes of an included member when the included member has a different language from the source member. Y indicates that includes are processed even if language boundaries are crossed. N indicates that only the includes of a member of the same language are processed. The value of the CROSLANG parameter is not affected by the SAMEAS parameter. The default for CROSLANG is Y.

Following is an example of how includes are processed given the two possible values for this parameter:

Member	Includes
SCRIPT1 language=SCRIPT	COBOL1 language=COBOL
COBOL1 language=COBOL	INCLUDE1 language=COBOL
INCLUDE1 language=COBOL	none

- If CROSLANG=Y when SCRIPT1 is built, the build processor checks the dates and times of COBOL1 and INCLUDE1 and puts them in the build map.
- If CROSLANG=N when SCRIPT1 is built, the build processor checks the dates and times of COBOL1 and puts them in the build map. The dates and times of INCLUDE1 are not processed.

**Note:** If both the SAMEAS and TYPES parameters are omitted for an FLMINCLS macro, no types are searched for that include set. This can be used when includes are only in data sets specified by FLMSYSLB macros or no includes of that type are allowed. Even if no parameters are specified on an FLMINCLS macro, it must be referenced by at least one FLMALLOC macro.

### Example 1

The following example shows how to define where the includes in the default include set are found. It indicates that the INCLUDE type is to be searched first, followed by the source type of the member being processed, and finally by the extended type of the source member if there is one. The types listed on this macro are used for all IOTYPE=I FLMALLOC macros where no INCLS parameter is specified.

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

### Example 2

The following example shows how to define where the includes in the MACRO and COPY include sets are found. It indicates that the MACRO type is the only type to be searched. Both the MACRO and COPY include sets are referenced by IOTYPE=I FLMALLOC macros. The IOTYPE=I FLMALLOC macros specify the allocation for the include hierarchies of the build translators. The MACRO FLMINCLS does not allow processing of includes across language boundaries. The COPY FLMINCLS processes the includes because the default value (Y) was not overridden for the CROSLANG parameter.

```
MACRO    FLMINCLS TYPES=(MACRO),CROSLANG=N
COPY     FLMINCLS SAMEAS=MACRO
```

### Example 3

The following example shows how to use different sequences of types for locating includes. This may be useful in situations in which includes in several different types have the same name.

```

        FLMLANGL    LANG=ABC,VERSION=1
*
* SEQUENCES OF TYPES FOR LOCATING INCLUDES
*
DBRM      FLMINCLS TYPES=(DBRMTYPE,@@FLMTYP,@@FLMETP)
SPECIAL  FLMINCLS TYPES=(COPYBOOK,SOURCE,MACRO,TOOLS)
*
* PARSER TRANSLATOR
*
        FLMTRNSL   CALLNAM='ABC PARSE',                C
                    FUNCTN=PARSE,                      C
                    COMPILE=FLMLPGEN,                 C
                    PORDER=1,                         C
                    GOODRC=0,                         C
                    OPTIONS=(SOURCEDD=SOURCE,         C
                              STATINFO=@@FLMSTP,     C
                              LISTINFO=@@FLMLIS,     C
                              LISTSIZE=@@FLMSIZ,     C
                              LANG=T)                C
*
        (* SOURCE      *)
        FLMALLOC   IOTYPE=A,DDNAME=SOURCE
        FLMCPYLB   @@FLMPRJ.@@FLMGRP.@@FLMTYP(@@FLMMBR)
*
* BUILD TRANSLATORS
*
        FLMTRNSL   CALLNAM='USE DEFAULT',                C
                    FUNCTN=BUILD,                      C
                    COMPILE=USEDFLT,                  C
                    VERSION=1.0,                     C
                    GOODRC=0,                         C
                    PORDER=1
*
* DDNAME ALLOCATIONS
* SYSLIB WILL USE DEFAULT OF THE TYPE FOR THE SINC MEMBER AND THE
* EXTENT AS DEFINED IN THE PROJECT DEFINITION
*
        FLMALLOC   IOTYPE=I,DDNAME=SYSLIB,KEYREF=SINC
        FLMALLOC   IOTYPE=S,DDNAME=SYSIN,KEYREF=SINC,RECNUM=2000
        FLMALLOC   IOTYPE=O,DDNAME=SYSLIN,KEYREF=OBJ,RECNUM=5000,DFLTTP=OBJ
*
*
        FLMTRNSL   CALLNAM='LOOK AT DBRM',                C
                    FUNCTN=BUILD,                      C
                    COMPILE=LOOKDBRM,                 C
                    VERSION=1.0,                     C
                    GOODRC=0,                         C
                    PORDER=1
*
* DDNAME ALLOCATIONS
* SYSLIB WILL USE DBRMTYPE FOLLOWED BY THE TYPE FOR THE SINC MEMBER AND
* THEN THE EXTENT OF THE SINC MEMBER TYPE AS DEFINED IN THE PROJECT
* DEFINITION
*
        FLMALLOC   IOTYPE=I,DDNAME=SYSLIB,KEYREF=SINC,INCLS=DBRM
        FLMALLOC   IOTYPE=S,DDNAME=SYSIN,KEYREF=SINC,RECNUM=2000
        FLMALLOC   IOTYPE=O,DDNAME=SYSLIN,KEYREF=OBJ,RECNUM=5000,DFLTTP=OBJ
*
*
        FLMTRNSL   CALLNAM='USE SPECIAL',                C
                    FUNCTN=BUILD,                      C
                    COMPILE=IKJSPECL,                 C

```

## FLMINCLS Macro

```
                                VERSION=1.0,                                C
                                GOODRC=0,                                  C
                                PORDER=1
*
* DDNAME ALLOCATIONS
* SYSLIB WILL USE COPYBOOK, SOURCE, MACRO, and TOOLS
*
FLMALLOC IOTYPE=I,DDNAME=SYSLIB,KEYREF=SINC,INCLS=SPECIAL
FLMALLOC IOTYPE=S,DDNAME=SYSIN,KEYREF=SINC,RECNUM=2000
FLMALLOC IOTYPE=O,DDNAME=SYSLIN,KEYREF=OBJ,RECNUM=5000,DFLTTP=OBJ
```

---

## FLMLANGL Macro

Use this macro to define a language to SCLM. Specify the name of the language and processing characteristics using the keywords supported by this macro. Specify the translators to be invoked for this language by using the FLMTRNSL macro after FLMLANGL.

The order in which data sets of various types are to be allocated for finding includes may be specified by using one or more FLMINCLS macros. The FLMALLOC macros following each FLMTRNSL macro are associated with FLMINCLS macros by use of the INCLS parameter.

## Macro Format

```
FLMLANGL LANG=language
          [,ALCSYSLB=N|Y]
          [,ARCH=N|Y]
          [,BUFSIZE=buffer_size|100]
          [,CANEDIT=Y|N]
          [,CHKSYSLB=PARSE|BUILD|IGNORE]
          [,COMPOOL=N|Y]
          [,DEPPRC=Y|N]
          [,DFLTCRF=default_CREF_reference]
          [,DFLTSRF=default_source_reference]
          [,SCOPE=LIMITED|NORMAL|SUBUNIT|EXTENDED]
          [,VERSION=language_version]
          [,MBRLMT=0]
```

## Parameters

**LANG**=*language*

A user-specified pseudonym for a language. It can be up to 8 characters. It is stored with the accounting information of editable members. Specify this name when you first define a member to SCLM.

**,ALCSYSLB**=N|Y

Indicates whether or not data sets on FLMSYSLB macros are allocated automatically for IOTYPE=I allocations (see FLMALLOC on page “FLMALLOC Macro” on page 130). If ALCSYSLB=N, use FLMCPYLB macros for each

FLMSYSLB data set on IOTYPE=I allocations. If ALCSYSLB=Y, FLMSYSLB data sets are allocated by build following the allocation of the data sets from the project.

FLMSYSLB data sets are concatenated to the IOTYPE=I allocations for FLMALLOC macros when both the FLMALLOC and FLMSYSLB macros specify the INCLS parameter with the same value. If no INCLS parameter is specified on the FLMSYSLB macro, the FLMSYSLB data sets are concatenated to the FLMALLOC macros with IOTYPE=I and no INCLS parameter.

**,ARCH=N|Y**

Indicates whether a member parsed in this language is an architecture member. The default is N.

**,BUFSIZE=*buffer\_size* | 100**

The number of \$list\_info records SCLM allocates for a parse, verify, build, copy, or purge translator. The translator returns dependency information in the allocated memory. The default size is 100. The *buffer\_size* must be large enough to accommodate the maximum number of entries returned in \$list\_info by any translator including one entry for the END record which is always required in a \$list\_info buffer. SCLM requires one record for each include, change code, user data record, or external dependency the translator returns.

**,CANEDIT=Y|N**

Indicates whether the language can be assigned to editable members. You should specify language definitions for linkage editors with CANEDIT=N. The default is Y.

**,CHKSYSLB=PARSE | BUILD | IGNORE**

Indicates when SCLM will check the FLMSYSLB data sets to determine if an include is to be tracked. If CHKSYSLB=PARSE, FLMSYSLB data sets are checked at parse time. Any includes not found in the hierarchy that are in FLMSYSLB data sets are not recorded in the accounting record. If CHKSYSLB=BUILD, FLMSYSLB data sets are checked at build time. Any includes not found in the hierarchy that are in FLMSYSLB data sets are recorded in the accounting record but not in the build map. If CHKSYSLB=IGNORE, any includes not found in the hierarchy at build time are ignored. They are recorded in the accounting record, but are not recorded in the build map. The build translator must determine if includes are missing and generate a return code indicating that the member could not be built.

Use IGNORE for workstation builds when the syslib data sets do not reside in a location that SCLM can check.

IGNORE can also be used to significantly improve performance when your system libraries are fairly stable. By specifying IGNORE, you bypass the overhead of checking all system libraries at either parse or build time. The performance improvement can be significant, particularly in cases where a large number of system libraries is specified in the language definition. The trade-off is that you invoke a translator that will fail when an include is not found. If your system libraries are fairly stable, it might be better to invoke the translator when occasionally an include might be missing, than to search all of the system libraries each time a member is either parsed or built.

**,COMPOOL=N|Y**

Indicates whether a compool output is required. If COMPOOL=Y is specified, SCLM verifies that a compool output is generated and saved in the hierarchy. SCLM issues a warning message if there is no output identified by the COMP architecture definition keyword.

## FLMLANGL Macro

### **,DEPPRCS=Y|N**

Indicates whether components depending on the member being built are rebuilt if some outputs from the translator were not saved for this member. The default is Y.

### **,DFLTCRF=*default\_CREF\_reference***

Identifies the type that is substituted into the @@FLMCRF variable for include-set definitions. The @@FLMCRF variable can be used in the list of types to search for includes. The CREF statement architecture statement can be used to override this value. If both the CREF statement and DFLTCRF parameter are omitted the @@FLMCRF variable is ignored.

The value that is substituted into the @@FLMECR variable for include-set definitions is the extended type of the DFLTCRF type. If there is no extended type for the DFLTCRF type, the @@FLMECR variable is ignored.

### **,DFLTSRF=*default\_source\_reference***

A type name that can be used to allocate a hierarchical view. This hierarchical view is typically used by the translator to resolve references to SCLM hierarchy members. This parameter has no effect unless an FLMALLOC macro with IOTYPE=I and KEYREF=SREF is used for the language. SCLM ignores this parameter during a build if a CC, Generic, or LEC architecture definition is used to build the source member.

### **,SCOPE=LIMITED|NORMAL|SUBUNIT|EXTENDED**

Indicates the minimum scope allowed. SCLM compares this parameter with the mode specified as input to build and promote functions to allow or disallow processing. The input mode must be of equal or greater value than the language scope. Valid scope values, in ascending order, are LIMITED, NORMAL, SUBUNIT, and EXTENDED. The default is NORMAL.

### **,VERSION=*language\_version***

The 8-character version name associated with this language. Altering this parameter causes all source members under this language to be rebuilt. If you do not specify it, SCLM sets this parameter to blank.

### **,MBRLMT=0**

Indicates the maximum number of source members that can be present in any input list presented to a translator. SCLM does not exceed the specified MBRLMT value. If you specify MBRLMT=0, there is no limit on the number of source members.

## Example 1

The language definition for PASCAL is defined.

```
FLMLANGL LANG=PASCAL,VERSION=1.0,ALCSYSLB=Y
```

---

## FLMLRBLD Macro

The FLMLRBLD macro causes members with a particular language to be rebuilt whenever they are promoted into particular groups. Rebuilding is often necessary when processing changes due to FLMTOPTS or FLMTCOND. The FLMLRBLD macro is only valid within a language definition; it must follow an FLMLANGL.

During the promotion of a member whose language requests a rebuild with the FLMLRBLD macro for that particular group, the build map is not copied during the promote. After the promote completes, the build function is invoked using the 'to group'. The build is conditional and is invoked against the same member that was promoted. Because the build maps will be missing for members having that

language, those members, and any dependent members, will be rebuilt. All other members will have the build maps copied, and will not be rebuilt during the conditional build.

If the promote copy succeeds, then the build will take place.

There can be multiple FLMLRBLD macros for each language.

## Macro Format

```
FLMLRBLD
  [GROUP=group_list]
```

## Parameters

### Group=group\_list

This parameter specifies the groups at which promoted members will be rebuilt. After a member with the language given on the previous FLMLANGL macro is promoted to one of the listed groups, the member is conditionally rebuilt. The group list must be enclosed in parenthesis or single quotes, with a comma and no spaces between the group names. The list of groups is not checked for validity when the project definition is assembled, or during build. This allows alternate project definitions to function without requiring that all groups be defined in the alternate project definition.

## Examples

The following example shows a part of a language definition of a language that changes translator options at group TEST. The FLMLRBLD macro specifies that members with language COMPLANG will be automatically rebuilt after a promotion.

```
FLMLANGL LANG=COMPLANG,VERSION=1.0,ALCSYSLB=Y
FLMLRBLD GROUP=(PROD)
FLMTRNSL CALLNAM='Compile', X
FUNCTN=BUILD, X
COMPILE=EXAMPLE, X
OPTIONS='ANSI'

FLMTOPTS OPTIONS='ANSI,NODEBUG,OPTIMIZE', X
GROUP=(PROD),ACTION=REPLACE
```

---

## FLMSYSLB Macro

Use this macro to define a set of system macro or include data sets for an include set in a language. The data sets defined by FLMSYSLB contain members that are referenced by SCLM members. Whether or not these include dependencies are tracked is determined by the CHKSYSLB parameter of the language. These data sets also can be allocated for the build translator(s) by using the ALCSYSLB parameter of the language.

Different sequences of data sets may be specified by using the INCLS parameter. FLMSYSLB macros with the INCLS parameter will be used in conjunction with FLMALLOC macros that have IOTYPE=I and an INCLS parameter with the same value. The value of the INCLS parameter is the name of an FLMINCLS macro in the language definition.

## FLMSYSLB Macro

### Macro Format

```
[language] FLMSYSLB dataset_name  
[,INCLS=include set_name]  
[,VOL=volser]
```

### Parameters

#### language

An 8-character language name. The language must be the same name as the language specified in the **LANG** field on the **FLMLANGL** macro. To specify multiple data sets for a language, omit the language on all but the first data set.

#### dataset\_name

The partitioned data set or member of a fully-qualified partitioned data set containing macros or includes from outside the project. The data set name must meet all of the requirements specified by the MVS data set naming conventions. The project definition allows up to 54 characters, including periods and parentheses, to support the specification of a member name. If the data set name is too long (for example, more than 44 characters for a data set name without a member specified), or it does not meet the MVS data set naming conventions, then errors occur during SCLM functions (for example, Parse or Build). The data sets are searched and allocated in the order that they occur in the project definition.

#### ,INCLS=include\_set\_name

This refers to the include-set name on an **FLMINCLS** macro. When searching for includes, SCLM first checks the types specified on the **FLMINCLS** macro, followed by the data set on this and other **FLMSYSLB** macros with the same include-set name. If no **INCLS** parameter is specified, this **FLMSYSLB** macro is used for the default include set. All of the **FLMSYSLB** statements for an include set must be specified together.

#### ,VOL=volser

Specifies the serial number of an eligible direct access volume on which the data set is located. This allows reference to a data set that is either uncatalogued or that is located on a different volume than the catalog specifies. The default action, if not specified, is to use the volume in the dataset's catalog entry.

**Note:** If an SMS managed volume is specified, the system will override this specification with the volume in the catalog entry.

### Example

The following example shows the **FLMSYSLB** macros that might be included in the project definition for a language that has includes in 3 different include sets.

```
DBAPPL      FLMSYSLB  SYS1.COMPILER.INCLUDES  
            FLMSYSLB  SYS1.DATABASE.INCLUDES,INCLS=DATABASE  
            FLMSYSLB  SYS1.TRANSACTION.INCLUDES,INCLS=DATABASE  
            FLMSYSLB  APPL.REUSE.INCLUDES,INCLS=REUSE
```

In this example the includes for members of language **DBAPPL** will first find their members in the project hierarchy. If the includes are not found in the project hierarchy the **FLMSYSLB** data sets will be searched. Only the **FLMSYSLB** data sets that are associated with the same include set as the include will be searched for the include.

For example, in the DBAPPL language definition:

1. FLMALLOC with IOTYPE=I and no INCLS parameter will be associated with SYS1.COMPILER.INCLUDES
2. FLMALLOC with IOTYPE=I and INCLS=DATABASE will be associated with SYS1.DATABASE.INCLUDES concatenated with SYS1.TRANSACTION.INCLUDES
3. FLMALLOC with IOTYPE=I and INCLS=REUSE will be associated with APPL.REUSE.INCLUDES
4. FLMALLOC with IOTYPE=I and INCLS=XXXXX will not have an associated FLMSYSLB since there are no FLMSYSLB macros associated with language DBAPPL that have an INCLS parameter with the value XXXXX

Includes in data sets outside of the project definition can either be tracked in the accounting record of the member that includes them or not tracked at all.

To track external includes in the accounting record:

1. Specify CHKSYSLB=BUILD or IGNORE in the language definition.
2. Specify an FLMSYSLB for each data set containing included members.
3. Either specify ALCSYSLB=Y in the language definition, or specify each of the data sets from FLMSYSLB macros on FLMCPYLB macros for the appropriate ddnames.

When CHKSYSLB is BUILD, SCLM checks the FLMSYSLB data sets at build time. When CHKSYSLB is IGNORE, the build translator determines if includes are missing. In either case, any includes not found in the hierarchy are recorded in the accounting record when the member is parsed.

To not track external includes at all:

1. Specify CHKSYSLB=PARSE in the language definition (the default)
2. Specify an FLMSYSLB for each data set containing included members
3. Either specify ALCSYSLB=Y in the language definition, or specify each of the data sets from FLMSYSLB macros on FLMCPYLB macros for the appropriate ddnames.

When CHKSYSLB is PARSE, SCLM verifies at parse time that any includes that are not found in the project hierarchy can be found in the FLMSYSLB data sets. The includes found in the FLMSYSLB data sets are not recorded as includes in the accounting record.

---

## **FLMTCOND Macro**

The FLMTCOND macro provides a means of running or skipping BUILD translators based upon the group at which the BUILD takes place and return codes from previous BUILD translators in the same language definition. The use of FLMTCOND is similar to the use of the COND keyword parameter on a JCL EXEC statement. This similarity is restricted by the requirement that multiple uses of FLMTCOND in a language definition require each corresponding FLMTRNSL macro to have identical output KEYREF and DFLTTYP keyword values on the FLMALLOC statements.

The FLMTCOND macro can be used to specify a group, combinations of return codes from previous BUILD translators in the same language definition, or both in order to:

- Run one of two BUILD translators

## FLMTCOND Macro

- Run or skip a BUILD translator only under certain conditions
- Run or skip several BUILD translators that have the same outputs

An FLMTCOND macro must follow an FLMTRNSL macro with FUNCTN=BUILD. Only one FLMTCOND macro can be specified for each FLMTRNSL.

The GROUP and NOTGROUP parameters are mutually exclusive. If neither GROUP nor NOTGROUP is specified, the relations list and action applies to all groups.

FLMTCOND can be used with the GROUP or NOTGROUP parameters to provide an IF-THEN-ELSE effect in which only one of two translators is used. The NOTGROUP keyword can provide flexibility in altering the hierarchy without similar alterations in the language definitions.

**Note:** Use of the FLMTCOND statement does not cause a recompile when a member is promoted to another group, it only specifies actions to be taken *if* a build is performed at the new group. To cause a rebuild to occur automatically, add an FLMLRBLD statement for the language.

## Macro Format

```
FLMTCOND
  [ GROUP=group_list|NOTGROUP=group_list]
  [,WHEN=relations_list]
  [,ACTION=RUN|SKIP]
```

The logic of the GROUP, NOTGROUP, WHEN, and ACTION is shown in the following illustration:

```
For specifying GROUP keyword:
IF the BUILD group is in the group_list
  AND
WHEN at least one relation is TRUE THEN
  DO ACTION
ELSE
  DO OTHER ACTION
```

```
For specifying NOTGROUP keyword:
IF the BUILD group is NOT in the group_list
  AND
WHEN at least one relation is TRUE THEN
  DO ACTION
ELSE
  DO OTHER ACTION
```

```
DEFAULTS:
GROUP = ALL GROUPS
WHEN = TRUE
ACTION = RUN
```

## Parameters

### GROUP=group\_list

This parameter specifies the groups where the relations list and action are used.

The group list must be enclosed in parentheses or single quotes with a comma and no spaces between the group names. The list of groups is not checked for validity when the project definition is assembled or during build. This allows

alternate project definitions to function without requiring that all groups be defined in the alternate project definition.

The other ACTION is taken for build groups not in the group\_list.

GROUP=() or GROUP=( ) specifies an empty group list.

**NOTGROUP=group\_list**

Use this parameter to specify groups to which you do not want the relations list and action applied.

NOTGROUP=() or NOTGROUP=( ) specifies an empty group list.

The format of the group\_list is the same as for the GROUP parameter.

**,WHEN=relations\_list**

This parameter specifies the conditions under which the translator is run. The default is TRUE.

The relations\_list is  $(s1,r1,v1)$  or  $((s1,r1,v1),\dots,(sn,rn,vn))$  where:

- *si* is the translator label for a previous FLMTRNSL macro of a BUILD translator in the same language definition. An asterisk (\*) can be used to match the previous FLMTRNSL (with or without a translator label) in the language definition for the BUILD translator that last executed. Using an asterisk allows you to refer back at run time to the BUILD translator that was last executed in the language definition at this point. There is no default.
- *ri* is a standard relation such as EQ, NE, LT, GT, LE, or GE. There is no default.
- *vi* is an unsigned integer with a maximum value of 999999999. This relation is compared with the return code from a previous Build translator identified by *si* in the language definition. There is no default.

At run time, SCLM stops examining the relations in a list as soon as a TRUE relation is found. Incorrect labels in relations that follow a TRUE relation do not result in an error message. The relations in a list can be viewed as boolean values connected by a boolean OR. Build translators that have not executed are ignored for *si* = \*.

SCLM stops examining previous Build translators for a relation when the label *si* is located even if the Build translator did not execute. The relation is evaluated as FALSE for Build translators that did not execute.

**,ACTION=RUN|SKIP**

This parameter specifies the action to take at run time.

The decision to RUN or SKIP the translator depends upon the build group, the GROUP|NOTGROUP parameter, and the WHEN parameter.

All FLMTRNSL macros in a language definition that also use FLMTCOND with a WHEN keyword must use the same FLMALLOC KEYREF and DFLTTYP keyword values for all output allocations (KEYREF is OBJ, OUTx, COMP, LIST, LOAD or LMAP). Use of the same keywords results in an architecture definition that is correct for all possible return codes at run time.

When the default architecture definition is constructed, SCLM does not know what the return codes will be at run time. The following assumptions are made:

- The WHEN keyword value contains a TRUE relation.
- The FLMTRNSL macros will not be executed.

## FLMTCOND Macro

### Examples

Code example	Result
not specified	Run the translator for all groups.
FLMTCOND	Run the translator for all groups. This is legal, but has the same effect as not specifying FLMTCOND.
FLMTCOND ACTION=SKIP	Skip the translator for all groups.
FLMTCOND WHEN=(STEP1,EQ,4)	Run the translator for all groups if the return code from the previous translator with label STEP1 equaled 4.  Skip the translator for all groups if the return code from the previous translator with label STEP1 did not equal 4.
FLMTCOND WHEN=(STEP1,EQ,4),ACTION=SKIP	Run the translator for all groups if the return code from the previous translator with label STEP1 did not equal 4.  Skip the translator for all groups if the return code from the previous translator with label STEP1 equaled 4.
FLMTCOND NOTGROUP=(FVT,SVT,PROD)	Run the translator if the group is not FVT, SVT, or PROD.  Skip the translator if the group is FVT, SVT, or PROD.
FLMTCOND NOTGROUP=(FVT,SVT,PROD), ACTION=SKIP	Run the translator if the group is FVT, SVT, or PROD.  Skip the translator if the group is not FVT, SVT, or PROD.
FLMTCOND NOTGROUP=(FVT,SVT,PROD), WHEN=(STEP1,EQ,4)	Run the translator if the group is not FVT, SVT, or PROD and the return code from the previous translator with label STEP1 equaled 4.  Skip the translator if the return code from the previous translator with label STEP1 did not equal 4.  Skip the translator if the group is FVT, SVT, or PROD.
FLMTCOND NOTGROUP=(FVT,SVT,PROD), WHEN=(STEP1,EQ,4),ACTION=SKIP	Run the translator if the return code from the previous translator with label STEP1 did not equal 4 or if the group is FVT, SVT, or PROD.  Skip the translator if the group is not FVT, SVT, or PROD and the return code from the previous translator with label STEP1 equaled 4.
FLMTCOND GROUP=(FVT,SVT,PROD)	Run the translator if the group is FVT, SVT, or PROD.  Skip the translator if the group is not FVT, SVT, or PROD.
FLMTCOND GROUP=(FVT,SVT,PROD), ACTION=SKIP	Run the translator if the group is not FVT, SVT, or PROD.  Skip the translator if the group is FVT, SVT, or PROD.
FLMTCOND GROUP=(FVT,SVT,PROD), WHEN=(STEP1,EQ,4)	Run the translator if the group is FVT, SVT, or PROD and the return code from the previous translator with label STEP1 equaled 4.  Skip the translator if the return code from the previous translator with label STEP1 did not equal 4.  Skip the translator if the group is not FVT, SVT, or PROD.

Code example	Result
FLMTCND GROUP=(FVT,SVT,PROD), WHEN=(STEP1,EQ,4),ACTION=SKIP	Run the translator if the return code from the previous translator with label STEP1 did not equal 4 or if the group is not FVT, SVT, or PROD.  Skip the translator if the group is FVT, SVT, or PROD and the return code from the previous translator with label STEP1 equaled 4.

## FLMTOPTS Macro

The FLMTOPTS macro is used 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. Multiple FLMTOPTS macros can be specified for each FLMTRNSL in which case the FLMTOPTS will be applied in the order they appear in the project definition. The GROUP and NOTGROUP parameters are mutually exclusive. If neither GROUP nor NOTGROUP is specified, the options list and action will apply to all groups.

**Note:** Use of the FLMTOPTS statement does not cause a recompile when a member is promoted to another group, it only specifies the options to be used *if* a build is performed at the new group. To cause a rebuild to occur automatically, add an FLMLRBLD statement for the language.

## Macro Format

```
FLMTOPTS OPTIONS=options_list
[,GROUP=group_list|NOTGROUP=group_list]
[,ACTION=APPEND|REPLACE]
```

## Parameters

### OPTIONS=options\_list

This parameter specifies the options that are added to the end of the existing options or replace the existing options. See the OPTIONS parameter on the FLMTRNSL macro for more information on the content and format of options lists.

### ,GROUP=group\_list

This parameter specifies the groups where the options list and action are to be applied. If the build group is found in the list then the translator options list will be updated. The group list must be enclosed in parentheses or single quotes with a comma and no spaces between the group names. The list of groups is not checked for validity when the project definition is assembled or during build. This allows alternate project definitions to function without requiring that all groups be defined in the alternate project definition.

### ,NOTGROUP=group\_list

This parameter specifies the groups where the options list and action are **not** to be applied. If the build group is found in the list then the translator options list will not be updated. The format of the group list is the same as for the GROUP parameter.

### ,ACTION=APPEND|REPLACE

This parameter specifies how the translator options will be updated.

## FLMTOPTS Macro

If APPEND (the default value) is specified the options list will be appended to the end of the existing options list for the translator. No commas or spaces will be added between the existing options list and the options list from the FLMTOPTS macro.

If REPLACE is specified the existing options list will be replaced with the options list from the FLMTOPTS macro.

## Examples

The following example shows a part of a language definition that contains an FLMTRNSL followed by multiple FLMTOPTS macros. The options passed to the translator will vary by the group where the build is taking place. If options were specified in an architecture definition member, they would be added to the end of the options shown here.

Build Group	Options passed to translator
PROD	ANSI,NODEBUG,OPTIMIZE,LIST
TEST	ANSI,OPTIMIZE,LIST
INT	ANSI,LIST
PERFTEST	ANSI,OPTIMIZE,TIMER,LIST
others	ANSI,DEBUG,NOOPTIMIZE,LIST
	FLMTRNSL CALLNAM='Compile', X
	FUNCTN=BUILD, X
	COMPILE=EXAMPLE, X
	OPTIONS='ANSI'
*	
	FLMTOPTS OPTIONS=',DEBUG,NOOPTIMIZE', X
	NOTGROUP=(PROD,TEST,INT),ACTION=APPEND
*	
	FLMTOPTS OPTIONS='ANSI,OPTIMIZE,TIMER', X
	GROUP=(PERFTEST),ACTION=REPLACE
*	
	FLMTOPTS OPTIONS='ANSI,NODEBUG,OPTIMIZE', X
	GROUP=(PROD),ACTION=REPLACE
*	
	FLMTOPTS OPTIONS=',OPTIMIZE', X
	GROUP=(TEST)
*	
	FLMTOPTS OPTIONS=',LIST'

---

## FLMTRNSL Macro

This macro serves a function similar to JCL EXECUTE (EXEC) statements in your procedure libraries. Several keyword parameters in this macro contain data identical to your procedures.

Use this macro once for each translator to be invoked for a language. Specify the translator load module name, translator load data set name, version of the translator, and translator options using this macro's parameters.

## Macro Format

```
[translator label] FLMTRNSL CALLNAM='call_name'  
  
    [,FUNCTN=PARSE|VERIFY|BUILD|COPY|PURGE]  
  
    ,COMPILE=translator_name
```

```
[,DSNAME=translator_dataset_name]
[,GOODRC=good_return_code|0]
[,NOSVEXT=no_save_external_rc|0]
[,OPTFLAG=N|Y]
[,OPTIONS=option_list]
[,PARMKWD=parameter_keyword]
[,PDSDATA=Y|N]
[,PORDE=0|1|2|3]
[,VERSION=translator_version]
[,CALLMETH=ATTACH|LINK|TSOLNK|ISPLNK]
[,TASKLIB=translator_ddname]
[,INPLIST=N|Y]
[,MBRRC=maximum_good_return_code]
```

## Parameters

### *translator label*

A 1- to 8-character string containing no blanks or commas. The translator label is used by the FLMTCOND macro to identify build translators in a language definition to examine their return codes at run time for conditional execution of build translators.

### **CALLNAM='call\_name'**

The name of the translator with a maximum of 16 characters. This name appears in SCLM messages along with translator return codes. If you want imbedded blanks in the call name, surround the string with single quotes.

### **,FUNCTN=PARSE|VERIFY|BUILD|COPY|PURGE**

Identifies the function that the translator performs. The default is PARSE.

- A parse translator gathers statistics and dependencies. Parse translators run during migration, when the member is saved in an edit session, or when the SAVE or PARSE service is called. A parse translator can also be used to define user data and change codes for the member.
- A verify translator can be used to perform validation in addition to default SCLM validation. The verify translator can be used to check the change codes or user data defined for members. Another example could be verification of data that is related to an SCLM-controlled member but is not under SCLM control itself. Verify translators run during build and promote verification.

For builds, SCLM invokes a verify translator to verify inputs to build translators. For example, when an LEC architecture definition is being built, the source member is verified prior to compiling and the object member is verified prior to linking.

For promotes, SCLM invokes a verify translator to verify build inputs and outputs. For example, when an LEC architecture definition is being promoted, the source, object, and load members are verified prior to the promote copy phase.

## FLMTRNSL Macro

- A build translator can assemble, compile, link, or otherwise process a member so that the outputs have different formats than the inputs. For example, building a COBOL source program generates a listing and an object module.
- A copy translator is invoked when Promote copies an SCLM-controlled member to the next group in the hierarchy. Copy translators are invoked before Promote copies the SCLM-controlled member. If the copy translators for a member fail, Promote does not attempt to copy the controlled member. Copy translators can be used to copy data that is related to an SCLM-controlled member but is not under SCLM control itself.
- Purge translators can be used to purge data that is related to an SCLM-controlled member but is not under SCLM control itself. Purge translators are invoked whenever SCLM performs a delete operation on an SCLM-controlled member during build or promote.

### **,COMPILE=***translator\_name*

The entry point name on the translator load module (for example, the name of the member containing the translator) or REXX exec being invoked.

For CALLMETH of ATTACH, LINK, and TSOLNK, this is a REXX exec, CLIST, or entry point to a load module. For a CALLMETH of ISPLNK, this must have a value of SELECT.

### **,DSNAME=***translator\_dataset\_name*

The name of the data set containing the translator load module (COMPILE parameter) or REXX exec being invoked. The data set name parameter is not required with the translator load module that resides in the system concatenation. Use the TASKLIB parameter to specify additional libraries to be searched. SCLM looks in the data set specified by the DSNAME parameter first, followed by the data sets allocated to the TASKLIB ddname, if specified, and then follows the normal MVS search order. The data set name can be up to 44 characters.

### **,GOODRC=***good\_return\_code*10

Definition of an acceptable return code from the translator that must be a positive integer or 0. If you get a return code value greater than *good\_return\_code* from a translator, the process has failed, and no accounting information is saved in the hierarchy. The default is 0. CALLMETH=TSOLNK will result in a return code equal to the translator return code for normal completion, the abend code from the translator, or a 40 in the event of an IKJEFTSR failure.

### **,NOSVEXT=***no\_save\_external\_rc*10

A return code value indicating whether any translator outputs targeted to an external data set were saved (valid for FUNCTN=BUILD). Use this parameter in conjunction with the DEPPRCS parameter on the FLMLANGL macro. It allows or disallows dependency processing if you save some outputs produced by the translator.

The build processor determines that external outputs were not saved by the translator if *no\_save\_external\_rc* is equal to a translator return code other than zero. The default is 0.

### **,OPTFLAG=N|Y**

Indicates whether developers can override default translator options. The default is Y. This parameter has no effect if you specify OPTOVER=N on the FLMCNTRL macro.

**,OPTIONS=***option\_list*

The default translator options. Delimit the options with single quotes or parentheses. They can also contain variables to provide dynamic information to a translator. The maximum length is 255 characters, including delimiters. The @@FLMMBR and @@FLMTYPE variables will be replaced with the name of the member and type of the last SINC, INCL, or INCLD statement in the architecture definition. If a source member is being built, it will be the name of the source member. See Chapter 6. SCLM Variables and Metavariables. Also see the following PARMKWD parameter for more information on options.

The IBM linkage editor requires that the DCBS option parameter be passed in order for the SYSLMOD block size to be used in creating load modules. If the DCBS option is not specified, the linkage editor creates load modules using the maximum record size for the device type. Use the OPTIONS= parameter on the FLMTRNSL macro to pass the DCBS option. Failure to do so can result in message FLM44507 RC4.

The CALLMETH of ISPLNK requires that the option string contain the keywords and parameters for the ISPF SELECT service. The options must be in the format expected by the ISPF SELECT service. For a description of the ISPF SELECT service, refer to the *ISPF Services Guide*

**,PARMKWD=***parameter\_keyword*

The keyword (PARM0..PARM9) used in architecture members to specify additional options for this translator.

**Note:** The complete options list passed to the translator has a maximum length of 512 characters and has the following format:

```
string1
,string2
,string3
```

where

**string1**

is the options from the OPTIONS parameter on the FLMTRNSL macro.

**string2**

is the options from the PARM statements in the architecture definition.

**string3**

is the options from the PARMx statements in the architecture separated by commas.

Extraneous blanks are removed by SCLM.

**,PDSDATA=Y|N**

Specifies whether the input members for this translator reside in SCLM-controlled partitioned data sets. The value of this parameter must be Y for parse and build translators.

If this parameter is not specified, the default varies according to translator function type. The default for parse, build, and verify translators is Y; the default for copy and purge translators is N.

If multiple translators are defined for copy and purge functions, you must not specify Y for one translator and N for another.

## FLMTRNSL Macro

**Note:** SCLM PROMOTE will only invoke Copy and Purge translators for SCLM-controlled partitioned data set members if PDSDATA is set to Y. Copy and Purge translators that operate on nonpartitioned data set controlled parts (such as CSP MSLs) must have PDSDATA set to N.

### **,PORDER=0|1|2|3**

An integer indicating the parameter order to the translator. The translator parameter order must be an integer from 0 to 3. The default is 1. SCLM can pass two kinds of parameters to the translator: the option list and the ddname substitution list. The option list contains the translator options (OPTIONS parameter) concatenated with the options specified in the architecture member (see PARMKWD parameter). The ddname substitution list contains the ddnames specified for allocation. See the DDNAME parameter of "FLMALLOC Macro" on page 130. The following list defines the valid values for the translator parameter order:

- 0 No parameters passed
- 1 Pass option list
- 2 Pass ddname substitution list
- 3 Pass option list followed by ddname substitution list

Ddname substitution lists are a feature of many translators (such as precompilers, utilities, assemblers, and compilers). The correct format of a ddname substitution list is usually unique for each translator and can be located in the programming guide for the translator.

The ddname substitution list is a string of ddnames allocated for the translator. The ddnames appear on the substitution list in the order specified by the FLMALLOC macros in the language definition. Refer to the appendix, "Invoking Utility Programs from Application Programs" in the DFSMS Utilities manual for general information about ddname substitution lists. Refer to the manuals for the specific translator being invoked for details on the substitution list contents expected. For IBM supplied compilers, this information is located in the compiler's Programmers Guide manual under "Invoking Compiler from Application Programs" or "Dynamic Invocation of Compiler". SCLM limits the size of the ddname substitution list to 512 characters or 64 ddnames.

### **,VERSION=*translator\_version***

An 8-character representation of the translator version. This parameter is informational only. SCLM stores this parameter in the account record for each output member saved from the translators. If you do not specify this parameter, SCLM sets it to blank.

### **,CALLMETH=ATTACH|LINK|TSOLNK|ISPLNK**

Indicates whether the translator is to be linked, attached, or invoked by the TSO service facility routine or called through ISPF services. Use ATTACH for load modules unless you need access to ISPF variables or services; in that case, use LINK. Using LINK can result in loops or out-of-space abends because storage is not freed between calls to the translators. The default is ATTACH.

TSOLNK is for translators written as REXX execs. TSOLNK results in the translator being invoked from IKJEFTSR (TSO service facility routine) with parameter 1 of x'00010001'. This parameter indicates that the TSO service facility should invoke the requested translator from an unauthorized environment and that the translator can be a TSO command, REXX exec, or CLIST.

ISPLNK is for translators that must have access to ISPF variables or services. The value specified on the COMPILE parameter is the ISPF service that is used

to call the translator. The only supported value is SELECT. The keywords, including the command to run, are specified in the OPTIONS parameter. The name of the load module, CLIST, REXX exec or other command is specified in the OPTIONS parameter.

The following example shows the CALLMETH, COMPILE, and OPTIONS parameters on an FLMTRNSL macro used to run the FLMLRC37 parser using ISPLNK:

```

FLMTRNSL  CALLNAM='C PARSE',           C
          FUNCTN=PARSE,                 C
          CALLMETH=ISPLNK,              C
          COMPILE=SELECT,                C
          PORDER=1,                     C
          OPTIONS='PGM(FMLRC37) PARM(STATINFO=@@FLMSTP,LISTINFO=@C
          @FLMLIS,LISTSIZE=@@FLMSIZ)'
```

**,TASKLIB=translator\_ddname**

The ddname associated with one or more data sets that contain the translator load module. The data sets must be specified using an FLMALLOC macro with DDNAME=translator\_ddname. When specified for a translator using a DDNAME substitution list, the TASKLIB allocation does not appear in the list passed to the translator.

TASKLIB is only valid for CALLMETH=ATTACH. The operating system searches for executable members in the specified DSNNAME parameter, then in the TASKLIB concatenation, and then in the system concatenation.

**,INPLIST=N|Y**

Indicates that this translator supports Input List Processing. You must specify INPLIST=Y to use the IBM Ada/370 Compiler input list function.

**,MBRRC=maximum\_good\_return\_code**

Use this parameter in conjunction with the INPLIST parameter. MBRRC indicates the maximum value of the *good\_return\_code* for each member in the Input List. This parameter is similar in function to the GOODRC parameter for the translator. However, the GOODRC parameter applies to a single return code supplied by the translator. The MBRRC parameter indicates the maximum valid value for any member of the Input List.

## Examples

A translator for the Pascal compiler is defined. The compiler is member PASCALVS in data set SYS2.VSPASCAL.LOAD. The translator can only be invoked by the build processor (FUNCTN=BUILD). The build processor refers to the compiler by its call name, PASCAL COMPILER. Only the option list can be passed to the translator (PORDER=1). The default options for this translator are specified by the OPTIONS parameter. Build considers any translator return code greater than 0 as an error (GOODRC=0).

```

FLMTRNSL CALLNAM='PASCAL COMPILER',     X
          FUNCTN=BUILD,                  X
          COMPILE=PASCALVS,              X
          DSNNAME=SYS2.VSPASCAL.LOAD,    X
          VERSION=1.0,                   X
          GOODRC=0,                       X
          PORDER=1,                       X
          OPTIONS='NOXREF,CHECK,LINECOUNT(75),NOOPT'
```

### FLMTYPE Macro

Use this macro to define each type in the project definition. This macro is required and can be used multiple times in a project definition.

#### Macro Format

```
name FLMTYPE [EXTEND=extended_type]
```

#### Parameters

**name**

An 8-character type name.

**EXTEND=***extended\_type*

An 8-character name that can be used as an alternate type when resolving include dependencies.

The type specified for the EXTEND parameter is substituted into the @@FLMETP variable on FLMINCLS macros in language definitions. The @@FLMETP is used to define the types that are searched to find included members.

#### Example

Six types are defined. Type SOURCE2 is an extension of type SOURCE. In SCLM, if a member exists in type SOURCE, its included dependencies can exist in either SOURCE or SOURCE2.

```
OBJ      FLMTYPE
LIST     FLMTYPE
LMAP     FLMTYPE
LOAD     FLMTYPE
SOURCE   FLMTYPE EXTEND=SOURCE2
SOURCE2  FLMTYPE
```

---

## Chapter 5. SCLM Translators

This chapter describes the translators delivered with SCLM. The translators are used in language definitions shipped with SCLM. You can modify these language definitions for the specific needs of your site and environment. The supplied parsers might not recognize all syntax rules for a specific language, and you might have to modify statements to adhere to generic syntax.

*Table 6. Translators*

Translator	Page
FLMCSPDB	"FLMCSPDB DB2 Bind/Free Translator" on page 199
FLMDTLC	"FLMDTLC DTL Processor Build Translator" on page 202
FLMLPCBL	"FLMLPCBL COBOL Parser" on page 203
FLMLPFRT	"FLMLPFRT FORTRAN Parser" on page 206
FLMLPGEN	"FLMLPGEN General Purpose Parser" on page 210
FLMLRASM	"FLMLRASM REXX Assembler Parser" on page 214
FLMLRCBL	"FLMLRCBL REXX COBOL Parser" on page 218
FLMLRCIS	"FLMLRCIS MVS C/C++ parser with include set support" on page 222
FLMLRC2	"FLMLRC2 C, C++, and Resource file parser for workstation source" on page 225
FLMLRC37	"FLMLRC37 REXX C370 Parser" on page 228
FLMLRDTL	"FLMLRDTL REXX DTL Parser" on page 232
FLMLRIPF	"FLMLRIPF Script and OS/2 IPF Source Parser" on page 233
FLMLSS	"FLMLSS General Purpose Parser" on page 236
FLMLTWST	"FLMLTWST Workstation Build Translator" on page 240
FLMTBMAP	"FLMTBMAP Build Map Print - Build Translator" on page 256
FLMTMSI	"FLMTMSI Interface to SCRIPT/VS" on page 258
FLMTPRE	"FLMTPRE" on page 259
FLMTPST	"FLMTPST" on page 261
FLMTXFER	"FLMTXFER Workstation Transfer - Build Translator" on page 263

There are five types of translators:

- A parse translator gathers statistics and dependencies. Parse translators run during migration, when the member is saved in an edit session, or when the SAVE or PARSE service is called. A parse translator can also be used to define user data and change codes for the member.
- A verify translator can be used to perform validation in addition to default SCLM validation. The verify translator can be used to check the change codes or user data defined for members. Another example could be verification of data

that is related to an SCLM-controlled member but is not under SCLM control itself. Verify translators run during build and promote verification.

For builds, SCLM invokes a verify translator to verify inputs to build translators. For example, when an LEC architecture definition is being built, the source member is verified prior to compiling and the object member is verified prior to linking.

For promotes, SCLM invokes a verify translator to verify build inputs and outputs. For example, when an LEC architecture definition is being promoted, the source, object, and load members are verified prior to the promote copy phase.

- A build translator can assemble, compile, link, or otherwise process a member so that the outputs have different formats than the inputs. For example, building a COBOL source program generates a listing and an object module.
- A copy translator is invoked when Promote copies an SCLM-controlled member to the next group in the hierarchy. Copy translators are invoked before Promote copies the SCLM-controlled member. If the copy translators for a member fail, Promote does not attempt to copy the controlled member. Copy translators can be used to copy data that is related to an SCLM-controlled member but is not under SCLM control itself.
- Purge translators can be used to purge data that is related to an SCLM-controlled member but is not under SCLM control itself. Purge translators are invoked whenever SCLM performs a delete operation on an SCLM-controlled member during build or promote.

---

## FLMCSPDB DB2 Bind/Free Translator

### Purpose

This is the DB2 Bind/Free translator to be used for binding and freeing DB2 plans. It is necessary to create a DB2 CLIST that will specify the DBRMs to be bound with the DB2 plan, as well as the name of the DB2 plan. An example of these can be found in *ISPF Software Configuration and Library Manager (SCLM) Developer's and Project Manager's Guide*.

### Parameters

The following guidelines apply when specifying parameters:

- The order of the parameters is not important.
- See the language definitions provided by SCLM for the actual usage of the parameters for the translator.

The following keyword parameters are expected as input for the translator:

#### ALTPROJ

The variable name for the alternate project name. This parameter is required, and must be set to @@FLMALT.

#### DBRMTYPE

The type name where the DBRMs are stored. This parameter is required.

**Note:** The FLMDBALC CLIST is run by the FLMCSPDB translator to allocate the DBRM type to the DBRMLIB DD name.

#### FUNCTN

The SCLM function invoking the translator: BUILD, COPY, or PURGE. This parameter is required.

#### GROUP

The variable name for the build group name. This parameter is required, and must be set to @@FLMGRP.

#### MEMBER

The variable name for the DBRM member name. This parameter is required, and must be set to @@FLMMBR.

#### OPTION

The operation to be performed with the DB2 Plan: BIND or FREE. This parameter is required.

#### PROJECT

The variable name for the project name. This parameter is required, and must be set to @@FLMPRJ.

#### SCLMINFO

The variable name for the SCLM internal pointer. This parameter is required, and must be set to @@FLMINF.

#### TOGROUP

The variable name for the group to promote to. This parameter is required, and must be set to @@FLMTOG. This variable is ignored for FUNCTN=BUILD.

### Return Codes

---

0

**Explanation:** Success

**User response:** None.

**Project manager response:** None.

---

4

**Explanation:** A WARNING message was produced.

**User response:** Look at the message. Fix the problem if necessary.

**Project manager response:** None.

---

01300

**Module:** FLMCSPDB

**Explanation:** The FUNCTN parameter is not specified correctly in the input options defined for the translator. This parameter is one of the OPTIONS parameters for this translator. This parameter is passed to the translator by way of the OPTIONS= parameter for calls to the FLMCSPDB translator. This parameter is not to be confused with the FUNCTN= parameter passed to SCLM using the FLMTRNSL macro; rather, it is a secondary parameter value for the OPTIONS= parameter that is passed to the translator using the FLMTRNSL macro.

**User response:** None.

**Project manager response:** Verify that "OPTIONS=(FUNCTN=BUILD...)" or "OPTIONS=(FUNCTN=PROMOTE...)" is specified for the FLMTRNSL macro invoking the FLMCSPDB translator.

---

01310

**Module:** FLMCSPDB

**Explanation:** The OPTION parameter is not specified correctly in the input options defined for the translator.

**User response:** None.

**Project manager response:** Verify that "OPTION=BIND" or "OPTION=FREE" is specified as an option for the translator in the language definition.

---

01320

**Module:** FLMCSPDB

**Explanation:** The GROUP parameter is not specified in the input options defined for the translator.

**User response:** None.

**Project manager response:** Verify that "GROUP=@@FLMGRP" is specified as an option for the translator in the language definition.

---

01330

**Module:** FLMCSPDB

**Explanation:** The TOGROUP parameter is not specified in the input options defined for the translator.

**User response:** None.

**Project manager response:** Verify that "TOGROUP=@@FLMTOG" is specified as an option for the translator in the language definition.

---

01340

**Module:** FLMCSPDB

**Explanation:** The MEMBER parameter is not specified as a PARM input for the translator.

**User response:** None.

**Project manager response:** Verify that "MEMBER=@@FLMMBR" is specified as an option for the translator in the language definition.

Verify that OPTOVER=Y on the FLMCNTRL macro in the project definition.

---

01350

**Module:** FLMCSPDB

**Explanation:** The SCLMINFO parameter is not specified in the input options defined for the translator.

**User response:** None.

**Project manager response:** Verify that "SCLMINFO=@@FLMINF" is specified as an option for the translator in the language definition.

---

01360

**Module:** FLMCSPDB

**Explanation:** The DBRMTYPE parameter is not specified in the input options defined for the translator.

**User response:** None.

**Project manager response:** Verify that "DBRMTYPE=dbrmtype" is specified as an option for the translator in the language definition. Where dbrmtype has been defined as a valid type in the project definition.

---

01370

**Module:** FLMCSPDB

**Explanation:** An error occurred while executing the DB2 CLIST member.

**User response:** Verify that DB2 is installed and that you have invoked it correctly from the DB2 CLIST. Trace the execution of your CLIST to verify the return code.

---

**Project manager response:** None.

---

**01380**

**Module:** FLMCSPDB

**Explanation:** The DB2 CLIST member does not contain DSN commands for the group being processed.

**User response:** Verify that the DB2 CLIST member has code specifying the DSN commands required for the group being processed.

**Project manager response:** None.

---

**01390**

**Module:** FLMCSPDB

**Explanation:** The PROJECT parameter is not specified in the input options defined for the translator.

**User response:** None.

**Project manager response:** Verify that "PROJECT=@@FLMPRJ" is specified as an option for the translator in the language definition.

---

**01420**

**Module:** FLMCSPDB

**Explanation:** The data set allocation (DSALLOC) service failed.

**User response:** Contact the project manager.

**Project manager response:** Verify that SCLM skeleton FLMLIBS has all necessary allocations for the CSP/AD and/or DB2 products.

---

**21310**

**Explanation:** SCLM received an error initializing the DB2 CLIST.

**User response:** None.

**Project manager response:** Verify that the DB2 CLIST exists.

---

**21320**

**Explanation:** SCLM received an error initializing the DB2 OUT data set.

**User response:** None.

**Project manager response:** Verify that the DB2 OUT data set exists.

---

**21330**

**Explanation:** SCLM received an error when it attempted to copy the DB2 CLIST to the DB2 OUT data set.

**User response:** None.

**Project manager response:** Verify that the DB2 OUT data set attributes are complementary with the DB2 CLIST dataset.

---

**21340**

**Explanation:** SCLM could not find the FLMPROXY ddname.

**User response:** None.

**Project manager response:** Verify that the FLMPROXY ddname was passed to the translator.

---

**21370**

**Explanation:** Non-key groups are not supported.

**User response:** None.

**Project manager response:** Delete non-key groups or re-specify non-key groups as key groups in the project definition. See return code 30108 for more detail.

---

# FLMDTLC DTL Processor Build Translator

## Purpose

This is the DTL Processor Build translator. It is called from SCLM builds of ISPF Dialog Tag Language to invoke the DTL Conversion Utility.

## Parameters

The following parameters are expected as input for the translator:

- Project level qualifier of source data set
- Group level qualifier for the build level of the source data set
- Type level qualifier for the source data set
- Member name of source DTL
- ISPF application-id.

The following are the outputs expected:

- Log listing with ISPDTLC information (FLMDTLC)
- Generated panel (\$PANELS)
- Generated message member (\$MSGGS)
- Generated keylist member or command table (\$TABLES).

## Return Codes

---

0

**Explanation:** Indicates successful completion.

**User response:** None.

**Project manager response:** None.

---

4

**Explanation:** DTL completed with a return code of 8.

**User response:** Only warnings were found during the DTL conversion. Check the DTL source code.

**Project manager response:** None.

---

8

**Explanation:** DTL completed with a return code of 16.

**User response:** At least one error was found during DTL conversion. Check the DTL source code.

**Project manager response:** None.

---

16

**Explanation:** DTL returned a return code of 20. Fatal error.

**User response:** Check the DTL source code.

**Project manager response:** None.

---

---

20

**Explanation:** DTL returned an unknown return code.

**User response:** Contact the project manager.

**Project manager response:** Review the messages in the DTL log. Contact IBM support if necessary.

---

# FLMLPCBL COBOL Parser

## Purpose

This is the COBOL parser translator that parses the source identified by the SOURCE DDNAME.

## Functions

One of the functions of an SCLM parser is to determine all of a module's dependencies. FLMLPCBL determines all of the names that will be copied into the COBOL source.

FLMLPCBL examines each line of the member. Lines located in the IDENTIFICATION DIVISION (ID DIVISION) will not be examined for COPY statements or quoted strings. This will permit the use of a comment entry for each paragraph in the ID DIVISION without the need for an asterisk or slash in column 7.

The parser uses the following syntax rules to locate dependency names outside of the ID DIVISION:

- The search for tokens is restricted to columns 8–72. Column 7 is ignored except when it contains \* and / (treated as a comment line) or - (treated as a concatenation). The use of - to concatenate strings for forming reserved words or dependency names is not supported.
- DBCS strings (delimited by shift-out and shift-in characters) in comments and quotes are allowed.
- When an uncommented, noncontinuation line has COPY after column 7, the next token is taken as the name of a dependency. Note the following exceptions:
  - If the member name is enclosed in single or double quotes, the quotes are ignored.
  - When an uncommented, noncontinuation line has EXEC, SQL, and INCLUDE as its first three tokens after column 7, the next token is taken as the name of a dependency.
  - When searching for the next token on a line and there are no more tokens on that line, the search continues with the next uncommented line.
  - Tokens inside quoted strings will be ignored (except for COPY member names). Reserved words inside quoted strings will not be counted as statements. COPY, EXEC SQL, and EXEC CICS\* inside quoted strings will be ignored.
  - If a token is longer than 8 characters, it will not be added as a dependency.

FLMLPCBL recognizes COPY MEMBER where MEMBER is a 1- to 8-character string with no separator periods. A separator period is *not* required after MEMBER.

COPY and MEMBER must be on the same line or on a continued line. However, splitting COPY or MEMBER by using a hyphen (-) in column 7 of the continued line is not supported. This is important to consider when using code generators that use the hyphen in column 7 to concatenate strings to form keywords and text names. Use of the hyphen to concatenate strings in order to form a MEMBER as in COPY MEMBER results in the COPY being ignored by FLMLPCBL. Use of the hyphen on the line after COPY when COPY is the last token on the line results in the COPY being ignored by FLMLPCBL.

FLMLPCBL can parse an odd number of quote delimiters if the first two nonblank characters after column 7 on a continuation line are two quote delimiters. FLMLPCBL expects to find an even number of quote delimiters in a literal. You might need to introduce a constant with a literal value that is also continued on the next line to produce an even number of quote delimiters. If you have an odd number of quote delimiters, the dependencies following the odd number of delimiters might be ignored. The following example illustrates a statement with an odd number of quote delimiters:

```
123456 VALUE 'This literal has a quote in column 72 and the next '
123457 '' 2 quote delimiters result in an odd number of quote
123458 delimiters for this statement '.
```

The parser also gathers statistics or metrics for each module to be parsed. SCLM saves 10 statistics, but this parser only generates 7. For COBOL, this parser defines the following statistics:

Total lines	The total number of records in the file.																																													
Comment lines	The number of lines with a slash (/) or an asterisk (*) in column 7.																																													
Noncomment lines	The number of total lines minus the number of comment lines.																																													
Blank lines	The number of lines that contain only blanks after column 6. Any sequence numbers in the rightmost columns of the line are ignored.																																													
Prolog lines	The number of comment lines that are found before the first noncomment line.																																													
Total statements	The number of noncomment, noncontinuation lines whose first token after column 7 is one of the following reserved words: <table border="0" style="margin-left: 40px;"> <tr> <td>ACCEPT</td> <td>EVALUATE</td> <td>PERFORM</td> </tr> <tr> <td>ADD</td> <td>EXAMINE</td> <td>READ</td> </tr> <tr> <td>ALTER</td> <td>EXIT</td> <td>RELEASE</td> </tr> <tr> <td>CALL</td> <td>GO</td> <td>RETURN</td> </tr> <tr> <td>CANCEL</td> <td>GOBACK</td> <td>REWRITE</td> </tr> <tr> <td>CLOSE</td> <td>GOTO</td> <td>SEARCH</td> </tr> <tr> <td>COMPUTE</td> <td>IF</td> <td>SET</td> </tr> <tr> <td>CONTINUE</td> <td>INITIALIZE</td> <td>SORT</td> </tr> <tr> <td>COPY</td> <td>INSPECT</td> <td>START</td> </tr> <tr> <td>DELETE</td> <td>MERGE</td> <td>STOP</td> </tr> <tr> <td>DISPLAY</td> <td>MOVE</td> <td>STRING</td> </tr> <tr> <td>DIVIDE</td> <td>MULTIPLY</td> <td>SUBTRACT</td> </tr> <tr> <td>ENTER</td> <td>NOTE</td> <td>TRANSFORM</td> </tr> <tr> <td>ENTRY</td> <td>ON</td> <td>UNSTRING</td> </tr> <tr> <td></td> <td>OPEN</td> <td>WRITE</td> </tr> </table> <p style="margin-left: 40px;">In addition, any EXEC SQL and EXEC CICS statements are treated as program statements.</p>	ACCEPT	EVALUATE	PERFORM	ADD	EXAMINE	READ	ALTER	EXIT	RELEASE	CALL	GO	RETURN	CANCEL	GOBACK	REWRITE	CLOSE	GOTO	SEARCH	COMPUTE	IF	SET	CONTINUE	INITIALIZE	SORT	COPY	INSPECT	START	DELETE	MERGE	STOP	DISPLAY	MOVE	STRING	DIVIDE	MULTIPLY	SUBTRACT	ENTER	NOTE	TRANSFORM	ENTRY	ON	UNSTRING		OPEN	WRITE
ACCEPT	EVALUATE	PERFORM																																												
ADD	EXAMINE	READ																																												
ALTER	EXIT	RELEASE																																												
CALL	GO	RETURN																																												
CANCEL	GOBACK	REWRITE																																												
CLOSE	GOTO	SEARCH																																												
COMPUTE	IF	SET																																												
CONTINUE	INITIALIZE	SORT																																												
COPY	INSPECT	START																																												
DELETE	MERGE	STOP																																												
DISPLAY	MOVE	STRING																																												
DIVIDE	MULTIPLY	SUBTRACT																																												
ENTER	NOTE	TRANSFORM																																												
ENTRY	ON	UNSTRING																																												
	OPEN	WRITE																																												
Comment statements	This value is always 0.																																													
Control statements	This value is always 0.																																													
Assignment statements	This value is always 0.																																													
Noncomment statements	This is the same as Total statements.																																													

## Parameters

The following positional parameters, separated by commas, are expected as input to FLMLPCBL:

**@@FLMLIS**

The address of the dependencies pointer. This parameter is required.

**@@FLMSIZ**

The size of the dependency list buffer in bytes. This parameter is required.

**@@FLMSTP**

The address of the statistics output buffer. This parameter is required.

## Return Codes

---

0

**Explanation:** Indicates successful completion.

**User response:** None.

**Project manager response:** None.

---

4

**Explanation:** The dependency list does not match the source code for one of the following reasons:

- Truncation to 8 characters
- No trailing quotation mark to match a leading quotation mark
- Token consists of only 1 quotation mark
- Token consists of only 2 quotation marks
- Token is split between 2 lines using a hyphen in column 7 for concatenation

The dependency is not added to the list. Processing continues.

**User response:** Change the syntax to fit the parser.

**Project manager response:** None.

---

8

**Explanation:** The number of parsed dependencies exceeds the size of the \$list\_info array, which is specified by the BUFSIZE parameter on the FLMLANGL macro.

**User response:** Either reduce the number of parsed dependencies for the member or contact the project manager.

**Project manager response:** Increase the buffer size (BUFSIZE=) on the FLMLANGL macro for the appropriate language definition. Reassemble and relink the project definition.

---

12

**Explanation:** FLMTRNSL parameters are incorrect or are not specified.

**User response:** Contact the project manager.

**Project manager response:** Verify that the FLMTRNSL parameters on the FLMTRNSL macro for the FLMLPCBL parser are valid.

---

16

**Explanation:** A GETMAIN error for I/O storage occurred. Processing stops.

**User response:** Contact the project manager.

**Project manager response:** Contact your IBM service representative.

---

20

**Explanation:** A severe error occurred. The source to be parsed cannot be opened or the LRECL is less than 16. Processing stops.

**User response:** Contact the project manager.

**Project manager response:** Verify the LRECL of the source file is 16 or greater.

---

22

**Explanation:** An I/O error occurred in the DCB while reading input. Processing stops.

**User response:** Contact the project manager.

**Project manager response:** Contact your IBM service representative.

---

# FLMLPFRT FORTRAN Parser

## Purpose

This is the FORTRAN parser translator that parses the source identified by the SOURCE DDNAME.

## Using FLMLPFRT

The FORTRAN parser uses the following rules:

- Source must be fixed 80 and must be of the fixed form input format. Comment characters ('C' and '\*') are recognized in column 1, continuation characters are recognized in column 6, and source statements are recognized in columns 7-72.
- Includes recognized are of the forms  
INCLUDE (NAME)  
EXEC SQL INCLUDE NAME
- INCLUDE statements can span lines if continuation characters are used.
- EXEC SQL INCLUDE statements are recognized and dependencies are generated (SQLCA and SQLDA are not flagged as external dependencies). SQL includes can span lines if continuation characters are used. All other EXEC statements are not flagged as a dependency.
- Comments and the contents of quoted strings are ignored.
- DBCS strings (delimited by shift-out and shift-in characters) in comments and quotes are allowed.

FLMLPFRT collects the following statistics about the source to be parsed:

Total lines	The total number of records in the file.
Comment lines	The number of lines with a (C) or an asterisk (*) in column 1 plus continued comments. A continued comment is a line that has a nonblank continuation character in column 6 and that follows a comment line or a continued comment.
Noncomment lines	The number of lines that are not comment lines, continued comment lines, or blank lines.
Blank lines	The number of lines that contain only blanks.
Prolog lines	The number of comment lines that are found before the first noncomment line.
Total statements	Comment statements plus noncomment statements.
Comment statements	The number of comment lines minus the number of lines that are continued comments.
Control statements	This value is always 0.
Assignment statements	This value is always 0.
Noncomment statements	The number of noncomment lines minus the number of lines that are continued noncomments. A continued noncomment is a line that has a nonblank continuation character in column 6 and that follows a noncomment line or a continued noncomment.

## Parameters

The following keyword parameters are expected as input for FLMLPFRT:

**LISTINFO**

Pointer to the SCLM list information record. This parameter is required and must be set to @@FLMLIS.

**LISTSIZE**

The size of the LISTINFO buffer. This parameter is required and must be set to @@FLMSIZ.

**PARSEDSN**

Data set name containing the member to be parsed. The SCLM variable @@FLMDSN is the recommended value. This parameter is required.

**PARSEMEM**

The name of the member to be parsed. The SCLM variable @@FLMMBR is the recommended value. This parameter is required.

**SOURCEDD**

The ddname of the source to be read. This parameter is optional. If a SOURCEDD is specified, it will override the PARSEDSN and PARSEMEM parameters.

**STATINFO**

Pointer to the SCLM statistics information record. This parameter is required and must be set to @@FLMSTP.

## Return Codes

FLMLPFRT uses ISPF services. When a failure is the result of an ISPF service error, the message returned by the ISPF service is logged in the user's ISPF log (if there is one).

---

0

**Explanation:** Indicates successful completion.

**User response:** None.

**Project manager response:** None.

---

1

**Explanation:** Data set name not found in parameter list.

**User response:** Contact the project manager.

**Project manager response:** Check the language definition syntax (PARSEDSN parameter on the OPTIONS parameter on the FLMTRNSL FORTRAN parser). Verify that PORDER=1 or PORDER=3 was used on the FLMTRNSL macro of the language definition. A PORDER of 0 or 2 in the FLMTRNSL macro for the FLMLPFRT parser will result in FLMLPFRT receiving control without the OPTION list. PORDER 0 and 2 are used for situations in which there are no OPTION lists.

---

2

**Explanation:** The statistical information address (@@FLMSTP) is not found in the parameter list.

**User response:** Contact the project manager.

**Project manager response:** Check the language

definition syntax (STATINFO parameter on the OPTIONS parameter on the FLMTRNSL FORTRAN parser).

---

3

**Explanation:** The list information address (@@FLMLIS) is not found in the parameter list.

**User response:** Contact the project manager.

**Project manager response:** Check the language definition syntax (LISTINFO parameter on the OPTIONS parameter on the FLMTRNSL FORTRAN parser).

---

4

**Explanation:** A dependency name was encountered that had more than 8 characters. The name is ignored and processing continues.

**User response:** Check the source member for dependency names longer than 8 characters. Dependency names are restricted to a length of 1 to 8 characters.

**Project manager response:** The language definition can be changed for GOODRC=4 if it is acceptable to ignore the dependency names that are longer than 8 characters.

---

5

**Explanation:** The STATINFO, LISTINFO, and/or LISTSIZE keyword parameters are invalid.

**User response:** Check the language definition for the correct values for keyword parameters STATINFO, LISTINFO, and LISTSIZE.

**Project manager response:** The invalid keyword parameters for FLMLPFRT OPTIONS in the language definition should be corrected and the project definition assembled and linked.

---

8

**Explanation:** The number of parsed dependencies exceeds the size of the \$list\_info array, which is specified by the BUFSIZE parameter on the FLMLANGL macro.

**User response:** Either reduce the number of parsed dependencies for the member or contact the project manager.

**Project manager response:** Increase the buffer size (BUFSIZE=) on the FLMLANGL macro for the appropriate language definition. Be sure that LISTSIZE on the FLMTRNSL macro is set to @@FLMSIZ. Reassemble and relink the project definition.

---

10

**Explanation:** The member name specified by the parse parameter is blank.

**User response:** None.

**Project manager response:** Check the language definition syntax and member specification.

---

12

**Explanation:** The LISTSIZE keyword parameter in the OPTIONS is too small. There is not enough room for one element in the dependency array.

**User response:** Contact the project manager.

**Project manager response:** Update the project definition, assemble, and link-edit. LISTSIZE must be set to @@FLMSIZ in order to get a proper value for BUFSIZE.

---

24

**Explanation:** Parser was not linked AMODE(24), RMODE(24).

**User response:** Contact the project manager.

**Project manager response:** Reinstall the parser by relinking it AMODE(24). See ISPF log for more details.

---

100

**Explanation:** The value for the PARSEDSN keyword is invalid.

**User response:** Check the language definition and verify that PARSEDSN keyword value is valid.

**Project manager response:** None.

---

101 - 199

**Explanation:** The data set specified by the PARSEDSN keyword could not be allocated.

**User response:** Verify that the data set designated by the keyword exists.

**Project manager response:** None.

---

201 - 299

**Explanation:** The data set specified by the PARSEDSN keyword could not be opened, or is already opened.

**User response:** Verify that the data set exists, is not in use, and has not been allocated with a disposition of SHR or MOD.

**Project manager response:** None.

---

401 - 499

**Explanation:** An error occurred reading the data set specified by the PARSEDSN keyword. The data set is either empty, not opened for input, or has exceeded its space capacity.

**User response:** Verify that the data set exists, it is not empty, and the space allocation will support the process.

**Project manager response:** None.

---

500

**Explanation:** An error occurred when closing the file or when freeing storage.

**User response:** Contact the project manager.

**Project manager response:** Contact your IBM Service Representative.

---

599

**Explanation:** An ABEND was detected during I/O or allocation of the data set to be parsed.

**User response:** Check if the data set member to be parsed exists. An improper value for STATINFO in the OPTIONS of the FLMTRNSL for the parser can be another cause.

**Project manager response:** Make sure the data set member exists. Make corrections to the project

---

definition; assemble and link the project definition.

---

# FLMLPGEN General Purpose Parser

## Purpose

FLMLPGEN is a general purpose parser that can get dependency information and statistics for the following languages:

370 Assembler  
PL/I  
REXX  
CLIST  
TEXT

General information:

- Comments and the contents of quoted strings are ignored.
- DBCS strings (delimited by shift-out and shift-in characters) in comments and quotes are allowed.
- Total lines and blank lines are always counted.
- Control statements and assignment statements are always set to zero.

## Using FLMLPGEN as an Assembler Parser

The Assembler parser uses the following rules:

- Set LANG=A for Assembler in the option list of the OPTIONS parameter, in the FLMTRNSL macro, and in the language definition macro (FLMLANGL).
- COPY statements with a continuation character in column 72 will be ignored.
- Any opcode not recognized as a standard 370 opcode is considered to be an external dependency (see next item).
- Macros that are defined inline are not flagged as external dependencies.
- Vector, ESA, and z/Series opcodes are recognized.
- OPSYN, ISEQ, ICTL, and others that alter the language or defaults are ignored.
- EXEC SQL INCLUDE statements are recognized and dependencies are generated (SQLCA and SQLDA are not flagged as external dependencies). SQL includes can span lines. All other EXEC statements are not flagged as a dependency.

## Using FLMLPGEN as a PL/I Parser

The PL/I parser uses the following rules:

- In the language definition, set LANG=I or LANG=1 for PL/I.
- Statements are just the number of semicolons not in comments or quotes plus commas not in parentheses. The following example has six statements (note the first DCL statement counts as three statements, but the second only counts as one because the commas are in parentheses).

```
EXAMPLE:PROC;  
  DCL ONE  FIXED(31),  
      TWO  FIXED(31),  
      THREE FIXED(31);  
  
  DCL (A_ONE, AN_A_TWO, AN_A_THREE) FIXED(31);  
  
END EXAMPLE;
```

- Include statements cannot span lines.
- Include statements can include a ddname (as per PL/I syntax).
- Only the first %INCLUDE on a line will be recognized. Multiple dependencies are allowed on one line:

```
%INCLUDE A, B, DD1(C), DD2(D) ...
```

- Preprocessor labels on include statements cause those includes to be missed.
- EXEC SQL INCLUDE statements are recognized and dependencies are generated (SQLCA and SQLDA are not flagged as external dependencies). SQL includes can span lines.
- Multiple EXEC SQL INCLUDE statements can appear on one line and the dependencies will be generated.
- Dependencies are recognized from all ddnames.

### Using FLMLPGEN as a CLIST, REXX or Generic Parser

FLMLPGEN uses the following rules for the CLIST, REXX, and generic parsers.

- In the language definition, set LANG=C for CLIST, LANG=R for REXX, or LANG=T for Generic Parser.
- Source can be any format (fixed or variable) up to record length 255.
- Sequence numbers are ignored.
- Continuation of statements is recognized by the following:
  - + and - for CLIST
  - , for REXX
- Open comments (/\* only) for CLIST are allowed. They are considered closed at the end of the line if there is no continuation character.
- The REXX language can be used to gather statistics for other languages that use /\* and \*/ as delimiters such as ISPF panels. (Statistics might not be correct if any commas are at the end of any lines.)

### Using FLMLPGEN as a TEXT Parser

FLMLPGEN uses the following rules for parsing TEXT members.

- In the language definition, set LANG=T for TEXT.
- Source can be any format (fixed or variable) and any valid record length.
- Sequence numbering is counted as nonblank lines.
- Only total lines and blank lines are counted.
- Control statements and assignment statements are always set to zero.

## Parameters

The following keyword parameters are expected as input for FLMLPGEN:

#### LANG=A|T|R|C|I|1|

Use the LANG= parameter to specify the language to use to parse the members. If you do not include the LANG= parameter, the members are parsed as 370 Assembler. Valid language values are:

LANG=A	Assembler only
LANG=T	TEXT... count lines only
LANG=R	REXX or similar languages that use /* and */ as comment delimiters
LANG=C	CLIST
LANG=I	PL/I
LANG=1	PL/I

#### LISTINFO

Pointer to the SCLM list information record. This parameter is required and must be set to @@FLMLIS.

#### LISTSIZE

The size of the LISTINFO buffer. This parameter is required and must be set to @@FLMSIZ.

## SOURCEDD

The ddname of the source to be read. This parameter is required.

## STATINFO

Pointer to the SCLM statistics information record. This parameter is required and must be set to @@FLMSTP.

## Return Codes

FLMLPGEN uses ISPF services. When a failure is the result of an ISPF service error, the message returned by the ISPF service is logged in the user's ISPF log (if there is one).

---

0

**Explanation:** Indicates successful completion.

**User response:** None.

**Project manager response:** None.

---

1

**Explanation:** Data set name not found in parameter list.

**User response:** Contact the project manager.

**Project manager response:** Check the language definition PORDER value and syntax.

---

2

**Explanation:** The statistical information address (@@FLMSTP) was not found in the parameter list.

**User response:** Contact the project manager.

**Project manager response:** Check the language definition syntax (STATINFO parameter on the OPTIONS parameter on the FLMTRNSL FLMLPGEN parser).

---

3

**Explanation:** The list information address (@@FLMLIS) was not found in the parameter list.

**User response:** Contact the project manager.

**Project manager response:** Check the language definition syntax (LISTINFO parameter on the OPTIONS parameter on the FLMTRNSL FLMLPGEN parser).

---

4

**Explanation:** Dependency name longer than 8 characters was recognized. The dependency is not added to the dependency list. Processing continues.

**User response:** Verify and correct the length of the dependency name.

**Project manager response:** None.

---

---

5

**Explanation:** Maximum list size (LISTSIZE) was not found in parameter list.

**User response:** None.

**Project manager response:** Check the language definition syntax (LISTSIZE parameter on the OPTIONS parameter on the FLMTRNSL FLMLPGEN parser).

---

7

**Explanation:** The number of parsed dependencies exceeds the size of the \$list\_info array, that is specified by the BUFSIZE parameter on the FLMLANGL macro.

**User response:** Either reduce the number of parsed dependencies for the member or contact the project manager.

**Project manager response:** Increase the buffer size (BUFSIZE=) on the FLMLANGL macro for the appropriate language definition. Be sure that LISTSIZE on the FLMTRNSL macro is set to @@FLMSIZ. Reassemble and relink the project definition.

---

10

**Explanation:** Member name not found.

**User response:** None.

**Project manager response:** Check the language definition syntax.

---

22

**Explanation:** I/O error.

**User response:** Check the source code for a dependency name greater than 8 characters. The I/O error may occur when the TSO prefix is not set and the parser attempts to allocate an error data set. If the TSO prefix was not set then set the TSO prefix and run the parse again if you cannot locate the dependency name that is greater than 8 characters.

**Project manager response:** Verify the data sets used by the parser OPEN and CLOSE properly.

---

---

24

**Explanation:** Parser was not linked AMODE(24), RMODE(24).

**User response:** None.

**Project manager response:** Reinstall the parser by relinking it AMODE(24). See ISPF log for more details.

---

project definition; assemble and link the project definition.

---

101 - 199

**Explanation:** The data set specified by the PARSEDSN keyword could not be allocated.

**User response:** Contact the project manager.

**Project manager response:** Verify that the data set designated by the keyword exists.

---

---

201 - 299

**Explanation:** The data set specified by the PARSEDSN keyword could not be opened, or is already opened.

**User response:** Verify that the data set exists, is not in use, and has not been allocated with a disposition of SHR or MOD.

**Project manager response:** None.

---

---

401 - 499

**Explanation:** An error occurred reading the data set specified by the PARSEDSN keyword. The data set is either empty, not opened for input, or has exceeded its space capacity.

**User response:** Verify that the data set exists, it is not empty, and the space allocation will support the process.

**Project manager response:** None.

---

---

500

**Explanation:** An error occurred when closing the file or when freeing storage.

**User response:** Contact the project manager.

**Project manager response:** Contact your IBM service representative.

---

---

599

**Explanation:** An ABEND was detected during I/O or allocation of the data set to be parsed.

**User response:** Check to see if the data set member to be parsed exists. An improper value for STATINFO in the OPTIONS of the FLMTRNSL for the parser can be another cause.

**Project manager response:** Make sure the data set member exists. Make any necessary corrections to the

---

# FLMLRASM REXX Assembler Parser

## Purpose

This is the assembler parser translator, written in REXX, that parses the source identified by the SOURCE DDNAME.

## Functions

One of the functions of an SCLM parser is to determine all of a module's dependencies. FLMLRASM determines all of the names that are to be copied into the Assembler source.

The parser uses the following syntax rules to locate dependency names:

- The search for tokens is restricted to columns 2 -71. Column 72 is checked for a nonnull element (treated as a continuation). The use of nonnull elements to continue strings for forming reserved words or dependency names is not supported.
- An opcode or dependency token that extends into the continuation column will not be added as a dependency; the parser return code will be set to 4, the line in error will be written to the error listing data set (*userid.SCLMERR.LISTING*), and processing will continue.
- When an uncommented, noncontinuation line has COPY after column 1, the next token is taken as the name of a dependency.

**Note:** If the member name is enclosed in single or double quotes, the quotes are ignored.

- When searching for the next token on a line and there are no more tokens on that line, the search continues with the next continued line, if there is one. Comment statements must not appear between an instruction statement and its continuation lines.
- Tokens inside quoted strings will be ignored (except that quotation marks around a member following a COPY or EXEC SQL INCLUDE statement are removed).
- Labels starting in column 1 to the end of the token are considered white space.

FLMLRASM will generate a dependency for the MEMBER# token under the following conditions:

- MEMBER# is the first token of a statement and is not one of the Op-codes for the z/Series processors (including assembler extended mnemonics, Vector facility and some obsolete 360/370 instructions).
- MEMBER# is the first token after a COPY or EXEC SQL INCLUDE instruction. It can be on a continued line.

The following example illustrates conditions under which dependencies will and will not be formed. Each MEMBER# token appears in an example of syntax that the parser recognizes as creating a dependency. A MEMBER# token can be from 1 to 8 characters. The BADCPY# statements in the example will not create dependencies for the following reasons:

- BADCPY1 follows an EXEC CICS instruction; dependencies are only generated for precompiler instructions (EXEC SQL INCLUDE).
- BADCPY2 first appears in a macro definition, so no dependency is created on subsequent appearances.
- BADCPY3 begins with an ampersand.
- BADCPY4 is not the first token of the statement in which it appears.

```

*-<-Column 1                                Column 72->
  MEMBER1 rest of line
LABEL MEMBER2 rest of line
  COPY MEMBER3 rest of line
  COPY                                         X
      MEMBER4
* A COMMENT LINE *****
* DB2 PREPROCESSOR STATEMENTS - each is 1 statement, 1 dependency
  EXEC SQL INCLUDE MEMBER5
  EXEC SQL INCLUDE                             X
      MEMBER6
* CICS PREPROCESSOR STATEMENT - 1 statement, no dependency
  EXEC CICS BADCPY1
* Statements for which no dependency is generated
  MACRO                                         X
      BADCPY2
&BADCPY3 rest of line
* previously defined macros ignored
  BADCPY2                                       X
      BADCPY4
* continued lines ignored, except after COPY & EXEC SQL INCLUDE

```

Another function of the parser is to gather statistics or metrics for each module to be parsed. There are ten such statistics saved by SCLM, but only 8 are generated by this parser. For assembler, this parser defines the ten statistics as follows:

Total lines	The total number of records in the file.
Comment lines	The number of lines with an asterisk in the first column.
Noncomment lines	The number of total lines minus the number of comment lines.
Blank lines	The number of lines that contain only blanks.
Prolog lines	The number of comment lines and blank lines that are found before the first noncomment line.
Total statements	The number of comment statements plus the number of noncomment statements.
Comment statements	This value is equal to the number of comment lines.
Control statements	This value is always 0.
Assignment statements	This value is always 0.
Noncomment statements	The number of statements whose first token is a reserved words, plus the number of EXEC SQL and EXEC CICS instructions.

## Parameters

The following guidelines apply when specifying parameters:

- The order of the parameters is not important.
- See the language definitions provided by SCLM for the actual usage of the parameters for FLMLRASM.

The following keyword parameters, separated by commas, are required as input to FLMLRASM:

### LISTINFO

Pointer to the SCLM list information record. This parameter is required and must be set to @@FLMLIS.

## LISTSIZE

The size of the LISTINFO buffer. This parameter is required and must be set to @@FLMSIZ. The parser checks to make sure that the LISTSIZE parameter is large enough to hold at least one entry of 228 bytes.

## STATINFO

Pointer to the SCLM statistics information record. This parameter is required and must be set to @@FLMSTP.

## Return Codes

---

0

**Explanation:** Indicates successful completion.

**User response:** None.

**Project manager response:** None.

---

4

**Explanation:** The dependency name does not match the source code for one of the following reasons:

- The dependency is greater than 8 characters. The error message in *userid.SCLMERR.LISTING* is:

```
4
line
dependency
```

where *line* is the source line that contains the dependency, and *dependency* is the dependency that exceeded 8 characters.

- The dependency name flows into column 72.

The error message in *userid.SCLMERR.LISTING* is:

```
4
line
dependency
```

where *line* is the source line that contains the dependency, and *dependency* is the dependency that extends into column 72. *dependency* will be positioned under its occurrence in *line* to show that it is too far over in the source file.

- The dependency name after a COPY is prefixed by an ampersand (&).

The error message in *userid.SCLMERR.LISTING* is:

```
4
line
&
```

where *line* is the source line that contains the dependency that begins with an ampersand, and & is printed under its occurrence in *line*.

- Mismatched quotes - a single or double quote was found that did not have a matching closing quote.

The error message in *userid.SCLMERR.LISTING* is:

```
4
mark line_no
```

where *mark* is either a single or double quotation mark, and *line\_no* is the line number that contains the unmatched quotation mark.

The dependency is not added to the list. Processing continues.

**User response:** Change the syntax to fit the parser.

**Project manager response:** None.

---

8

**Explanation:** The number of parsed dependencies exceeds the size of the \$list\_info array, which is specified by the BUFSIZE parameter on the FLMLANGL macro.

The error message in *userid.SCLMERR.LISTING* is:

```
8
line
dependency
```

where *line* is the source line that contains the dependency, and *dependency* is the dependency that exceeded the space allocated for the list.

**User response:** Either reduce the number of parsed dependencies for the member or contact the project manager.

**Project manager response:** Increase the buffer size (BUFSIZE=) on the FLMLANGL macro for the appropriate language definition. Be sure that LISTSIZE on the FLMTRNSL macro is set to @@FLMSIZ. Reassemble and relink the project definition.

---

10

**Explanation:** FLMTRNSL parameters are incorrect or are not specified.

The error message in *userid.SCLMERR.LISTING* is:

```
10
bad_parms
```

where *bad\_parms* are the parameters that are incorrect or not specified.

**User response:** Contact the project manager.

**Project manager response:** Verify that the FLMTRNSL

parameters for the FLMLRASM parser are valid and complete.

---

12

**Explanation:** Issued by TSOLNK; the parser was not found in the data set specified on the DSNAME parameter of the FLMTRNSL macro.

**User response:** Contact the project manager.

**Project manager response:** Verify that the value of the DSNAME parameter on the FLMTRNSL macro is correct.

---

16

**Explanation:** Error opening the error listings file.

**User response:** Contact the project manager.

**Project manager response:** Ensure that user has the authority to create and write to the file userid.SCLMERR.LISTING.

---

20

**Explanation:** Error closing the source file.

**User response:** Contact the project manager.

**Project manager response:** Contact your IBM service representative.

---

22

**Explanation:** An I/O error occurred in the DCB while reading input. Processing stops.

**User response:** Contact the project manager.

**Project manager response:** Contact your IBM service representative. Verify that the SOURCE DDNAME is allocated correctly. Verify that the data set and member to parse exist. Verify that the FLMALLOC macro is complete and valid for the parser.

---

40

**Explanation:** SCLM was not successful in invoking FLMLRASM using IKJEFTSR (TSOLNK).

**User response:** Contact the project manager.

**Project manager response:** FLMLRASM does not return a 40. A 40 can be encountered from SCLM for CALLMETH=TSOLNK. TSOLNK is used for executing interpretive REXX. A 40 means IKJEFTSR (TSOLNK) was not successful.

---

# FLMLRCBL REXX COBOL Parser

## Purpose

This is the COBOL parser translator, written in REXX, that parses the source identified by the SOURCE DDNAME.

## Functions

One of the functions of an SCLM parser is to determine a module's dependencies. FLMLRCBL determines the names of dependencies that are to be copied into the COBOL source.

The parser uses the following syntax rules to locate dependency names:

- The search for tokens is restricted to columns 8 -72. Column 7 is ignored except when it contains '\*' or '/' (treated as a comment line) or '-' (treated as a concatenation). The use of '-' to concatenate strings for forming reserved words or dependency names is not supported.
- When an uncommented, noncontinuation line has COPY after column 7, the next token is taken as the name of a dependency. Note the following exceptions.  
If the member name is enclosed in single or double quotes, the quotes are ignored and the member name is taken as the name of a dependency.
- When the 3 tokens EXEC, SQL, and INCLUDE are found in order on 1 or more uncommented lines after column 7, with no intervening text, the next token is taken as the name of a dependency. Note the following exceptions.  
If the member name is enclosed in single or double quotes, the quotes are ignored and the member name is taken as the name of a dependency.
- When searching for the next token on a line and there are no more tokens on that line, the search continues with the next uncommented line.
- Tokens inside quoted strings will be ignored, except for quoted member names following COPY statements. Reserved words inside quoted strings and comments will not be counted as statements.
- FLMLRCBL recognizes COPY or EXEC SQL INCLUDE anywhere in the source file (as long as they are not in quotation marks or comments).

Multiple COPY or EXEC SQL INCLUDE statements on any line or continued line are recognized.

The following example illustrates conditions under which dependencies will and will not be formed. Each MEMBER# token appears in an example of syntax that the parser recognizes as creating a dependency. A MEMBER# token must be from 1 to 8 characters. The BADCPY1 and BADCOPY02 statements in the example will not create dependencies for the following reasons:

- BADCPY1 and the COPY preceding it are inside a quoted string and are therefore ignored.
- BADCOPY02 is longer than 8 characters.

```
123456*-<-Column 7                               Column 72->
001010 FD TEST-FILE COPY MEMBER1.
001200 01 I-O-CNTL .      COPY 'MEMBER2'
001201 01 I-O-CNTL      COPY "MEMBER3" .
001201 01 LABEL PIC X VALUE 'EXTRA COPY
001201-                                     BADCPY1 '.
001202 EXEC SQL INCLUDE MEMBER4
001202 EXEC SQL INCLUDE 'MEMBER5'
001202 EXEC SQL INCLUDE "MEMBER6"
001300      COPY
```

```

001300* copy across a comment line
001300          MEMBER7.
001400 01 TESTNAMX COPY MEMBER8 . COPY MEMBER9.
001401  77 TESTNAME . COPY BADCOPY02.

```

Another function of the parser is to gather statistics or metrics for each module to be parsed. SCLM saves 10 statistics; only 7 are generated by this parser. For COBOL, this parser defines the following 10 statistics:

Total lines	The total number of records in the file.																																													
Comment lines	The number of lines with a slash (/) or an asterisk (*) in column 7.																																													
Noncomment lines	The number of total lines minus the number of comment lines.																																													
Blank lines	The number of lines that contain only blanks after column 6. Any sequence numbers in the rightmost columns of the line are ignored.																																													
Prolog lines	The number of comment lines and blank lines that are found before the first noncomment line.																																													
Total statements	The number of the following reserved words that appear on an uncommented line after column 7: <table border="0" style="margin-left: auto; margin-right: auto;"> <tr><td>ACCEPT</td><td>EVALUATE</td><td>PERFORM</td></tr> <tr><td>ADD</td><td>EXAMINE</td><td>READ</td></tr> <tr><td>ALTER</td><td>EXIT</td><td>RELEASE</td></tr> <tr><td>CALL</td><td>GO</td><td>RETURN</td></tr> <tr><td>CANCEL</td><td>GOBACK</td><td>REWRITE</td></tr> <tr><td>CLOSE</td><td>GOTO</td><td>SEARCH</td></tr> <tr><td>COMPUTE</td><td>IF</td><td>SET</td></tr> <tr><td>CONTINUE</td><td>INITIALIZE</td><td>SORT</td></tr> <tr><td>COPY</td><td>INSPECT</td><td>START</td></tr> <tr><td>DELETE</td><td>MERGE</td><td>STOP</td></tr> <tr><td>DISPLAY</td><td>MOVE</td><td>STRING</td></tr> <tr><td>DIVIDE</td><td>MULTIPLY</td><td>SUBTRACT</td></tr> <tr><td>ENTER</td><td>NOTE</td><td>TRANSFORM</td></tr> <tr><td>ENTRY</td><td>ON</td><td>UNSTRING</td></tr> <tr><td></td><td>OPEN</td><td>WRITE</td></tr> </table>	ACCEPT	EVALUATE	PERFORM	ADD	EXAMINE	READ	ALTER	EXIT	RELEASE	CALL	GO	RETURN	CANCEL	GOBACK	REWRITE	CLOSE	GOTO	SEARCH	COMPUTE	IF	SET	CONTINUE	INITIALIZE	SORT	COPY	INSPECT	START	DELETE	MERGE	STOP	DISPLAY	MOVE	STRING	DIVIDE	MULTIPLY	SUBTRACT	ENTER	NOTE	TRANSFORM	ENTRY	ON	UNSTRING		OPEN	WRITE
ACCEPT	EVALUATE	PERFORM																																												
ADD	EXAMINE	READ																																												
ALTER	EXIT	RELEASE																																												
CALL	GO	RETURN																																												
CANCEL	GOBACK	REWRITE																																												
CLOSE	GOTO	SEARCH																																												
COMPUTE	IF	SET																																												
CONTINUE	INITIALIZE	SORT																																												
COPY	INSPECT	START																																												
DELETE	MERGE	STOP																																												
DISPLAY	MOVE	STRING																																												
DIVIDE	MULTIPLY	SUBTRACT																																												
ENTER	NOTE	TRANSFORM																																												
ENTRY	ON	UNSTRING																																												
	OPEN	WRITE																																												
	In addition, any EXEC SQL and EXEC CICS statements are treated as program statements.																																													
Comment statements	This value is always 0.																																													
Control statements	This value is always 0.																																													
Assignment statements	This value is always 0.																																													
Noncomment statements	This is the same as total statements.																																													

## Parameters

Use the following guidelines to specify parameters:

- The order of the parameters is not important.
- See the language definitions provided by SCLM for the actual usage of the parameters for FLMLRCBL.

The following keyword parameters, separated by commas, are required as input for FLMLRCBL:

### LISTINFO

Pointer to the SCLM list information record. This parameter is required and must be set to @@FLMLIS.

### LISTSIZE

The size of the LISTINFO buffer. This parameter is required and must be set to @@FLMSIZ. The parser checks to make sure that the LISTSIZE parameter is large enough to hold at least one entry of 228 bytes.

### STATINFO

Pointer to the SCLM statistics information record. This parameter is required and must be set to @@FLMSTP.

## Return Codes

---

0

**Explanation:** Indicates successful completion.

**User response:** None.

**Project manager response:** None.

---

4

**Explanation:** The dependency name does not match the source code for one of the following reasons:

- Dependency name greater than 8 characters The error message in *userid.SCLMERR.LISTING* is:

```
4
line
dependency
```

where *line* is the source line that contains the dependency, and *dependency* is the dependency that exceeded 8 characters. The dependency is not added to the dependency list.

- Mismatched quotes. A single or double quote did not have a matching closing quote.

The error message in *userid.SCLMERR.LISTING* is:

```
4
mark line_no
```

where *mark* is either a single or double quotation mark that has no matching closing quote, and *line\_no* is the line number of the line that contains the unmatched quotation mark.

The dependency is not added to the list. Processing continues.

**User response:** Change the syntax to fit the parser.

**Project manager response:** None.

---

8

**Explanation:** The number of parsed dependencies exceeds the size of the \$list\_info array, which is specified by the BUFSIZE parameter on the FLMLANGL macro.

The error message in *userid.SCLMERR.LISTING* is:

8  
dependency

where *dependency* is the dependency that exceeded the space allocated for the list.

**User response:** Either reduce the number of parsed dependencies for the member or contact the project manager.

**Project manager response:** Increase the buffer size (BUFSIZE=) on the FLMLANGL macro for the appropriate language definition. Be sure that LISTSIZE on the FLMTRNSL macro is set to @@FLMSIZ. Reassemble and relink the project definition.

---

10

**Explanation:** FLMTRNSL parameters are incorrect or are not specified.

The error message in *userid.SCLMERR.LISTING* is:

```
10
bad_parms
```

where *bad\_parms* are the parameters that are incorrect or not specified.

**User response:** Contact the project manager.

**Project manager response:** Verify that the FLMTRNSL parameters for the FLMLRCBL parser are valid and complete.

---

12

**Explanation:** Issued by TSOLNK; the parser was not found in the data set specified on the DSNAME parameter of the FLMTRNSL macro.

**User response:** Contact the project manager.

**Project manager response:** Verify that the value of the DSNAME parameter on the FLMTRNSL macro is correct.

---

16

**Explanation:** Error opening the error listings file.

**User response:** Contact the project manager.

**Project manager response:** Ensure that user has the authority to create and write to the file userid.SCLMERR.LISTING.

---

20

**Explanation:** Error closing the source file.

**User response:** Contact the project manager.

**Project manager response:** Contact your IBM service representative.

---

22

**Explanation:** An I/O error occurred in the DCB while reading input. Processing stops.

**User response:** Contact the project manager.

**Project manager response:** Verify that the SOURCE DDNAME is allocated correctly. Verify that the data set and member to parse exist. Verify that the FLMALLOC macro is complete and valid for the parser. Contact your IBM service representative.

---

40

**Explanation:** SCLM was not successful in invoking FLMLRCBL using IKJEFTSR (TSOLNK).

**User response:** Contact the project manager.

**Project manager response:** FLMLRCBL does not return a 40. A 40 can be encountered from SCLM for CALLMETH=TSOLNK. TSOLNK is used for executing interpretive REXX. A 40 means IKJEFTSR (TSOLNK) was not successful.

---

## FLMLRCIS MVS C/C++ parser with include set support

### Purpose

The FLMLRCIS parser supports MVS C and C++ source files. The parser is written in REXX. The includes found by the parser are associated with an include set (See FLMINCLS macro.) that is the set name from the include statement.

### Functions

The parser uses the following syntax rules to locate dependency names:

- The search for tokens is restricted to uncommented text.  
The character strings `/*` and `*/` are recognized as comment delimiters that can span lines. The character string `//` is recognized as a begin comment token where the comment ends at the end of the line.
- Include dependencies are generated when the first token on the line is `#include`.  
The dependency consists of the member or include name and the include set name in the format `'member.set'`, where `set` is the include set name. It can be surrounded by double quotes (`"member.set"`) or by angle brackets (`<member.set>`).
- Tokens inside of strings are ignored.

The following table illustrates how include and include-set names are derived from source statements.

Source statement	Include name	Include-set name
<code>#include "abc"</code>	abc	
<code>#include "abc.h"</code>	abc	h

Another function of the parser is to gather statistics or metrics for each module to be parsed. SCLM saves 10 statistics, but only 4 are generated by this parser. This parser defines the ten statistics as follows:

Total lines	The total number of records in the file.
Comment lines	This value is always 0.
Noncomment lines	This is the same as the total lines.
Blank lines	The number of lines that contain only blanks.
Prolog lines	This value is always 0.
Total statements	This value is always 0.
Comment statements	The total number of <code>/* */</code> pairs in the member.
Control statements	This value is always 0.
Assignment statements	This value is always 0.
Noncomment statements	This value is always 0.

### Parameters

The following guidelines apply when specifying parameters:

- The order of the parameters is not important.
- See the language definition provided by SCLM for the actual use of the parameters for FLMLRCIS.

The following keyword parameters, separated by commas, are required as input to FLMLRCIS:

#### LISTINFO

Pointer to the SCLM list information record. This parameter is required and must be set to @@FLMLIS.

#### LISTSIZE

The size of the LISTINFO buffer. This parameter is required and must be set to @@FLMSIZ. The parser checks to make sure that the LISTSIZE parameter is large enough to hold at least one entry of 228 bytes.

#### STATINFO

Pointer to the SCLM statistics information record. This parameter is required and must be set to @@FLMSTP.

## Return Codes

---

0

**Explanation:** Indicates successful completion.

**User response:** None.

**Project manager response:** None.

---

4

**Explanation:** The dependency name does not match the source code for one of the following reasons:

- The include name is greater than 8 characters. The error message in *userid.SCLMERR.LISTING* is:

```
4
line
dependency
```

where *line* is the source line that contains the dependency, and *dependency* is the dependency that exceeded eight characters. The truncated form of the dependency is added to the dependency list.

- Unsupported form of include dependency. The error message in *userid.SCLMERR.LISTING* is:

```
4
line
dependency
```

where *line* is the source line that contains the dependency, and *dependency* is the text of the desired include member.

- #include statement spans multiple lines. The error message in *userid.SCLMERR.LISTING* is:

```
4
line
mark
```

where *line* is the source line that contains the mark, and *mark* is the mark (either a quotation mark or an angle bracket) that has no matching pair on the line.

- Mismatched quotes. A single or double quote was found that did not have a matching closing quote. The error message in *userid.SCLMERR.LISTING* is:

```
4
mark line_no
```

where *line\_no* is the line number that contains the unmatched quotation mark, and *mark* is the either a single or double quotation mark.

Processing continues.

**User response:** Change the syntax to fit the parser.

**Project manager response:** None.

---

8

**Explanation:** The number of parsed dependencies exceeds the size of the \$list\_info array, which is specified by the BUFSIZE parameter on the FLMLANGL macro. The error message in *userid.SCLMERR.LISTING* is:

```
8
line
dependency
```

where *line* is the source line that contains the dependency, and *dependency* is the dependency that exceeded the space allocated for the list.

**User response:** Either reduce the number of parsed dependencies for the member or contact the project manager.

**Project manager response:** Increase the buffer size (BUFSIZE=) on the FLMLANGL macro for the appropriate language definition. Be sure that LISTSIZE on the FLMTRNSL macro is set to @@FLMSIZ. Reassemble and relink the project definition.

---

10

**Explanation:** FLMTRNSL parameters are incorrect or are not specified.

The error message in *userid.SCLMERR.LISTING* is:

10  
bad\_parms

where *bad\_parms* are the parameters that are incorrect or not specified.

**User response:** Contact the project manager.

**Project manager response:** Verify that the FLMTRNSL parameters for the FLMLRC2 parser are valid and complete.

---

12

**Explanation:** Issued by TSOLNK; the parser was not found in the data set specified on the DSNAME parameter of the FLMTRNSL macro.

**User response:** Contact the project manager.

**Project manager response:** Verify that the value of the DSNAME parameter on the FLMTRNSL macro is correct.

---

16

**Explanation:** Error opening the error listings file.

**User response:** Contact the project manager.

**Project manager response:** Ensure that user has the authority to create and write to the file *userid.SCLMERR.LISTING*.

---

20

**Explanation:** Error closing the source file.

**User response:** Contact the project manager.

**Project manager response:** Contact your IBM service representative.

---

22

**Explanation:** An I/O error occurred in the DCB while reading input. Processing stops.

**User response:** Contact the project manager.

**Project manager response:** Contact your IBM service representative. Verify that the SOURCE DDNAME is allocated correctly. Verify that the data set and member to parse exist. Verify that the FLMALLOC macro is complete and valid for the parser.

---

40

**Explanation:** SCLM was not successful in invoking FLMLRCIS using IKJEFTSR (TSOLNK).

**User response:** Contact the project manager.

**Project manager response:** FLMLRC2 does not return a 40. A 40 can be encountered from SCLM for

CALLMETH=TSOLNK. TSOLNK is used for executing interpretive REXX. A 40 means IKJEFTSR (TSOLNK) was not successful.

---

## FLMLRC2 C, C++, and Resource file parser for workstation source

### Purpose

The FLMLRC2 parser supports C, C++ and resource files. The parser is written in REXX. The includes found by the parser are associated with an include set (See FLMINCLS macro.) that is the extension from the include statement.

### Functions

The parser uses the following syntax rules to locate dependency names:

- The search for tokens is restricted to uncommented text.

The character strings `/*` and `*/` are recognized as comment delimiters that can span lines. The character string `//` is recognized as a begin comment token where the comment ends at the end of the line.

- Include dependencies are generated in the following conditions:

- The first token on the line is `#include`. The included file name can be surrounded by double quotes (`"file.ext"`) or by angle brackets (`<file.ext>`).
- Dependencies are generated for some resource compiler statements. The statements support options between the statement and the include name, so the include name is taken as the last token on the line. Some of these statements have a format for includes and a format that does not support includes. The parser only finds includes when the statement does not contain a comma. The following statements are recognized as include statements:

```
BITMAP
FONT
ICON
POINTER
RESOURCE
RCINCLUDE
DLGINCLUDE
```

- Tokens inside of strings are ignored.

Include names are generated after removing excess characters (all characters up to and including the rightmost directory separator character, if any, and all characters from the first `'` to the end of the file name). The default is `\`. Any underscore characters (`_`) or blanks are replaced by at-signs (`@`). Include names longer than eight characters are truncated to eight characters and a return code of 4 is issued. The include-set names are generated from the characters following the first `'` to the end of the file name. Include-set names are also truncated to eight characters and underscore characters and blanks are replaced by at-signs. The following table illustrates how include and include-set names are derived from source statements.

Source statement	Include name	Include-set name
<code>#include "abc"</code>	abc	
<code>#include "abc.h"</code>	abc	h
<code>ICON 97, 101, 10, 10, 0, 0</code>		
<code>ICON ID_WINDOW mahjongg.ico</code>	mahjongg	ico
<code>#include "my file.h"</code>	my@file	h

Another function of the parser is to gather statistics or metrics for each module to be parsed. SCLM saves 10 statistics, but only 4 are generated by this parser. This parser defines the ten statistics as follows:

Total lines	The total number of records in the file.
Comment lines	This value is always 0.
Noncomment lines	This is the same as the total lines.
Blank lines	The number of lines that contain only blanks.
Prolog lines	This value is always 0.
Total statements	This value is always 0.
Comment statements	The total number of /* */ pairs in the member.
Control statements	This value is always 0.
Assignment statements	This value is always 0.
Noncomment statements	This value is always 0.

## Parameters

The following guidelines apply when specifying parameters:

- The order of the parameters is not important.
- See the language definitions provided by SCLM for the actual use of the parameters for FLMLRC2.
- The directory separator character defaults to \.

The DIR\_SEPARATOR keyword parameter may be used to specify a directory separator character.

The following keyword parameters, separated by commas, are required as input to FLMLRC2:

### LISTINFO

Pointer to the SCLM list information record. This parameter is required and must be set to @@FLMLIS.

### LISTSIZE

The size of the LISTINFO buffer. This parameter is required and must be set to @@FLMSIZ. The parser checks to make sure that the LISTSIZE parameter is large enough to hold at least one entry of 228 bytes.

### STATINFO

Pointer to the SCLM statistics information record. This parameter is required and must be set to @@FLMSTP.

## Return Codes

0

**Explanation:** Indicates successful completion.

**User response:** None.

**Project manager response:** None.

4

line  
dependency

where *line* is the source line that contains the dependency, and *dependency* is the dependency that exceeded eight characters. The truncated form of the dependency is added to the dependency list.

- Unsupported form of include dependency. The error message in *userid.SCLMERR.LISTING* is:

4

line  
dependency

**Explanation:** The dependency name does not match the source code for one of the following reasons:

- Dependency name greater than 8 characters. The error message in *userid.SCLMERR.LISTING* is:

where *line* is the source line that contains the dependency, and *dependency* is the text of the desired include member.

- #include statement spans multiple lines. The error message in *userid.SCLMERR.LISTING* is:

```
4  
line  
mark
```

where *line* is the source line that contains the mark, and *mark* is the mark (either a quotation mark or an angle bracket) that has no matching pair on the line.

- Mismatched quotes. A single or double quote was found that did not have a matching closing quote. The error message in *userid.SCLMERR.LISTING* is:

```
4  
mark line_no
```

where *line\_no* is the line number that contains the unmatched quotation mark, and *mark* is the either a single or double quotation mark.

Processing continues.

**User response:** Change the syntax to fit the parser.

**Project manager response:** None.

---

8

**Explanation:** The number of parsed dependencies exceeds the size of the \$list\_info array, which is specified by the BUFSIZE parameter on the FLMLANGL macro. The error message in *userid.SCLMERR.LISTING* is:

```
8  
line  
dependency
```

where *line* is the source line that contains the dependency, and *dependency* is the dependency that exceeded the space allocated for the list.

**User response:** Either reduce the number of parsed dependencies for the member or contact the project manager.

**Project manager response:** Increase the buffer size (BUFSIZE=) on the FLMLANGL macro for the appropriate language definition. Be sure that LISTSIZE on the FLMTRNSL macro is set to @@FLMSIZ. Reassemble and relink the project definition.

---

10

**Explanation:** FLMTRNSL parameters are incorrect or are not specified.

The error message in *userid.SCLMERR.LISTING* is:

```
10  
bad_parms
```

where *bad\_parms* are the parameters that are incorrect or not specified.

**User response:** Contact the project manager.

**Project manager response:** Verify that the FLMTRNSL parameters for the FLMLRC2 parser are valid and complete.

---

12

**Explanation:** Issued by TSOLNK; the parser was not found in the data set specified on the DSNAME parameter of the FLMTRNSL macro.

**User response:** Contact the project manager.

**Project manager response:** Verify that the value of the DSNAME parameter on the FLMTRNSL macro is correct.

---

16

**Explanation:** Error opening the error listings file.

**User response:** Contact the project manager.

**Project manager response:** Ensure that user has the authority to create and write to the file *userid.SCLMERR.LISTING*.

---

20

**Explanation:** Error closing the source file.

**User response:** Contact the project manager.

**Project manager response:** Contact your IBM service representative.

---

22

**Explanation:** An I/O error occurred in the DCB while reading input. Processing stops.

**User response:** Contact the project manager.

**Project manager response:** Contact your IBM service representative. Verify that the SOURCE DDNAME is allocated correctly. Verify that the data set and member to parse exist. Verify that the FLMALLOC macro is complete and valid for the parser.

---

40

**Explanation:** SCLM was not successful in invoking FLMLRC2 using IKJEFTSR (TSOLNK).

**User response:** Contact the project manager.

**Project manager response:** FLMLRC2 does not return a 40. A 40 can be encountered from SCLM for CALLMETH=TSOLNK. TSOLNK is used for executing interpretive REXX. A 40 means IKJEFTSR (TSOLNK) was not successful.

---

## FLMLRC37 REXX C370 Parser

### Purpose

This is the C/370 parser translator, written in REXX, that parses the source identified by the SOURCE DDNAME.

### Functions

One of the functions of an SCLM parser is to determine all of a module's dependencies. FLMLRC37 determines all of the names that will be copied into the C/370 source.

The parser uses the following syntax rules to locate dependency names:

- The search for tokens is restricted to uncommented text.
- When an uncommented line has #INCLUDE as the first token, followed by a token enclosed in double quotes ("MEMBER") or angle brackets (<MEMBER>), the enclosed token is accepted as the name of a dependency. Note the following exceptions.
  - When an uncommented line has EXEC, SQL, and INCLUDE as its first three tokens, the next token is accepted as the name of a dependency.
  - Tokens inside of strings or comments are ignored. /\* \*/ pairs are recognized as comment delimiters by the FLMLRC37 parser. Lines starting with // are also recognized as comments.

Dependencies are generated after removing excess characters (all characters up to and including the rightmost /, if any, and all characters from the first period (.) to the end of the file name). Any underscore characters (\_) are replaced by at sign characters (@). Dependency names longer than 8 characters are truncated to 8 characters and a return code of 4 is issued. The following table illustrates how dependencies are derived from include directives.

#include Directive	Dependency Generated	Return Code
#include "abc"	ABC	0
#include <sys/abc/xx>	XX	0
#include "Sys/abc/xx.h"	XX	0
#include <sys/name_1>	NAME@1	0
#include "Name2/App1.App2"	APP1	0
#include "xx.h.a"	XX	0
#include <DD:PLAN(YEAR)>	NONE	4
#include <'USER.SRC.MYINCS'>	NONE	4
#include "abc456789"	ABC45678	4

The following example further illustrates conditions under which dependencies will and will not be formed. Each MEMBER# token appears in an example of syntax that the parser recognizes as creating a dependency. The BADCPY# statements will not create dependencies for the following reasons:

- BADCPY1 is inside comment delimiters.
- BADCPY2 is not inside quotes or angle brackets.
- BADCPY3 is inside a string.

```

/* #include "badcpy1" */
#include "member1"
#include <member2>
#include badcpy2
EXEC SQL INCLUDE member3
printf '#include badcpy3'

```

Another function of the parser is to gather statistics or metrics for each module to be parsed. SCLM saves 10 statistics, but only 4 are generated by this parser. For C/370, this parser defines the ten statistics as follows:

Total lines	The total number of records in the file.
Comment lines	This value is always 0.
Noncomment lines	This is the same as the total lines.
Blank lines	The number of lines that contain only blanks.
Prolog lines	This value is always 0.
Total statements	This value is always 0.
Comment statements	The total number of /* */ pairs in the member.
Control statements	This value is always 0.
Assignment statements	This value is always 0.
Noncomment statements	This value is always 0.

## Parameters

The following guidelines apply when specifying parameters:

- The order of the parameters is not important.
- See the language definitions provided by SCLM for the actual use of the parameters for FLMLRC37.

The following keyword parameters, separated by commas, are required as input to FLMLRC37:

### LISTINFO

Pointer to the SCLM list information record. This parameter is required and must be set to @@FLMLIS.

### LISTSIZE

The size of the LISTINFO buffer. This parameter is required and must be set to @@FLMSIZ. The parser checks to make sure that the LISTSIZE parameter is large enough to hold at least one entry of 228 bytes.

### STATINFO

Pointer to the SCLM statistics information record. This parameter is required and must be set to @@FLMSTP.

## Return Codes

0

**Explanation:** Indicates successful completion.

**User response:** None.

**Project manager response:** None.

4

**Explanation:** The dependency name does not match the source code for one of the following reasons:

- Dependency name greater than 8 characters. The error message in *userid.SCLMERR.LISTING* is:

4  
line  
dependency

where *line* is the source line that contains the dependency, and *dependency* is the dependency that exceeded eight characters. The truncated form of the dependency is added to the dependency list.

- Unsupported form of include dependency. The error message in *userid.SCLMERR.LISTING* is:

4  
line  
dependency

where *line* is the source line that contains the dependency, and *dependency* is the text of the desired include member.

- #include statement spans multiple lines. The error message in *userid.SCLMERR.LISTING* is:

4  
line  
mark

where *line* is the source line that contains the mark, and *mark* is the mark (either a quotation mark or an angle bracket) that has no matching pair on the line.

- Mismatched quotes. A single or double quote was found that did not have a matching closing quote. The error message in *userid.SCLMERR.LISTING* is:

4  
mark line\_no

where *line\_no* is the line number that contains the unmatched quotation mark, and *mark* is the either a single or double quotation mark.

Processing continues.

**User response:** Change the syntax to fit the parser.

**Project manager response:** None.

---

8

**Explanation:** The number of parsed dependencies exceeds the size of the \$list\_info array, which is specified by the BUFSIZE parameter on the FLMLANGL macro. The error message in *userid.SCLMERR.LISTING* is:

8  
line  
dependency

where *line* is the source line that contains the dependency, and *dependency* is the dependency that exceeded the space allocated for the list.

**User response:** Either reduce the number of parsed dependencies for the member or contact the project manager.

**Project manager response:** Increase the buffer size

(BUFSIZE=) on the FLMLANGL macro for the appropriate language definition. Be sure that LISTSIZE on the FLMTRNSL macro is set to @@FLMSIZ. Reassemble and relink the project definition.

---

10

**Explanation:** FLMTRNSL parameters are incorrect or are not specified.

The error message in *userid.SCLMERR.LISTING* is:

10  
bad\_parms

where *bad\_parms* are the parameters that are incorrect or not specified.

**User response:** Contact the project manager.

**Project manager response:** Verify that the FLMTRNSL parameters for the FLMLRC37 parser are valid and complete.

---

12

**Explanation:** Issued by TSOLNK; the parser was not found in the data set specified on the DSNAME parameter of the FLMTRNSL macro.

**User response:** Contact the project manager.

**Project manager response:** Verify that the value of the DSNAME parameter on the FLMTRNSL macro is correct.

---

16

**Explanation:** Error opening the error listings file.

**User response:** Contact the project manager.

**Project manager response:** Ensure that user has the authority to create and write to the file *userid.SCLMERR.LISTING*.

---

20

**Explanation:** Error closing the source file.

**User response:** Contact the project manager.

**Project manager response:** Contact your IBM service representative.

---

22

**Explanation:** An I/O error occurred in the DCB while reading input. Processing stops.

**User response:** Contact the project manager.

**Project manager response:** Contact your IBM service representative. Verify that the SOURCE DDNAME is allocated correctly. Verify that the data set and member to parse exist. Verify that the FLMALLOC macro is

complete and valid for the parser.

---

**40**

**Explanation:** SCLM was not successful in invoking FLMLRC37 using IKJEFTSR (TSOLNK).

**User response:** Contact the project manager.

**Project manager response:** FLMLRC37 does not return a 40. A 40 can be encountered from SCLM for CALLMETH=TSOLNK. TSOLNK is used for executing interpretive REXX. A 40 means IKJEFTSR (TSOLNK) was not successful.

---

## FLMLRDTL REXX DTL Parser

### Purpose

This is the DTL Parser translator, written in REXX. Comments and split lines are not supported. The only recognized references are:

- <:entity inclname system> or
- <!entity inclname system> or
- <:entity % inclname system> or
- <!entity % inclname system>
- <?inclname>
- <?inclname otherstuff>

### Parameters

The following parameters are expected as input for the translator:

- Address of the storage to hold the list of included members
- Size of the storage to hold the list of included members.

### Return Codes

---

0

**Explanation:** Indicates successful completion.

**User response:** None.

**Project manager response:** None.

---

20

**Explanation:** Too many includes to fit in the storage provided.

**User response:** Increase the storage.

**Project manager response:** None.

---

# FLMLRIPF Script and OS/2 IPF Source Parser

## Purpose

The FLMLRIPF parser supports script and OS/2 ipf source files. The parser is written in REXX. The includes found by the parser are associated with an include set (See FLMINCLS macro.) that is the extension from the include statement.

## Functions

The parser uses the following syntax rules to locate dependency names:

- The search for tokens is restricted to uncommented text.  
Lines beginning with `.*` are recognized as comments.
- Include dependencies are generated in the following conditions:
  - The first token on the line is `.im`. The second token on the line is the include name. The include set will be the extension from the file name.
  - The first token on the line is `:artwork`. The token following `name=` is the include name.
- Tokens inside of strings are ignored.

Include names are generated after removing excess characters (all characters up to and including the far-right directory separator character (default is `\`), if any, and all characters from the first period (`.`) to the end of the file name). Any underscore characters (`_`) or blanks are replaced by at-signs (`'@'`). Include names longer than eight characters are truncated to eight characters and a return code of 4 is issued. The include-set names are generated from the characters following the first period (`.`) to the end of the file name. Include-set names are also truncated to eight characters and underscore characters and blanks are replaced by at-signs. The following table illustrates how include and include-set names are derived from source statements.

Source statement	Include name	Include-set name
<code>.im abc</code>	abc	
<code>:artwork name='tile_c_1.bmp' runin.</code>	tile@c@1	bmp

Another function of the parser is to gather statistics or metrics for each module to be parsed. SCLM saves 10 statistics, but only 4 are generated by this parser. This parser defines the ten statistics as follows:

Total lines	The total number of records in the file.
Comment lines	This value is always 0.
Noncomment lines	This is the same as the total lines.
Blank lines	The number of lines that contain only blanks.
Prolog lines	This value is always 0.
Total statements	This value is always 0.
Comment statements	The total number of <code>/* */</code> pairs in the member.
Control statements	This value is always 0.
Assignment statements	This value is always 0.
Noncomment statements	This value is always 0.

## Parameters

The following guidelines apply when specifying parameters:

- The order of the parameters is not important.
- See the language definitions provided by SCLM for the actual use of the parameters for FLMLRIPF.
- The directory separator character defaults to \.

The DIR\_SEPARATOR keyword parameter may be used to specify a directory separator character.

The following keyword parameters, separated by commas, are required as input to FLMLRIPF:

### LISTINFO

Pointer to the SCLM list information record. This parameter is required and must be set to @@FLMLIS.

### LISTSIZE

The size of the LISTINFO buffer. This parameter is required and must be set to @@FLMSIZ. The parser checks to make sure that the LISTSIZE parameter is large enough to hold at least one entry of 228 bytes.

### STATINFO

Pointer to the SCLM statistics information record. This parameter is required and must be set to @@FLMSTP.

## Return Codes

---

0

**Explanation:** Indicates successful completion.

**User response:** None.

**Project manager response:** None.

---

4

**Explanation:** The dependency name does not match the source code for one of the following reasons:

- Dependency name greater than 8 characters. The error message in *userid.SCLMERR.LISTING* is:

```
4
line
dependency
```

where *line* is the source line that contains the dependency, and *dependency* is the dependency that exceeded eight characters. The truncated form of the dependency is added to the dependency list.

- Unsupported form of include dependency. The error message in *userid.SCLMERR.LISTING* is:

```
4
line
dependency
```

where *line* is the source line that contains the dependency, and *dependency* is the text of the desired include member.

- #include statement spans multiple lines. The error message in *userid.SCLMERR.LISTING* is:

```
4
line
mark
```

where *line* is the source line that contains the mark, and *mark* is the mark (either a quotation mark or an angle bracket) that has no matching pair on the line.

- Mismatched quotes. A single or double quote was found that did not have a matching closing quote. The error message in *userid.SCLMERR.LISTING* is:

```
4
mark line_no
```

where *line\_no* is the line number that contains the unmatched quotation mark, and *mark* is the either a single or double quotation mark.

Processing continues.

**User response:** Change the syntax to fit the parser.

**Project manager response:** None.

---

8

**Explanation:** The number of parsed dependencies exceeds the size of the \$list\_info array, which is specified by the BUFSIZE parameter on the FLMLANGL macro. The error message in *userid.SCLMERR.LISTING* is:

8  
line  
dependency

where *line* is the source line that contains the dependency, and *dependency* is the dependency that exceeded the space allocated for the list.

**User response:** Either reduce the number of parsed dependencies for the member or contact the project manager.

**Project manager response:** Increase the buffer size (BUFSIZE=) on the FLMLANGL macro for the appropriate language definition. Be sure that LISTSIZE on the FLMTRNSL macro is set to @@FLMSIZ. Reassemble and relink the project definition.

---

10

**Explanation:** FLMTRNSL parameters are incorrect or are not specified.

The error message in *userid.SCLMERR.LISTING* is:

```
10  
bad_parms
```

where *bad\_parms* are the parameters that are incorrect or not specified.

**User response:** Contact the project manager.

**Project manager response:** Verify that the FLMTRNSL parameters for the FLMLRIPF parser are valid and complete.

---

12

**Explanation:** Issued by TSOLNK; the parser was not found in the data set specified on the DSNAME parameter of the FLMTRNSL macro.

**User response:** Contact the project manager.

**Project manager response:** Verify that the value of the DSNAME parameter on the FLMTRNSL macro is correct.

---

16

**Explanation:** Error opening the error listings file.

**User response:** Contact the project manager.

**Project manager response:** Ensure that user has the authority to create and write to the file *userid.SCLMERR.LISTING*.

---

20

**Explanation:** Error closing the source file.

**User response:** Contact the project manager.

**Project manager response:** Contact your IBM service representative.

---

22

**Explanation:** An I/O error occurred in the DCB while reading input. Processing stops.

**User response:** Contact the project manager.

**Project manager response:** Contact your IBM service representative. Verify that the SOURCE DDNAME is allocated correctly. Verify that the data set and member to parse exist. Verify that the FLMALLOC macro is complete and valid for the parser.

---

40

**Explanation:** SCLM was not successful in invoking FLMLRIPF using IKJEFTSR (TSOLNK).

**User response:** Contact the project manager.

**Project manager response:** FLMLRIPF does not return a 40. A 40 can be encountered from SCLM for CALLMETH=TSOLNK. TSOLNK is used for executing interpretive REXX. A 40 means IKJEFTSR (TSOLNK) was not successful.

---

# FLMLSS General Purpose Parser

## Purpose

This translator provides an interface to the general purpose SYNTRAN parser. Parsing criteria are provided to this translator through tables.

General information:

- Comments and the contents of quoted strings are ignored.
- DBCS strings (delimited by shift-out and shift-in characters) in comments and quotes are allowed.
- Total lines and blank lines are always counted.
- Control statements and assignment statements are always set to zero.
- The FLMLSS FLMPC370 parser is not case-sensitive.
- Dependencies are truncated to 8 characters before being added to the dependency list.
- Dependencies are ONLY found if they are outside comments. Comment syntax for each table is listed below. No other comment syntax is supported.

**Note:** This comment syntax does not match that allowed by some compilers.

Table Name	Syntax
FLMPALST	* indicates a comment that ends at the end of the line.
FLMPBOOK	.* or .CM in column 1 or following a semicolon (;) indicates a comment that ends at the end of the line.
FLMC370	<p>/* or */ turns comments on or off. These delimiters are used interchangeably.</p> <p>In the following example, bold text indicates areas considered to be comments by the FLMLSS parser.</p> <pre>/* <b>Comment 1</b> */ #include &lt;i1&gt; /* <b>Comment 2</b> */ #include &lt;i2&gt; /* <b>Comment 3</b> /* <b>Comment 4</b> */ #include &lt;i3&gt; /*#include &lt;i4&gt;</pre> <p>In the example, include dependencies are found for i1, i2, and i4, but not for i3.</p>
FLMDBRM	No comments are processed.

Table Name	Syntax
FLMPJOV	<p>Two types of comments are supported. " turns the first type of comment on or off. % turns the second type of comment on or off. This allows for nesting of comments. Comments are allowed between the !COPY or !COMPOOL statement and the copy or compool name.</p> <p>In the following example, bold text indicates areas considered to be comments by the FLMLSS parser.</p> <pre> !COMPOOL %COMMENT1&gt;(' I1'); !COMPOOL "COMMENT2"(' I2'); !COMPOOL %COMMENT3(' I3'); !COMPOOL "COMMENT4(' I4'); !COMPOOL "COMMENT5%COMMENT6"(' I5'); !COMPOOL %COMMENT7"COMMENT8"%' I6'); !COMPOOL "COMMENT9%COMMENT10"%' I7'); !COMPOOL %COMMENT11"COMMENT12"(' I8'); </pre> <p>In the example, include dependencies are found for I1, I2, I5, and I6, but not for I3, I4, I7 or I8.</p>
FLMPPAS	<p>Two types of comments are supported. (* or *) turns the first type of comment on or off and is used interchangeably. /* or */ turns the second type of comment on or off and is used interchangeably. The two comment delimiters let you nest comments.</p> <p>In the following example, bold text indicates areas considered to be comments by the FLMLSS parser.</p> <pre> %INCLUDE I1; (<b>* COMMENT 1 *</b>) %INCLUDE I2; /* <b>COMMENT 2</b> */ %INCLUDE I3; (<b>* COMMENT 3</b> (<b>*</b> %INCLUDE I4; <b>*/ COMMENT 4</b> <b>*/</b> %INCLUDE I5; (<b>* COMMENT 5</b> <b>/*NESTED COMMENT 5</b> <b>*/ *</b>) %INCLUDE I6; /* <b>COMMENT 6</b> (<b>*NESTED COMMENT 6</b> <b>*</b>) <b>*/</b> %INCLUDE I7; (<b>* COMMENT 7</b> %<b>INCLUDE I8</b>; (<b>*COMMENT 8</b> %INCLUDE I9; <b>/* COMMENT 9</b> <b>/* COMMENT 10</b> <b>*/</b> %<b>INCLUDE I10</b>; </pre> <p>In the example, include dependencies are found for I1, I2, I3, I4, I5, I6, I7 and I9 but not for I8 and I10.</p>
FLMPSCRIP	<p>.* or .CM in column 1 or following a semicolon (;) indicates a comment that ends at the end of the line.</p>

## Parameters

The following keyword parameters are expected as input to FLMLSS:

### CONTIN

The column in which the continuation line indicator is set in the input file. If you specify 0 for this parameter, the parser will not concatenate continued lines. The default is column 72.

### EOLCOL

The maximum number of characters from each input line to be processed by the parser. The parser ignores any information past this point. The default is 0.

### LISTINFO

Pointer to the SCLM statistics information record. This parameter is required and must be set to @@FLMLIS.

**LISTSIZE**

The size of the listinfo buffer. This parameter is required and must be set to @@FLMSIZ.

**PTABLEDD**

The ddname assigned to the parser data set load module. This parameter is required.

**SOURCEDD**

The ddname assigned to the source file to be parsed. This parameter is required.

**STATINFO**

Pointer to the SCLM statistics information record. This parameter is required and must be set to @@FLMSTP.

**TBLNAME**

The name of the parser table load module. This parameter is required. The following tables are provided with the SCLM product:

<b>FLMPALST</b>	Architecture definition
<b>FLMPBOOK</b>	BookMaster
<b>FLMPC370</b>	C/370
<b>FLMPDBRM</b>	DBRM
<b>FLMPEDL</b>	Series/1 EDL
<b>FLMPJOV</b>	JOVIAL
<b>FLMPPAS</b>	PASCAL
<b>FLMPSCRIP</b>	Script
<b>FLMPS1A</b>	Series/1 assembler

## Return Codes

---

0

**Explanation:** Indicates successful completion.

**User response:** None.

**Project manager response:** None.

---

4

**Explanation:** Indicates a warning condition. The limit of 3000 characters for concatenated continuation lines has been exceeded in the input file. The parser ignores any information past the 3000-character limit, but will continue parsing with the next line that is not a continuation line.

**User response:** In order to remove this warning, modify the input file so that concatenated continuation lines will not exceed the 3000-character limit. If the information past the 3000-character limit is not important, there is no need to change the source file.

**Project manager response:** None.

---

8

**Explanation:** Indicates an error condition. The parser completed successfully, but detected a syntax error in the file being parsed.

**User response:** Check the input file for syntax errors.

---

**Project manager response:** None.

---

12

**Explanation:** Indicates an error condition. Unable to load the SCLM table for the parser.

**User response:** Contact the project manager.

**Project manager response:** Verify that the user has access to the table through proper library concatenations.

---

16

**Explanation:** An invalid input parameter was specified, or a required input parameter was not specified.

**User response:** Contact the project manager.

**Project manager response:** Verify that the input parameters are specified correctly in the FLMTRNSL macro that defines this parser. Reassemble the project definition. Verify that no errors occurred. Relink the project definition.

---

20

**Explanation:** The number of parsed dependencies exceeds the size of the \$list\_info array, which is

specified by the BUFSIZE parameter on the FLMLANGL macro.

**User response:** Either reduce the number of parsed dependencies for the member or contact the project manager.

**Project manager response:** Increase the buffer size (BUFSIZE=) on the FLMLANGL macro for the appropriate language definition. Be sure that LISTSIZE on the FLMTRNSL macro is set to @@FLMSIZ. Reassemble and relink the project definition.

---

24

**Explanation:** Indicates an error condition. The parser was unable to load the SCLM table (FLMTABLE) or the NLS table.

**User response:** Contact the project manager.

**Project manager response:** Verify that SCLM was installed correctly.

---

# FLMLTWST Workstation Build Translator

## Purpose

The FLMLTWST translator is used to perform compiles, links, or other services on an ISPF connected workstation. It cannot be used when ISPF is accessed from a web browser via the ISPF JAVA environment.

This translator is used for languages that have the source in SCLM and the compiler, linker, or other tools on the workstation. FLMLTWST uses the ISPF SELECT service to execute workstation commands and the FILEXFER ISPF service to transfer files between the host and the workstation. This section describes the FLMLTWST translator as supplied with the ISPF product.

The FLMLTWST translator is written in REXX to let the project manager customize it to fit the needs of the project and workstation tools.

The translator does the following tasks:

- Initialization  
Parses and validates the parameters.
- Reads action information  
The action definitions are read from the ddname specified by the **ACTINFODD** parameter. This information includes:
  - The names of actions that can be specified with the **ACTION** parameter
  - The workstation commands and parameters for each action
  - Message file names
  - The mapping between SCLM type names and workstation extensions and subdirectories
  - File transfer format (ASCII or BINARY).
- Gets user-specific information  
Gets information such as which directory to store the files in on the workstation and the response file name.
- Retrieves information from the build map  
Gets the list of source members, includes, compiler options, and outputs from the build map. The build map information is obtained by calling the FLMTBMAP translator. The following build map keywords are processed. All other keywords are ignored.

### **SINC, SINC\***

Members on SINC statements:

- Will be transferred to the workstation.
- Can be added to the workstation command depending on the workstation extension that the member's type is mapped to, and the parameter information specified for the workstation command in the FLMLTWST logic.

**Innn** Include members identified by Innn statements in the build map will be transferred to the workstation.

### **COMP, LIST, LMAP, LOAD, OBJ, OUTx**

Output members identified by these statements in the build map:

- Will be transferred from the workstation to the ddname associated with that output keyword.

- Can be added to the workstation command depending on the workstation extension that the member's type is mapped to, and the parameter information specified for the workstation command in the FLMLTWST logic.

**CMD** The processing of the CMD statement depends on the blank delimited keyword that follows the command statement. CMD statements that do not have one of the keywords listed below will cause an error.

#### PARMS

The string following this keyword will be added to the workstation command. If multiple CMD PARMS appear in the archdef, they will be added to the workstation command in the order they appear. Where the parameters appear in the command in relation to the other parameters (input and output files) is determined by the information in the setup part of the FLMLTWST translator.

There will be no separator character following the value of PARMS in the language definition. If a separator character is desired then one should appear in the PARMS keyword as the last character.

If CMD ACTION statements are present in the build map, the parameters apply only to the workstation command for the action that they follow. If they appear before any CMD ACTION statements, they apply to the workstation command for the action from the ACTION parameter on the FLMLTWST translator definition.

#### ACTION

The string following this keyword must be a valid action (see the ACTION parameter for this translator). You can use the ACTION keyword to have FLMLTWST issue multiple workstation commands. The first workstation command is the command from the action specified on the ACTION parameter.

The following example shows how to add a binary resource file to an .exe file by specifying ACTION=LINKEXE on the FLMTRNSL in the language definition and using an architecture member.

```
*
LKED EXE                * use the EXE language
*
CMD PARMS /O+ /Ss
*
KREF OBJ                * OBJBIN is generated by an OBJ keyword
*
INCLD SAMPLE C         * includes SAMPLE OBJBIN
*
INCLD COMMON C         * includes COMMON OBJBIN
*
LOAD SAMPLE EXEBIN
LMAP SAMPLE LMAP
*
CMD ACTION RCEXE       * Add resources to the .exe
*
KREF OUT1              * RESBIN is generated by an OUT1 keyword
*
INCLD SAMPLE RC        * includes SAMPLE RESBIN
*
```

This causes two workstation commands to be issued. First the sample.exe file is generated; then the resource compiler adds the resources in the sample.res file to the sample.exe file. The sample.exe file is not stored into SCLM until after the resource compiler has run.

- Generates a response file if needed.  
Workstation compilers and other tools that support response files will have one generated. The response file contains the parameters for the compiler or other tool.
- Transfers the inputs (source, includes, and so on) to the workstation.  
The source members, includes, and response file are transferred to the workstation using the FLMTXFER translator. If multiple workstation commands are being issued, the response files for all commands after the first will be transferred to the workstation just prior to issuing the command.
- Runs the workstation command.  
Constructs the workstation command and sends it to the workstation.
- Transfers the outputs (obj, exe, dll, and so on) to the host.  
Transfers the outputs to the host using the FLMTXFER translator. The SCLM outputs are placed in ddnames allocated by FLMALLOC so that build can place them into the hierarchy.

## Parameters

The following parameters are specified in the OPTIONS list for the FLMTRNSL macro that has COMPILE=FLMLTWST. All parameters are keyword parameters and can be specified in any order. Parameters must be separated by commas. Extraneous parameters are ignored without any messages being produced. An FLMALLOC is required for the following data definitions:

- MESSAGEDD
- MSGXFERDD
- RESPONSEDD
- FILESDD
- BMAPDD
- USERINFODD
- ACTINFODD

### **ACTION=COMPILE|action\_name**

This parameter is optional. If not specified, it defaults to COMPILE. The valid values are specified in the ACTINFO data set.

### **BMAPINFO=@@FLM\$MP**

This parameter is required and must be specified in the options list with the value from @@FLM\$MP.

### **BLDINFO=@@FLMBIO**

This parameter is required and must be specified in the options list with the value from @@FLMBIO.

### **SCLMINFO=@@FLMINF**

This parameter is required and must be specified in the options list with the value from @@FLMINF.

### **PARMS=command\_parms**

This parameter is optional. If specified, the string that follows it will be added as the first parameter on all workstation commands.

**MESSAGEDD=dd\_name**

This parameter is optional. If not specified, it defaults to MESSAGE. This is the ddname where messages are written.

**MSGXFERDD=dd\_name**

This parameter is optional. If not specified, it defaults to MSGXFER. This is the ddname to temporarily hold messages from the workstation command. The messages are appended to the data set specified by the MESSAGEDD parameter. The FLMALLOC for this ddname must specify CATLG=Y.

**RESPONSEDD=dd\_name**

This parameter is optional. If not specified, it defaults to RESPONSE. This is the ddname used to generate the response file for the workstation if the workstation command requires a response file. The FLMALLOC for this ddname must specify CATLG=Y.

**FILESDD=dd\_name**

This parameter is optional. If not specified, it defaults to FILES. This ddname is used to communicate between the FLMLTWST and FLMTXFER translators. See the description of the content of this ddname in the FLMTXFER translator description.

**BMAPDD=dd\_name**

This parameter is optional. If not specified, it defaults to BMAP. This ddname is used to communicate between the FLMLTWST and FLMTBMAP translators. See the description of the content of this ddname in the FLMTBMAP translator description.

**USERINFODD=dd\_name**

This parameter is optional. If not specified, it defaults to USERINFO. This ddname contains the information about the workstation where the tools will be run. Each line in the data set allocated to the ddname can contain either a comment or keyword and its value. Comment lines begin with an asterisk (\*) and are ignored. Lines that contain invalid keywords are ignored. Keywords and their values must be separated by one or more spaces.

**ACTINFODD=dd\_name**

This parameter is optional. If not specified, it defaults to ACTINFO. This ddname contains the information about the workstation actions such as COMPILE and LINKEXE. Each line in the data set allocated to the ddname contains either a comment or a keyword. Comment lines begin with an asterisk (\*) and are ignored. Lines that contain invalid keywords are ignored. Keywords on the statement must be separated by at least one space.

**output\_keyword=dd\_name**

These parameters can be used to specify the ddnames to hold the output for each build output keyword. These parameters are not required. If not specified, the ddname is the same as the keyword. An example of these parameters is OBJ=SYSIN,LIST=SYSPRINT. This example defines FLMALLOC statements for the SYSIN and SYSPRINT ddnames with IOTYPE=O or P. If these parameters had not been specified, there would be FLMALLOC statements for OBJ and LIST ddnames with IOTYPE=O or P.

The FLMALLOC statements for the output ddnames must specify CATLG=Y. All allocations must be IOTYPE=O or P. If CATLG=Y is not specified, the file transfer will fail.

**USERINFODD statements**

Valid keywords for statements in the USERINFODD dataset are:

Keyword	Value	Description
---------	-------	-------------

<b>Asterisk (*)</b>	Comment lines start with asterisks and are ignored.
<b>RESPONSE_FILE</b>	The name of the response file on the workstation. The file name must include the full path, because the DATA_DIR value will not be prefixed to the file name. The default is <i>response.fil</i> .
<b>DEL_CMD</b>	The delete command to be used on the workstation. The default is DEL. The message files that are to be created by a workstation command will first be deleted using this command. This is done so that message files with the same name from previous commands are not transferred to the host after completion of a workstation command.
<b>DATA_DIR</b>	The base directory where the files are stored on the workstation. DATA_DIR must include the full directory name. All SCLM members are transferred to and from this directory and its subdirectories. The subdirectories are based on the subdirectory value specified in the ACTINFO file. This defaults to '\'. The FLMLTWST translator supplied by IBM appends the subdirectory to the DATA_DIR value before appending the file name.
<b>MODE</b>	<p>MODE specifies how the command is to start on the workstation. MODE may be:</p> <ul style="list-style-type: none"> <li>• MIN (minimized - the default)</li> <li>• MAX (maximized)</li> <li>• VIS (visible, if possible)</li> <li>• INVIS (invisible, if possible).</li> </ul> <p>A MODE value specified in USERINFODD will override a MODE value specified in ACTINFODD. For more information on MODE, see the ISPF SELECT service in the ISPF Services Guide</p>
<b>WSDIR</b>	WSDIR specifies the workstation directory. This is the directory from which the workstation command will be executed. The default is the directory from which ISPF Client/Server is running. The WSDIR value from the ACTINFODD dataset will be concatenated after the WSDIR value from the USERINFODD dataset.

### **ACTINFODD statements**

Statements in the ACTINFODD file are composed of a keyword and a value. The keywords TYPE, EXTENSION, TRANSFER\_FORMAT, and SUBDIRECTORY are used to define the SCLM-type-to-workstation-file-extension mapping information to SCLM. The other keywords in this file are used to specify information about each workstation command. Keywords CPARM, EXTENSION, MESSAGE\_FILE, and RPARM apply only to the previous ACTION keyword. Valid keywords for statements in the ACTINFO dataset are:

<b>Keyword</b>	<b>Value Description</b>
<b>Asterisk (*)</b>	Comment lines start with asterisks and are ignored.
<b>WSDIR</b>	WSDIR specifies the workstation directory for executing the command. The WSDIR value from

	the ACTINFODD dataset will be concatenated after the WSDIR value from the USERINFODD dataset.
<b>QUOTE</b>	The character to use for quoting strings in CPARM, CPARMSEP, and RPARM statements. This character is used for statements that follow this QUOTE statement until the next QUOTE statement is found. The default quote character is a single quote (').
<b>ACTION</b>	The name of an action that can be specified on the ACTION parameter to FLMLTWST.
<b>COMMAND</b>	The command to be issued on the workstation for the previous action. If multiple COMMAND statements are found following an ACTION statement only the last COMMAND is used.
<b>MODE</b>	<p>MODE specifies how the command is to start on the workstation. MODE may be:</p> <ul style="list-style-type: none"> <li>• MIN (minimized - the default)</li> <li>• MAX (maximized)</li> <li>• VIS (visible, if possible)</li> <li>• INVIS (invisible, if possible).</li> </ul> <p>A MODE value specified in USERINFODD will override a MODE value specified in ACTINFODD. For more information on MODE, see the ISPF SELECT service in the ISPF Services Guide The default MODE is MIN.</p>
<b>CPARM</b>	<p>Parameters to add to the last workstation COMMAND. The parameters are added to the command in the order they are found in the file. Each parameter is made up of three parts: a prefix, a type name, and a suffix. The type name indicates that the parameters on this CPARM statement only apply to members in SCLM of that type. If no type name is specified, the parameter is added to the command. In the parameter string, the SCLM type name is replaced with the name of SCLM members of that type that are inputs or outputs to the build. The parameters added to the command are composed by concatenating the prefix, the workstation file name for the SCLM member, and the suffix. If multiple members match the type on the CPARM statement, the prefix and suffix are concatenated to each file name. See the examples at the end of the description of FLMLTWST for more information.</p> <ul style="list-style-type: none"> <li>• A prefix string. The string must be surrounded by quotes if it contains blanks.</li> <li>• An SCLM type name. If the type name is specified, the parameters are added if there is an input or output of this type to the command. If there are inputs and outputs of this type, the file name containing each input and output is added</li> </ul>

to the workstation command preceded by the prefix string and followed by the suffix string. No blanks are placed between the file name and the prefix and suffix strings. To get blanks between the prefix and suffix strings and the file name, use quotes around the strings and put the blank inside the quotes.

- A suffix string. The string must be surrounded by quotes if it contains blanks. This string defaults to blank if not specified.

The following variables can be specified in the prefix and suffix strings:

**&CMD\_PARMS&**

Is replaced with the parameters specified on "CMD PARMS" statements in the architecture definition if there are any. "Null" will be used when "CMD PARMS" is not found. &CMD\_PARMS& must be present in order to use a CMD PARMS statement in the architecture definition.

**&RESPONSE\_FILE&**

Is replaced with the response file name.

**&DATA\_DIR&**

Is replaced with the base directory from the user information.

**CPARMSEP**

The value to be used as a separator between the command parameter strings specified by the CPARM keywords. No separator character will be added to the end of the parameter string. Also, there will be no separator character following the value of PARMS in the language definition. If you want a separator character, then it should appear in the PARMS keyword as the last character.

- CPARMSEP NULL results in no separator character.
- CPARMSEP by itself results in a blank being used as the separator character.
- CPARMSEP with a quoted string will have the quotes removed from the front and back if the quote characters match the value of the QUOTE keyword. (If there is no QUOTE keyword the single quote default will be used.) An error message will appear if the first character is a quote (as per the value of the QUOTE keyword) and the last character is not a quote.
- No separator character will be added to the end of the parameter string.

**RPARM**

Parameters to add to the response file for the last workstation command. The format of this statement is the same as the CPARM statement. Use CPARM if the parameters are specified as part

of the workstation command. Use RPARM if the parameters are to be put in a response file that the workstation command will read. Only use response file parameters if the workstation command supports a response file. If parameters are specified in a response file, make sure the response file name is specified on one of the CPARM statements if the workstation command requires it.

The variables described for CPARM can also be used on RPARM statements.

**TYPE**

The name of an SCLM type to transfer to the workstation. The type name can include a single asterisk (\*) as a wild card character.

**EXTENSION**

The workstation extension to use for the types from the previous TYPE statement. A single asterisk (\*) can be included and is replaced with any characters that matched the \* in the TYPE statement. The default value is \*.

The value is either a single asterisk or a character string. Strings using an asterisk and other characters (such as H\*) will result in the asterisk being used as part of the extension.

**SUBDIRECTORY**

The subdirectory to use for the file names derived from the previous TYPE statement. The subdirectory is placed between the data directory as specified by the DATA\_DIR keyword in the USERINFODD user information and the file name to construct the fully-qualified workstation file name. The default value is a \.

**TRANSFER\_FORMAT**

The transfer format for files of the types from the previous TYPE statement. Valid values are:

**ASCII** Translate the file to ASCII format. This is the default.

**BINARY**

Perform no translation.

**WSCASE**

The case for workstation file names. Valid values are:

**UPPER**

Transfer the files to or from the workstation with the workstation file name in uppercase letters. This is the default setting.

**LOWER**

Transfer the files to or from the workstation with the workstation file name in lowercase letters.

**ULOWER**

Transfer the files to or from the workstation with the workstation file name having an initial capital letter followed by lowercase letters.

## MESSAGE\_FILE

The name of a message file that was created on the workstation. It will be written to the ddname specified by the MESSAGEDD parameter. This statement can be used to get the messages from the workstation command into the BUILD.LISTxx data set. Multiple MESSAGE\_FILE statements can be specified. Each MESSAGE\_FILE statement applies to the previous ACTION. The default message file name is c:\sclm.msg.

## Environment

The FLMTXFER translator must have access to ISPF services. FLMLTWST must be invoked by specifying CALLMETH=ISPLNK for the FLMTRNSL macro.

## Return Codes

In addition to the return codes listed here, messages can be written to the ddname specified by the MESSAGEDD parameter.

---

0

**Explanation:** The workstation build was successful.

**Project manager response:** None.

---

1-900, 904-908, >999

**Explanation:** Workstation command error.

**User response:** Review the messages from the workstation command. Refer to the ISPF SELECT service in the ISPF Services Guide for additional information about problems with executing the command.

**Project manager response:** None.

---

901

**Explanation:** The call to FLMTBMAP failed.

**User response:** Check the return code from FLMTBMAP.

**Project manager response:** None.

---

902

**Explanation:** The call to FLMTXFER to transfer the inputs to the workstation failed.

**User response:** Check the return code and messages from FLMTXFER. Messages are in the ddname specified by the MESSAGEDD parameter. Refer to the FILEXFER ISPF service for additional information about problems with transferring files.

**Project manager response:** None.

---

903

**Explanation:** The call to FLMTXFER to receive outputs from the workstation failed.

**User response:** Check the return code and messages from FLMTXFER. Messages are in the ddname specified by the MESSAGEDD parameter. Refer to the FILEXFER ISPF service for additional information about problems with transferring files.

**Project manager response:** None.

---

999

**Explanation:** An error occurred in FLMLTWST.

**User response:** Check the messages from FLMLTWST.

**Project manager response:** None.

---

## User Information Example

The following shows a sample USERINFO data set and can be found in the FLMWBUSR member in the ISPF sample library.

```
*
* Data_dir. Default is c:\.
* The subdirectory value from the actinfo file or the default of '\'
* will be appended to this value before the name of the workstation
* file. This directory and its subdirectories will be where SCLM
* transfers files to/from on the workstation.
*
data_dir      c:\wb
*
```

```

* Wsdir. Default is ISPF Client/Server directory.
* Directory from which the command will be executed. If a WSDIR
* keyword is also specified in the ACTINFO file, then it will be
* concatenated to the end of what is specified here.
* For commands that must run from the directory holding the data
* files (such as the resource compiler), this value should be the
* same as the data_dir value.
*
wsdir      c:\wb
*
* Response_file. Default is c:\response.fil.
* Fully qualified name of the response file to contain all RPARM
* statements for an action. The name should include the drive and
* any subdirectories.
*
response_file c:\wb\response.fil
*
* Mode. Workstation build default is minimized. This will override any
* value set in the ACTINFO file.
* This is the mode for the workstation command. Valid values are:
* MAXimized, MINimized, VISible and INVISible. See the WSCMDV section
* of the ISPF SELECT service for additional information.
*
mode      max
*
* Del_cmd. Default is 'del'.
* This keyword specifies the delete command to be used to remove the
* message file(s) from previous builds or build steps from the
* workstation before executing the workstation command.
*
del_cmd   erase

```

## ACTINFO Example

The following example shows an ACTINFO data set and can be found in the FLMWBAIO member in the ISPF sample library.

```

*****
*
* SCLM type to workstation file name mapping
*
* The following statements define the mapping between SCLM type names
* and workstation file extensions as well as the transfer format
* for the data. The statements are processed in order. If the
* member being processed does not match the first TYPE criteria, then
* it will be compared to the next TYPE criteria and so on until a
* match is found.
*
*****
*
* Types containing BINARY data
*
* This mapping indicates that the host SCLM type will be the
* workstation file extension followed by BIN. The BIN pattern in
* the type name will be used to indicate that members in these types
* contain binary data. As an example, a member stored in the OBJBIN
* type in SCLM, will be transferred as binary (no ASCII-to-EBCDIC
* conversion) with the workstation file name 'member.OBJ'.
*
*****
TYPE          *BIN
EXTENSION     .*
TRANSFER_FORMAT BINARY
SUBDIRECTORY  \
*****
*
* Types containing ASCII data
*

```

```

* This mapping indicates that any members whose type does not match *
* the previous criteria (TYPE * will always match), will be processed *
* as ascii/text files. The workstation file extension will be the *
* same as the SCLM type. As an example, a member in the CPP type in *
* SCLM will have the workstation file name 'member.CPP'. *
*
*****
TYPE *
EXTENSION .*
TRANSFER_FORMAT ASCII
SUBDIRECTORY \
*
*****
*
* Workstation Commands *
*
* The following statements define the actions/commands to be executed *
* on the workstation. *
*
*****
* C and C++ compile to generate object
*
ACTION COMPILE
COMMAND icc
RPARM -Fd
RPARM -c
RPARM -Gm+
RPARM -O+
RPARM &CMD_PARMS&
RPARM /Fo OBJBIN
RPARM /Fl LST
RPARM '' C
RPARM '' CPP
CPARM @&RESPONSE_FILE&
CPARM 1>&DATA_DIR&\stdout.msg
CPARM 2>&DATA_DIR&\stderr.msg
MESSAGE_FILE &DATA_DIR&\stdout.msg
MESSAGE_FILE &DATA_DIR&\stderr.msg
*
* Preprocessor only
*
ACTION PREPROCESS
COMMAND icc
RPARM /Pe+
RPARM &CMD_PARMS&
RPARM '' C
RPARM '' CPP
RPARM '' IPF
RPARM '' RC
CPARM @&RESPONSE_FILE&
CPARM 1>&DATA_DIR&\stdout.msg
CPARM 2>&DATA_DIR&\stderr.msg
MESSAGE_FILE &DATA_DIR&\stdout.msg
MESSAGE_FILE &DATA_DIR&\stderr.msg
*
* Dummy file for resource DLLs
* (compile with /Ge- option)
*
ACTION DUMMY
COMMAND icc
RPARM -Fd
RPARM -c
RPARM /Ge-
RPARM &CMD_PARMS&
RPARM /Fo OBJBIN

```

```

RPARAM /F1 LST
RPARAM '' C
RPARAM '' CPP
CPARM @&RESPONSE_FILE&
CPARM 1>&DATA_DIR&\stdout.msg
CPARM 2>&DATA_DIR&\stderr.msg
MESSAGE_FILE &DATA_DIR&\stdout.msg
MESSAGE_FILE &DATA_DIR&\stderr.msg
*
* Link object into exe using CSET++
*
ACTION      LINKEXE
COMMAND     icc
RPARAM /Ge+
RPARAM &CMD_PARMS&
RPARAM /Fe EXEBIN
RPARAM /Fm MAP
RPARAM '' OBJBIN
RPARAM '' DEF
CPARM @&RESPONSE_FILE&
CPARM 1>&DATA_DIR&\stdout.msg
CPARM 2>&DATA_DIR&\stderr.msg
MESSAGE_FILE &DATA_DIR&\stdout.msg
MESSAGE_FILE &DATA_DIR&\stderr.msg
*
* Link object into dll using CSET++
*
ACTION      LINKDLL
COMMAND     icc
RPARAM /Ge-
RPARAM &CMD_PARMS&
RPARAM /Fe DLLBIN
RPARAM /Fm MAP
RPARAM '' OBJBIN
RPARAM '' DEF
CPARM @&RESPONSE_FILE&
CPARM 1>&DATA_DIR&\stdout.msg
CPARM 2>&DATA_DIR&\stderr.msg
MESSAGE_FILE &DATA_DIR&\stdout.msg
MESSAGE_FILE &DATA_DIR&\stderr.msg
*
* Link object into exe or dll using LINK386
*
ACTION      LINK386
COMMAND     link386
CPARMSEP NULL
CPARM &CMD_PARMS&
CPARM '' OBJBIN
CPARM ,
CPARM '' DLLBIN
CPARM '' EXEBIN
CPARM ,
CPARM '' MAP
CPARM ,
CPARM '' LIBBIN
CPARM ,
CPARMSEP
CPARM '' DEF
CPARM 1>&DATA_DIR&\stdout.msg
CPARM 2>&DATA_DIR&\stderr.msg
MESSAGE_FILE &DATA_DIR&\stdout.msg
MESSAGE_FILE &DATA_DIR&\stderr.msg
*
* Generate res file
*
ACTION      RC
COMMAND     rc

```

```

CPARM -r
CPARM '-i &DATA_DIR&'
CPARM &CMD_PARMS&
CPARM '' RC
CPARM '' RESBIN
CPARM 1>&DATA_DIR&\stdout.msg
CPARM 2>&DATA_DIR&\stderr.msg
MESSAGE_FILE &DATA_DIR&\stdout.msg
MESSAGE_FILE &DATA_DIR&\stderr.msg
*
* Apply res file to an exe or dll
*
ACTION      RCEXE
COMMAND     rc
CPARM &CMD_PARMS&
CPARM '' RESBIN
CPARM '' EXEBIN
CPARM '' DLLBIN
CPARM 1>&DATA_DIR&\stdout.msg
CPARM 2>&DATA_DIR&\stderr.msg
MESSAGE_FILE &DATA_DIR&\stdout.msg
MESSAGE_FILE &DATA_DIR&\stderr.msg
*
* Generate hlp file
*
ACTION      IPFC
COMMAND     ipfc
CPARM '' IPF
CPARM &CMD_PARMS&
CPARM 1>&DATA_DIR&\stdout.msg
CPARM 2>&DATA_DIR&\stderr.msg
MESSAGE_FILE &DATA_DIR&\stdout.msg
MESSAGE_FILE &DATA_DIR&\stderr.msg
*
* Generate hlp file - input file specified on CMD statement
*
ACTION      IPFCP
COMMAND     ipfc
CPARM &CMD_PARMS&
CPARM 1>&DATA_DIR&\stdout.msg
CPARM 2>&DATA_DIR&\stderr.msg
MESSAGE_FILE &DATA_DIR&\stdout.msg
MESSAGE_FILE &DATA_DIR&\stderr.msg

```

The following example shows an architecture definition and the resulting workstation commands using the previous ACTINFO dataset. This architecture definition can be found in the ISPF sample library, member FLMWBSLE.

The architecture definition:

```

*
LKED EXE                * link language
*
KREF OBJ                * include generated object modules
*
INCLD MAHJONGG C        * MAHJONGG SOURCE
INCLD TILE C           * TILE SOURCE
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 which produces MAHJONGG RESBIN
*
```

The language EXE (as specified by the LKED keyword) uses the action LINKEXE. This results in the following command and response file:

The command:

```
icc @c:\wb\response.fil 1>c:\wb\stdout.msg 2>c:\wb\stderr.msg
```

The contents of the response file, c:\wb\response.fil:

```
/Ge+
/Fec:\wb\MAHJONGG.EXE
/Fmc:\wb\MAHJONGG.MAP
c:\wb\MAHJONGG.OBJ
c:\wb\TILE.OBJ
c:\wb\MAHJONGG.DEF
```

The CMD ACTION statement results in a second action, RCEXE. RCEXE issues the following command:

```
rc c:\wb\MAHJONGG.RES c:\wb\MAHJONGG.EXE 1>c:\wb\stdout.msg 2>c:\wb\stderr.msg
```

### **Language Definition Example**

The following example shows a language definition using FLMLTWST to compile C or C++ source on the workstation. This sample can be found in the ISPF macro library as member FLM@WICC.

```

*****
*
*   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 6. 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
          DFLTTY=OBJBIN,DFLTMEM=*,LANG=EXE
*      (* LIST *)
FLMALLOC  IOTYPE=O,RECFM=VB,LRECL=256,          C
          RECNUM=4000,DDNAME=LIST,CATLG=Y,PRINT=I,   C
          KEYREF=LIST,DFLTTY=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 6. C Language Definition (Part 2 of 2)

---

# FLMTBMAP Build Map Print - Build Translator

## Purpose

The FLMTBMAP translator generates a DBUTIL (Database Contents Utility) style printout of the information in the build map of the member being built. The information in the report matches the build map that will be saved if the build of the member is successful. The information in this report may be different from the information in the build map currently stored in VSAM.

Information about outputs that do not have the member name specified do not show up in the report. The output information will be missing in the following conditions:

- An architecture member is being built and the output is the result of an output keyword with an '\*' as the member name,
- A nonarchitecture member is being built and the output is the result of an FLMALLOC with IOTYPE=P and no DFLTMEM was specified.

For architecture members that include the outputs of other members through INCL or INCLD statements, the outputs that are included are identified by a SINC\* keyword before the output.

FLMTBMAP is a build translator and can be run only within the build environment.

For an example of the usage of this translator see the FLMLTWST translator.

## Parameters

All parameters are keyword parameters and can be specified in any order. Parameters must be separated by commas. Extraneous parameters are ignored without any messages being produced.

### **BMAPINFO=@@FLM\$MP**

This parameter is required and must be specified in the options list with the value from @@FLM\$MP.

### **SCLMINFO=@@FLMINF**

This parameter is required and must be specified in the options list with the value from @@FLMINF.

### **BLDINFO=@@FLMBIO**

This parameter is required and must be specified in the options list with the value from @@FLMBIO.

### **BMAPDD=dd\_name**

This parameter is optional. It specifies the ddname where the build map report will be written. If it is not specified, the FLMTBMAP translator will attempt to write the report to a ddname of BMAP. This parameter will be truncated to 8 characters.

## Return Codes

---

0

**Project manager response:** None.

**Explanation:** The report was generated successfully.

**User response:** None.

---

4

**Explanation:** The build map is empty, no report was produced.

**User response:** None.

**Project manager response:** None.

---

8

**Explanation:** The ddname for the report is not allocated.

**User response:** Contact the project manager.

**Project manager response:** Ensure that the language definition has an FLMALLOC for the ddname specified by the BMAPDD parameter.

---

12

**Explanation:** The value for BMAPINFO is not valid.

**User response:** Contact the project manager.

**Project manager response:** Ensure that the BMAPINFO parameter is specified in the options list, that the parameter has a value of @@FLM\$MP, and that the translator has FUNCTN=BUILD.

---

16

**Explanation:** The value for SCLMINFO is not valid.

**User response:** Contact the project manager.

**Project manager response:** Ensure that the SCLMINFO parameter is specified in the options list and that the parameter has a value of @@FLMINF.

---

# FLMTMSI Interface to SCRIPT/VS

## Purpose

This translator provides an interface to SCRIPT/VS via the TSO Service Facility.

SCRIPT is a TSO command and needs to be invoked by using the TSO command processor interface. It cannot be invoked directly by FLMTRNSL. FLMTMSI builds a SCRIPT command with a concatenation of the string following '/' in the OPTIONS list.

## Parameters

The options string passed to this translator should contain the user ID to be used by SCRIPT/VS, delimited by a slash (/), and followed by a list of desired SCRIPT/VS options. For example,

```
OPTIONS=(@@FLMUID/DEV(3800N8),CH(GT12,GB12),TW,CO,      C
B(7,7),M(DELAY,TRACE,ID))
```

## Return Codes

For the return codes 0 through 20 and return code 40, refer to the *DCF SCRIPT/VS* documentation for the explanation of each return code.

---

24

**Explanation:** SCLM did not allocate TEXTOUT.

**User response:** Contact the project manager.

**Project manager response:** Verify that an FLMALLOC macro has been coded for this translator with a ddname of TEXTOUT for the SCRIPT/VS output file. Reassemble the project definition. Verify that no errors occurred. Relink the project definition. For more information see "FLMALLOC Macro" on page 130.

---

28

**Explanation:** SCLM did not allocate TEXTIN.

**User response:** Contact the project manager.

**Project manager response:** Verify that an FLMALLOC macro has been coded for this translator with a ddname of TEXTIN for the SCRIPT/VS source file. Reassemble the project definition. Verify that no errors occurred. Relink the project definition. For more information see "FLMALLOC Macro" on page 130.

---

36

**Explanation:** The user ID was not specified in the input, or the parameter list has an incorrect format.

**User response:** Contact the project manager.

**Project manager response:** Verify that the options list is in the correct format for this translator. Reassemble the project definition. Verify that no errors occurred. Relink the project definition.

---

# FLMTPRE

## Purpose

FLMTPRE is the precompile processor called before a translator that processes input lists. FLMTPRE supports both non-Ada and Ada input lists. It initializes the ADA library file (DDNAME=ADALIB) with the names of the sublibraries required to perform the ADA compile by the next translator. It also initializes the input list file (DDNAME=ADAIN) with the names of the source members to be compiled. The translator that processes the data sets uses this list of data sets as input.

The input list data set allocated to the ADAIN ddname contains a list of members in compilation order to be built. Each member is listed in the data set using its fully-qualified name enclosed in single quotes. Here is a description of the ADAIN format and a sample input list.

Start Column	Length	Description
1	56	Fully-qualified project partitioned data set name, enclosed in single quotes.

```
'PROJECT1.RELEASE.SOURCE(SUB4)'  
'PROJECT1.INT.SOURCE(PROC2)'  
'PROJECT1.STAGE.SOURCE(SUB2)'  
'PROJECT1.USER.SOURCE(PROC1)'  
'PROJECT1.INT.SOURCE(MAIN)'
```

If a non-Ada language is used, direct the ddname ADALIB to NULLFILE:

```
FLMALLOC IOTYPE=W,DDNAME=ADALIB  
FLMCPYLB NULLFILE
```

and do not specify parameters SUBLIB1...SUBLIB8 on the FLMTRNSL macro for calling FLMTPRE.

The names of the sublibrary data sets will be placed in the Ada Library file by FLMTPRE. The names are listed in the following order:

```
sublibraries controlled by SCLM  
sublibraries not controlled by SCLM
```

You can pass up to 8 sublibraries NOT under SCLM control to FLMTPRE using the SUBLIB# parameter (as noted below). These sublibraries are usually system-level or runtime sublibraries.

The CU qualifier used to create the sublibrary names is the SUFFIX specified on the input parameter string. If the SUFFIX parameter is not specified, the cu\_qual on the FLMLANGL macro is used to generate sublibrary names.

## Parameters

The following keyword parameters are expected as input to FLMTPRE:

### DDMSG

This is an optional parameter to specify a ddname for messages. The default is FLMTMSG.

### FLM\_INFO

This parameter is used to access the list of members to be placed into the input list file. The name of each member is placed into the input list file, and then the input list file is allocated to the ddname ADAIN. This parameter is required and must be set to @@FLMINF.

### SUBLIB1...SUBLIB8

Sublibrary not under SCLM control to be added. These are optional parameters. You can specify up to 8 of these sublibraries. When not specified, these parameters default to DUMMY.

### SUFFIX

Suffix to use when generating sublibrary names (that is, the CU qualifier). This parameter is optional. If it is not specified, the CU qualifier on the FLMLANGL macro is used to generate the sublibrary name.

## Return Codes

---

0

**Explanation:** Indicates a successful completion.

**User response:** None.

**Project manager response:** None.

---

4

**Explanation:** The sublibrary name is longer than MVS allows. The sublibrary name is formed by concatenating the suffix to the project data set name specified for the group and type being processed.

**User response:** Contact the project manager.

**Project manager response:** The physical data set name concatenated with the suffix cannot exceed 44 characters. The data set name must be shortened in the project definition.

---

8

**Explanation:** The FLM\_INFO parameter does not specify a valid SCLM information record pointer.

**User response:** Contact the project manager.

**Project manager response:** The FLM\_INFO parameter is either missing or does not specify a valid pointer to the SCLM information record. This parameter should be specified as follows: FLM\_INFO=@@FLMINF. Correct this parameter on the translator definition in the project definition being used. Regenerate the project definition. Submit the job again.

---

# FLMTPST

## Purpose

This translator is the Input List compiler post-compile processor. The Input list feature supports translators that allow the user to specify a list of input data sets for each invocation. Because the translator performing the processing will be operating on a list of files, a list of return codes must be provided to SCLM to correctly manage the build. The FLMTPST translator passes return code information back to SCLM.

The FLMTPST translator takes as input the file allocated to the ADAOUT ddname. The data set pointed to by the ddname ADAOUT must contain lines beginning with \* or RC=XX, with the first nonblank character in column one. Each line within the data set that contains RC= in columns 1–3 is processed. Lines beginning with an asterisk(\*) are considered comments and are ignored. Each line containing RC= in columns 1 through 3 must follow this format:

```
RC=XX 'DATA SET NAME(MEMBER)'
```

The data set name is the same as that specified in the Input list generated by the FLMPRE translator. The lines in the ADAOUT file must match the order of the lines in the ADAIN file that was generated by FLMPRE. If the order does not match, an error is generated and the FLMTPST translator stops. Here is the format and a sample of the ADAOUT file.

Start Column	Length	Description
1	1	The character "*" followed by anything up to 255 characters - OR -
1	3	String 'RC='
4	2	Two-character integer return code
6	1	Blank space
7	56	Fully-qualified project partitioned data set name, enclosed in single quotes.

```
RC=00 'PROJECT1.RELEASE.SOURCE(SUB4)'  
RC=04 'PROJECT1.INT.SOURCE(PROC2)'  
RC=00 'PROJECT1.STAGE.SOURCE(SUB2)'  
RC=04 'PROJECT1.USER.SOURCE(PROC1)'  
RC=00 'PROJECT1.INT.SOURCE(MAIN)'
```

## Parameters

FLMTPST requires the following keyword parameter:

### FLM\_INFO

Pointer to the SCLM information record. This parameter must be set to @@FLMINF.

## Return Codes

---

0

**Explanation:** Indicates a successful completion.

**User response:** None.

**Project manager response:** None.

---

4

**Explanation:** At least one, but not all, of the members processed were built successfully.

**User response:** Check the build messages produced to determine which members failed to build successfully.

**Project manager response:** None.

---

8

**Explanation:** The FLM\_INFO parameter does not specify a valid SCLM information record pointer.

**User response:** Contact the project manager.

**Project manager response:** The FLM\_INFO parameter

is either missing or does not specify a valid pointer to the SCLM information record. This parameter should be specified as follows: FLM\_INFO=@@FLMINF. Correct this parameter on the translator definition in the project definition being used. Regenerate the project definition. Submit the job again.

---

12

**Explanation:** The compiler output file allocated to ddname ADAOUT contains some unexpected information.

**User response:** Contact the project manager.

**Project manager response:** Verify that the file allocated to the ADAOUT ddname is the output file generated by the input list compiler. To do this, look at the contents of the file allocated to ADAOUT. Specify the PRINT=Y parameter on the FLMALLOC macro for the ADAOUT file allocation for the input list compiler translator definition. Regenerate the project definition you are using, and submit the job again. The job output will contain the contents of the file allocated to ADAOUT.

For IBM Ada/370, the compiler documentation lists the contents of the output file generated by the input list compile. Verify that the contents of the ADAOUT file printed in the job output conform to the documented values in the compiler documentation. If they do not match, report this problem to your IBM service representative.

When the information in the ADAOUT file is not as expected, contact your IBM service representative for assistance.

---

16

**Explanation:** Indicates that a line in the output file allocated to ddname ADAOUT has an unexpected format.

**User response:** Contact the project manager.

**Project manager response:** See the project manager response described for return code 12 of this translator.

---

20

**Explanation:** Indicates the file allocated to the ADAOUT ddname is empty.

**User response:** Verify that the input list compiler was successfully invoked and produced an output file. If necessary, contact the project manager.

**Project manager response:** Verify that the ADAOUT ddname was allocated for the invocation of the input list compiler. If necessary, add an FLMALLOC macro for the ADAOUT ddname to the input list compiler translator definition. Regenerate the project definition and submit the job again. If this problem recurs, report

this problem to your IBM service representative for assistance.

---

24

**Explanation:** Indicates that the ADAOUT ddname is not allocated.

**User response:** Contact the project manager.

**Project manager response:** This could indicate improper usage of the FLMTTPST translator. The FLMTTPST translator should be invoked only after the input list compiler has been invoked. Verify that the language definition being used is for input list processing and that the FLMTTPST translator is being invoked after the input list translator.

Also, verify that there is an FLMALLOC macro specified for the ADAOUT ddname for a previous build translator in the current Ada language definition. If either of these problems are present, change the language definition, the project definition, or both to correct the problem. Regenerate the project definition and submit the job again.

---

## FLMTXFER Workstation Transfer - Build Translator

### Purpose

The FLMTXFER translator uses the FILEXFER service to send and receive files from a workstation. When sending files to the workstation, the source data sets on the host (MVS) system can be SCLM members, sequential data sets, or members of partitioned data sets. When receiving files from the workstation, the target data sets on the host (MVS) system can be sequential data sets or members of partitioned data sets.

When transferring SCLM members to the workstation, SCLM keeps track of which members have been sent to the workstation during a build and only sends each member to the workstation one time during the build.

For an example of the usage of this translator, see the FLMLTWST translator.

### Parameters

All parameters are keyword parameters and can be specified in any order. Parameters must be separated by commas. Extraneous parameters are ignored without any messages being produced.

#### COMMAND=PUT|GET

This parameter is required and must be specified in the options list. The valid values are:

**PUT** Use this value to send files to the workstation.

**GET** Use this value to retrieve files from the workstation.

This parameter will be truncated to 3 characters.

#### BLDINFO=@@FLMBIO

This parameter is required and must be specified in the options list with the value from @@FLMBIO.

#### SCLMINFO=@@FLMINF

This parameter is required and must be specified in the options list with the value from @@FLMINF.

#### MESSAGEDD=dd\_name

This parameter is optional. If not specified, it defaults to MESSAGE. This is the ddname where messages will be written. This parameter will be truncated to 8 characters.

#### FILESDD=dd\_name

This parameter is optional. If not specified, it defaults to FILES. This is the ddname containing the list of files to transfer. This parameter will be truncated to 8 characters. The data set allocated to this ddname must list one transfer specification per line. Each part of the transfer specification must be separated from other parts by one or more spaces. The following information must be provided with each transfer specification:

##### transfer format

This is a single character value that specifies the format of the file transfer.

**A** Translate to ASCII

**B** No translation

**host data** This specifies what data set or member is the source or target

of the transfer. If COMMAND=PUT is specified, this is the source of the file transfer. If COMMAND=GET is specified, this is the target of the file transfer. The format varies depending on the data being transferred.

For COMMAND=PUT, use IOTYPE=P to take advantage of the build caching function.

<b>Data</b>	<b>Format</b>
-------------	---------------

<b>SCLM member</b>	
--------------------	--

member.type

This format is only valid for COMMAND=PUT. The member name must be separated from the type name by a period. The member must be in the scope of the build. These members are tracked during the build and only sent to the workstation once even if FLMTXFER is called multiple times with the same member. FLMTXFER finds the member in the SCLM hierarchy and generates a fully-qualified data set name with the member name for the file transfer.

<b>Data set</b>	'data.set.name'
-----------------	-----------------

This format transfers a fully-qualified data set name. The data set must be sequential or specify the member name. The data set name must be surrounded by quotes. SCLM does not track the data sets sent to the workstation. If FLMTXFER is called multiple times to transfer the same data set, the data set is transferred each time.

<b>ddname</b>	DDNAME:member
---------------	---------------

The member name is optional, but the colon (:) must be specified. The data set allocated to the ddname must be cataloged (CATLG=Y on the FLMALLOC). FLMTXFER gets the data set name allocated to the ddname and specifies it in the file transfer command.

<b>workstation file name</b>	
------------------------------	--

The fully-qualified workstation file name including the drive and path, if they apply.

## **Environment**

The FLMTXFER translator must have access to ISPF services. It must be called from an FLMTRNSL with CALLMETH=ISPLNK.

## **Return Codes**

In addition to the return codes listed here, messages can be written to the ddname specified by the MESSAGEDD parameter.

---

0

**Explanation:** The transfer was successful.

**User response:** None.

**Project manager response:** None.

---

8

**Explanation:** An error occurred. See the ddname specified by the MESSAGEDD parameter for more information.

**User response:** Refer to the generated messages. See the *ISPF Messages and Codes* manual for an explanation of the messages.

**Project manager response:** None.

---

12

**Explanation:** The ddname specified by the MESSAGEDD parameter is not allocated.

**User response:** Contact the project manager.

**Project manager response:** Ensure that the ddname for messages is allocated by an FLMALLOC in the language definition or by the translator that calls FLMTXFER.

---

16

**Explanation:** The value for SCLMINFO is not valid.

**User response:** Contact the project manager.

**Project manager response:** Ensure that the SCLMINFO parameter is specified in the options list and that the parameter has a value of @@FLMINF.

---

### FILES DD Example

The following examples show the content of the FILES ddname. The first shows how to send compiler inputs to a workstation and the second how to retrieve the outputs.

Given COMMAND=PUT, in the first example the following transfers take place:

1. The data set allocated to the RESPONSE ddname is sent to c:\temp\response.file in ASCII format.
2. The member PMLINES in type C is found in the SCLM hierarchy and sent to c:\temp\PMLINES.c in ASCII format.
3. The member PMLINES in type H is found in the SCLM hierarchy and sent to c:\temp\PMLINES.h in ASCII format.
4. The data set 'PROJ1.C.LIB' is sent to c:\temp\proj1.lib in BINARY format.

```
A RESPONSE: c:\temp\response.fil
A PMLINES.C c:\temp\PMLINES.c
A PMLINES.H c:\temp\PMLINES.h
B 'PROJ1.C.LIB' c:\temp\proj1.lib
```

Given COMMAND=GET, in the second example the following transfers take place:

1. The file c:\temp\PMLINES.obj is sent to the member PMLINES in the data set allocated to the OBJ ddname in BINARY format.
2. The file c:\temp\PMLINES.lst is sent to the member PMLINES in the data set allocated to the LIST ddname in ASCII format.
3. The file c:\temp\temp.msg is sent to the data set 'SCLMUSR.C.MSGS' in ASCII format.

```
B OBJ:PMLINES c:\temp\PMLINES.obj
A LIST:PMLINES c:\temp\PMLINES.lst
A 'SCLMUSR.C.MSGS' c:\temp\temp.msg
```

---

## SCLM Parser Restrictions

The SCLM parsers gather statistics on various language constructs. This section describes the constructs that the SCLM parsers cannot identify. Because a user-defined parser can be used to replace an SCLM parser, the restrictions of the SCLM-supplied parsers can be overcome, if necessary.

Unsupported constructs do not necessarily prevent members from being used in SCLM. Invalid constructs, however, prevent statistics from being gathered accurately and can result in SCLM finding too many or too few include references. Extra or missing includes can result in dependency processing errors being detected by the build and promote processors.

SCLM does not support three general types of language constructs. Each of these constructs involves include and compool references. The constructs discussed in this chapter are:

- Cross-type references
- Non-explicit references
- Separation of references.

### Non-Explicit References

SCLM-supplied parsers do not support include references that are not explicitly stated on a single line of code.

The following list shows three kinds of non-explicit reference constructs.

- **Conditional References**

*Conditional references* are include reference constructs that depend on information outside the scope of a single line. For the assembler language parser, for instance, all macros are considered include references whether or not they are defined within that assembly source member. All include references must exist as SCLM members, or they must exist in data set FLMSYSLB references.

- **Dynamic References**

*Dynamic references* are references that involve a variable. SCLM does not support macro names passed as parameters in assembler language for include references. The following source statements for SCRIPT/VS depict a simple case of a dynamic imbed reference that SCLM does not support:

```
.set count = 1
.im member
```

- **Variable Delimiters**

The delimiters you use to identify information must have fixed values. For example, SCLM does not support the following format of the **.DM** script keyword:

```
.DM name /.im seg1/.im seg2/.im seg3/
```

where / can be any character. This character delimits statements in the macro. SCLM does not find imbed statements entered in the **.DM** macro when the macro appears in this way.

SCLM also does not support the following format of the **.DM** macro:

```
.DM name ON
.im seg1
.im seg2
.im seg3
.DM OFF
```

## Separation of References

Generally, you *must* separate include reference verbs of a language from referenced member names with blanks only, and they must appear on the same line.

However, there are two exceptions:

- For PL/I includes and JOVIAL members, when coding !COPY or !COMPOOL statements, you can insert comments between !COPY and the include member name. (Note that only JOVIAL uses !COMPOOL.)
- The following parsers support include references on separate lines:
  - FLMLPCBL COBOL parser
  - FLMLRCBL REXX COBOL parser
  - FLMLRASM REXX Assembler parser.

SCLM does not support the following Pascal source statement because a comment separates the referenced member name.

```
%INCLUDE (* comment *) MEMNAME;
```

The include reference verb and the reference name must reside on the same line.

SCLM does not support the following Pascal statement:

```
%INCLUDE  
INCLMEM ;
```



---

## Chapter 6. SCLM Variables and Metavariables

This chapter lists the SCLM variables and metavariables you can use in various stages of SCLM processing.

---

### 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 10 on page 280 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 9 on page 279 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
- @@FLMGRB
- @@FLMGRP
- @@FLMMBR
- @@FLMPRJ
- @@FLMSRF
- @@FLMTYP
- @@FLMUID

Note that @@FLMDSN and @@FLMGRP reflect the group where the member being built resides. Use @@FLMGRB for the name of the group where the build is being performed.

- On the Database Contents Utility line format parameter (DBUTIL)
- On the DSNNAME 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 7 lists the SCLM variables in alphabetic order by description and indicates for which translator types they can be used. Table 8 on page 275 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 7 lists the SCLM variables in alphabetic order by their short description.

*Table 7. 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						✓

Table 7. SCLM Variable Descriptions, Names, and Their SCLM Functions (continued)

SCLM Short Description	Variable	Build	Copy	Parse	Purge	Verify	Utils
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						✓
Change Code Date with 4-character year	@@FLM\$C4						✓
Change Code Time	@@FLM\$CT						✓
Change Date	@@FLMCDDT		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					

Table 7. SCLM Variable Descriptions, Names, and Their SCLM Functions (continued)

SCLM Short Description	Variable	Build	Copy	Parse	Purge	Verify	Utils
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					
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						✓

Table 7. SCLM Variable Descriptions, Names, and Their SCLM Functions (continued)

SCLM Short Description	Variable	Build	Copy	Parse	Purge	Verify	Utils
Output Member Name	@@FLMONM						
OUT0 Member Name	@@FLMOU0	P					
OUT1 Member Name	@@FLMOU1	P					
OUT2 Member Name	@@FLMOU2	P					
OUT3 Member Name	@@FLMOU3	P					
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	

Table 7. SCLM Variable Descriptions, Names, and Their SCLM Functions (continued)

SCLM Short Description	Variable	Build	Copy	Parse	Purge	Verify	Utils
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 8 lists the SCLM variables in alphabetic order by variable name.

Table 8. 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						✓
@@FLMCDDT	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						✓

Table 8. SCLM Variables and Their SCLM Functions (continued)

Variable	SCLM Short Description	Build	Copy	Parse	Purge	Verify	Utils
@@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					
@@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					
@@FLMDS D	Destination Group Data Set Name		P		P	P	
@@FLMDS F	Group Found Data Set Name		P		P	P	
@@FLMDS N	Accounting Group Data Set Name	P	P	P	P	P	↖
@@FLMDS T	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	

Table 8. SCLM Variables and Their SCLM Functions (continued)

Variable	SCLM Short Description	Build	Copy	Parse	Purge	Verify	Utils
@@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		
@@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	✓
@@FLMNMN	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					

Table 8. SCLM Variables and Their SCLM Functions (continued)

Variable	SCLM Short Description	Build	Copy	Parse	Purge	Verify	Utils
@@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						✓
@@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 9 lists the SCLM metavariables in alphabetic order by description. Metavariables are only used with the DBUTIL service.

*Table 9. 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 10 on page 280 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 10 on page 280.

Table 10. 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
@@FLM#AL	@@FLM\$XT @@FLM\$XN @@FLM\$UD

---

## Description of Group Variables

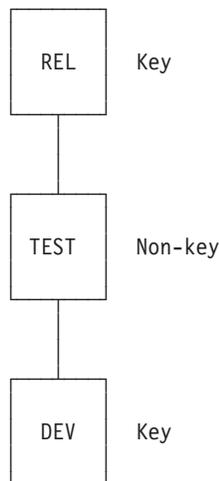
This section further explains the use of group variables. Table 11 lists each group variable and associated group data set name variable. This shows the relationship between SCLM groups and the data sets defined in the project definition for each group.

Table 12 on page 282 is an example that lists the values of each group variable during the phases of a promote. After Table 11 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 11. 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	@@FLMDSO	Destination Group and Destination Group Data Set Name

The following hierarchy will be used in the description:



*Figure 7. 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 12. SCLM Group Variable Description

<b>Translator</b>	<b>Accounting Group</b>	<b>Group Found</b>	<b>Target Group</b>	<b>Destination Group</b>
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 primarily 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	Language Environment
C++	MVS
DFSMSdfp	MVS/ESA
DFSMSdss	OS/2
DFSMSHsm	OS/390
DFSMSrmm	OS/390 Security Server
DFSMS/MVS	RACF
DFSORT	Resource Access Control Facility
ESCON	SOMobjects
FFST	System View
GDDM	VisualLift
IBM	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 (overlaid 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 copies 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 use 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

@@FLM#AF 279  
@@FLM#AL 279  
@@FLM\$C4 272, 278  
@@FLM\$CC 272, 278  
@@FLM\$CD 272, 278  
@@FLM\$CT 272, 278  
@@FLM\$IN 273, 278  
@@FLM\$IS 273, 278  
@@FLM\$MP 271, 278  
@@FLM\$UD 275, 278  
@@FLMACC 271, 275  
@@FLMACD 271, 275  
@@FLMACK 271, 275  
@@FLMALT 271, 275  
@@FLMASG 271, 275  
@@FLMATP 271, 275  
@@FLMBD4 274, 275  
@@FLMBDT 274, 275  
@@FLMBIO 271, 275  
@@FLMBLL 271, 275  
@@FLMBMD 272, 275  
@@FLMBTM 274, 275  
@@FLMCD4 272, 275  
@@FLMCDT 272, 275  
@@FLMCLV 272, 275  
@@FLMCMCL 272, 275  
@@FLMCMCML 272, 275  
@@FLMCMCNS 272, 275  
@@FLMCMCRF 272, 275  
@@FLMCTM 272, 275  
@@FLMCUN 275  
@@FLMCUS 272, 275  
@@FLMDBQ 272, 276  
@@FLMDDDN 273, 276  
@@FLMDDO 274, 276  
@@FLMDO0 272, 276  
@@FLMDO1 272, 276  
@@FLMDO2 272, 276  
@@FLMDO3 272, 276  
@@FLMDO4 272, 276  
@@FLMDO5 272, 276  
@@FLMDO6 272, 276  
@@FLMDO7 273, 276  
@@FLMDO8 273, 276  
@@FLMDO9 273, 276  
@@FLMDSD 273, 276  
@@FLMDSF 273, 276  
@@FLMDSN 271, 276  
@@FLMDST 274, 276  
@@FLMECR 273, 276  
@@FLMETP 273, 276  
@@FLMFDT 273, 276  
@@FLMFNM 272, 276  
@@FLMFTM 273, 276  
@@FLMGRB 271, 276  
@@FLMGRD 273, 276  
@@FLMGRF 273, 276  
@@FLMGRP 271, 277  
@@FLMID4 272, 277  
@@FLMIDT 272, 277

@@FLMINC 273, 277  
@@FLMINF 274, 277  
@@FLMITM 272  
@@FLMLAN 273, 277  
@@FLMLIS 273, 277  
@@FLMLST 272, 277  
@@FLMLVS 273, 277  
@@FLMMBR 271, 277  
@@FLMMD4 271, 277  
@@FLMMDT 271, 277  
@@FLMMNM 271, 277  
@@FLMMSC 272, 277  
@@FLMMTM 272, 277  
@@FLMMVR 273, 277  
@@FLMNCC 273, 277  
@@FLMNCL 273, 277  
@@FLMNCS 273, 277  
@@FLMNIN 273, 277  
@@FLMNUE 273, 277  
@@FLMONM 274, 277  
@@FLMOU0 274, 277  
@@FLMOU1 274, 277  
@@FLMOU2 274, 277  
@@FLMOU3 274, 277  
@@FLMOU4 274, 278  
@@FLMOU5 274, 278  
@@FLMOU6 274, 278  
@@FLMOU7 274, 278  
@@FLMOU8 274, 278  
@@FLMOU9 274, 278  
@@FLMPD4 274, 278  
@@FLMPDT 274, 278  
@@FLMPRJ 274, 278  
@@FLMPRL 274, 278  
@@FLMPTM 274, 278  
@@FLMPUS 274, 278  
@@FLMSIZ 271, 278  
@@FLMSRF 273, 278  
@@FLMSTA 271, 278  
@@FLMSTP 274, 278  
@@FLMTLL 275, 278  
@@FLMTLS 275, 278  
@@FLMTOG 274, 278  
@@FLMTVS 275, 278  
@@FLMTYP 271, 278  
@@FLMUID 274, 278  
@@FLMVER 274, 278  
\$acct\_info 14  
\$list\_info 15  
    accounting records 15  
\$msg\_array 13  
\$stats\_info 15

## A

access key  
    incorrect 43  
    locking a member 64  
    purpose for 65  
    resetting 92  
    variable 275

access key (*continued*)  
    verification 65  
accounting group  
    definition of 39  
    variable 277  
accounting information, field format 14  
accounting member variable 271, 277  
accounting record type variable 271, 275  
accounting records  
    DBACCT service 36  
    DELETE service 42  
    DELGROUP service 44  
    metavariables 279  
    retrieve 36  
    variables 270  
accounting status variable 271, 278  
accounting type variable 271, 278  
ACCTINFO Service 25  
allocating SCLM data sets pointer  
    parameters 12  
arrays  
    accounting information 14  
    list information 15  
    message 13  
    statistical information 15  
assignment statement variable 271, 275  
AUTHCODE Service 28  
authorization code  
    variable 271, 275  
    verification  
        LOCK service 64, 66  
        MIGRATE service 69  
        SAVE service 84  
authorization code change variable 271, 275

## B

blank lines variable 271, 275  
buffer size  
    definition of 181  
    variable 271, 278  
build function  
    build map variables 271, 277  
    parameters 33  
build map information variable 271  
build map variable 271, 278  
BUILD service 32  
build user exit routine specification 34

## C

call format  
    C 11  
    COBOL 11  
    FORTRAN 10  
    Pascal 10  
    PL/I 11  
calling function name variable 272, 276

- change code
  - array record 15
  - variables 272, 278
- character parameters 12
- CLIST
  - command procedure 8
  - variable 6
- code, authorization
  - variable 271, 275
  - verification
    - LOCK service 64, 66
    - MIGRATE service 69
    - SAVE service 84
- code, change
  - array record 15
  - variables 272, 278
- code, return
  - BUILD service 35
  - DBACCT service 37
  - DBUTIL service 41
  - DELETE service 43
  - DELGROUP service 47
  - DSALLOC service 50
  - EDIT service 54
  - END service 55
  - EXPORT service 57
  - FREE service 58
  - general categories 20
  - GOODRC 192
  - IMPORT service 61
  - INIT service 63
  - LOCK service 67
  - MIGRATE service 70
  - PARSE service 75
  - PROMOTE service 79
  - RPTARCH service 82
  - SAVE service 86
  - START service 88
  - STORE service 91
  - UNLOCK service 94
  - VERDEL service 96
  - VERINFO service 99
  - VERRECOV service 102
- command
  - data set conventions 7
  - FLMCMD 6
  - interactive processing 8
  - invocation format 6
  - QUIT 8
  - service invocation 5, 6
- command processing, interactive 8
- comment lines variable 272, 275
- comment statements variable 272, 275
- considerations, performance 7
- control options
  - DASDUNIT 160
- control statements variable 272, 275
- cross reference variables 271, 275
- CU list variable 272, 277

## D

- DASDUNIT control option 160
- data set naming conventions
  - ALTC parameter 175
  - FLMGROUP 174
  - using FLMALTC 148

- data set protection 12
- database contents utility, selection criteria
  - pattern examples 13
- database qualifier
  - format 14
  - variable 272, 276
- DBACCT service 36
- DBUTIL service 38
- DDNAME parameters 12
- ddname substitution list
  - use of 131
  - variable 273, 276
- default type variable 273, 278
- defining software component 147
- DELETE service 42
- DELGROUP service 44
- dependencies pointer variable 273, 277
- DSALLOC service 48
- dynamic includes variable 273, 277

## E

- EDIT service 51
- END service 54
- EXPACCT control option 158
- EXPORT service 55
- extended CREF type variable 273
- Extended Type 196

## F

- field name metavariables 279
- field name variables 271, 275
- FILE format 6
- flexible data set names
  - ALTC parameter 175
  - FLMGROUP 174
  - using FLMALTC 148
- FLMABEG macro 129
- FLMAEND macro 130
- FLMAGRP macro 130
- FLMALLOC macro, defining language
  - definitions 130
- FLMALTC macro 148
- FLMATVER macro 152
- FLMCMD command
  - CLIST command procedure 8
  - command line format 7
  - data set example 8
  - FILE format 6
  - interactive processing 8
  - invocation format 6
  - parameters 6
- FLMCNTRL macro 155
- FLMCPYLB macro 173
- FLMCPYLB required statements 1
- FLMCSPDB translator 199
- FLMDTLC translator 202
- FLMGROUP macro 174
- FLMINCLS macro 176
- FLMLANGL macro 180
- FLMLNK subroutine interface
  - call format 9
  - character parameters 12
  - parameter conventions 9
  - pointer parameters 13

- FLMLPCBL parser 203
- FLMLPFRT parser 206
- FLMLPGEN parser
  - used as a CLIST or REXX parser 211
  - used as a generic parser 211
  - used as a PL/I parser 210
  - used as a TEXT parser 211
  - used as an Assembler parser 210
- FLMLRASM REXX Assembler
  - parser 214
- FLMLRBLD macro 182
- FLMLRC2 C, C++, and Resource file
  - parser for workstation source 225
- FLMLRC37 REXX C370 parser 228
- FLMLRCBL REXX COBOL parser 218
- FLMLRCIS MVS C/C++ parser with
  - include set support 222
- FLMLRDTL translator 232
- FLMLRIPF Script and OS/2 IPF Source
  - Sarser 233
- FLMLSS parser 236
- FLMLTWST translator 240
- FLMSYSLB macro 183
- FLMTBMAP translator 256
- FLMTCOND macro 185
- FLMTMSI translator 258
- FLMTOPTS macro 189
- FLMTPRE translator 259
- FLMTPST translator 261
- FLMTRNSL 190
- FLMTXFER translator 263
- FLMTYPE macro 196
- FREE service 58
- function invocation variables
  - build group 276
  - date 273, 276
  - time 273, 276

## G

- GOODRC 192
- group
  - development library 64
  - variables description 281
  - verification 64
- group found variable 273, 276
- group\_list 183

## I

- IMPORT service 59
- include reference variable 273, 278
- include-sets for includes variable 273
- INIT service 62
- initialize parameter variables 10
- interactive command processing 8
- intermediate variables 275, 279
- ISPF variables 17

## K

- keywords
  - assembler call statement 10
  - FLMALLOC macro 130
  - FLMLANGL macro 180
  - FLMLRBLD macro 182

keywords (*continued*)  
FLMTRNSL macro 190

## L

language  
  constructs 266  
  variable 273, 277  
language restrictions  
  on non-explicit references 266  
  on separation of references 267  
list information array 15  
listing data set, output specification 34  
LOCK service  
  invocation of 63

## M

macro  
  FLMABEG 129  
  FLMAEND 130  
  FLMAGRP 130  
  FLMALLOC 132  
  FLMALTC 148  
  FLMATVER 152  
  FLMCNTRL 157  
  FLMCPYLB 173  
  FLMGROUP 174  
  FLMINCLS 176  
  FLMLANGL 180  
  FLMLRBLD 182  
  FLMSYSLB 183  
  FLMTCOND 185  
  FLMTOPTS 189  
  FLMTRNSL 190  
  FLMTYPE 196  
  instructions 127  
messages  
  array 13  
  DBUTIL service 40  
  output specification 34  
  RPTARCH service 82  
metavariables  
  cross-reference 279  
  field names 279  
  functions 279  
  list of 279  
  report 271, 279  
  uses for 279  
MIGRATE service 68  
migration considerations  
  FLMCPYLB statements required 1  
  SCLM 1

## N

NEXTGRP Service 71  
notation conventions 5, 127

## O

OPTFLAG 192  
options, control  
  DASDUNIT 160  
OPTOVER control option 192

output member name variable 274, 277

## P

packed data set, saving 68, 83  
parameters  
  ACCTINFO service 25  
  AUTHCODE service 28  
  BUILD service 33  
  character 12  
  DBACCT service 36  
  DBUTIL service 38  
  DDNAME 12  
  DELETE service 43  
  DELGROUP service 45  
  DSALLOC service 49  
  EDIT service 52  
  END service 55  
  EXPORT service 56  
  FLMABEG macro 129  
  FLMAEND macro 130  
  FLMAGRP macro 130  
  FLMALLOC macro 133  
  FLMALTC macro 148  
  FLMATVER macro 152  
  FLMCNTRL macro 157  
  FLMCPYLB macro 173  
  FLMGROUP macro 174  
  FLMINCLS macro 176  
  FLMLANGL macro 180  
  FLMLRBLD macro 182  
  FLMSYSLB macro 183  
  FLMTCOND macro 185  
  FLMTOPTS macro 189  
  FLMTRNSL macro 190  
  FLMTYPE macro 196  
  FREE service 58  
  IMPORT service 60  
  INIT service 62  
  LOCK service 66  
  MIGRATE service 69  
  NEXTGRP service 71  
  PARSE service 74  
  pointer 13  
  PROMOTE service 77  
  RPTARCH service 81  
  SAVE service 84  
  START service 88  
  STORE service 90  
  UNLOCK service 93  
  VERDEL service 95  
  VERINFO service 97  
  VERRECOV service 100  
  PARSE service  
    invocation of 73  
  parser restrictions 266  
  Pascal  
    integer variable 21  
    program sample 105  
  patterns for selection criteria 12  
  performance considerations 7  
  PL/I program sample 121  
  pointer parameters  
    \$acct\_info 14  
    \$list\_info 15  
    \$msg\_array 13  
    \$stats\_info 15

precedence verification 64  
predecessor, definition of 64  
processing interactive command 8  
program sample, Pascal 105  
program sample, PL/I 121  
PROMOTE service 76

## R

report  
  cutoff 81  
  output specification 34, 70  
return codes  
  BUILD service 35  
  DBACCT service 37  
  DBUTIL service 41  
  DELETE service 43  
  DELGROUP service 47  
  DSALLOC service 50  
  EDIT service 54  
  END service 55  
  EXPORT service 57  
  FREE service 58  
  general categories 20  
  GOODRC 192  
  IMPORT service 61  
  INIT service 63  
  LOCK service 67  
  MIGRATE service 70  
  PARSE service 75  
  PROMOTE service 79  
  RPTARCH service 82  
  SAVE service 86  
  START service 88  
  STORE service 91  
  UNLOCK service 94  
  VERDEL service 96  
  VERINFO service 99  
  VERRECOV service 102  
RPTARCH service 80

## S

sample program  
  Pascal 105  
  PL/I 121  
SAVE service 83  
SCLM internal data pointer  
  definition of 14  
  variable 274, 277  
SCLM metavariables  
  account report fixed  
    (@@FLM#AF) 279  
  account report long  
    (@@FLM#AL) 279  
SCLM migration considerations 1  
SCLM services  
  data set protection 12  
  general discussion 5  
  performance considerations 7  
SCLM variables  
  access key (@@FLMACK) 271, 275  
  accounting group (@@FLMGRP) 271,  
  277  
  accounting group data set name  
    (@@FLMDSN) 271, 276

SCLM variables (*continued*)

accounting member  
 (@@FLMMBR) 271, 277  
 accounting record type  
 (@@FLMATP) 271, 275  
 accounting status (@@FLMSTA) 271,  
 278  
 accounting type (@@FLMTYP) 271,  
 278  
 alternate project definition  
 (@@FLMALT) 271, 275  
 assignment statements  
 (@@FLMASG) 271, 275  
 authorization code  
 (@@FLMACD) 271, 275  
 authorization code change  
 (@@FLMACC) 271, 275  
 blank lines (@@FLMBLL) 271, 275  
 buffer size in bytes (@@FLMSIZ) 271,  
 278  
 build group (@@FLMGRB) 271, 276  
 build map (@@FLM\$MP) 271, 278  
 build map date (@@FLMMD4) 271,  
 277  
 build map date (@@FLMMDT) 271,  
 277  
 build map information  
 (@@FLMBIO) 271, 275  
 build map name  
 (@@FLMMNM) 271, 277  
 build map time (@@FLMMTM) 272,  
 277  
 build map type (@@FLMMSC) 272,  
 277  
 build mode (@@FLMBMD) 272, 275  
 calling function name  
 (@@FLMFNM) 272, 276  
 change code (@@FLM\$CC) 272, 278  
 change code data (@@FLM\$C4) 272  
 change code data (@@FLM\$CD) 272  
 change code date (@@FLM\$C4) 278  
 change code date (@@FLM\$CD) 278  
 change code time (@@FLM\$CT) 272,  
 278  
 change date (@@FLMCD4) 272, 275  
 change date (@@FLMCDT) 272, 275  
 change group (@@FLMCLV) 272, 275  
 change time (@@FLMCTM) 272, 275  
 change user ID (@@FLMCUS) 272,  
 275  
 comment lines (@@FLMCML) 272,  
 275  
 comment statements  
 (@@FLMCMS) 272, 275  
 control statements (@@FLMCNS) 272  
 control statments (@@FLMCNS) 275  
 creation date (@@FLMID4) 272, 277  
 creation date (@@FLMIDT) 272, 277  
 creation time (@@FLMITM) 272  
 CREF type (@@FLMCREF) 272, 275  
 CU list (@@FLMLST) 272, 277  
 data set name for OUT0  
 (@@FLMDO0) 272, 276  
 data set name for OUT1  
 (@@FLMDO1) 272, 276  
 data set name for OUT2  
 (@@FLMDO2) 272, 276

SCLM variables (*continued*)

data set name for OUT3  
 (@@FLMDO3) 272, 276  
 data set name for OUT4  
 (@@FLMDO4) 272, 276  
 data set name for OUT5  
 (@@FLMDO5) 272, 276  
 data set name for OUT6  
 (@@FLMDO6) 272, 276  
 data set name for OUT7  
 (@@FLMDO7) 273, 276  
 data set name for OUT8  
 (@@FLMDO8) 273, 276  
 data set name for OUT9  
 (@@FLMDO9) 273, 276  
 database qualifier (@@FLMDBQ) 272,  
 276  
 DDNAME substitution list  
 (@@FLMDDN) 273, 276  
 default type (@@FLMSRF) 273, 278  
 dependencies pointer  
 (@@FLMLIS) 273, 277  
 destination group  
 (@@FLMGRD) 273, 276  
 destination group data set name  
 (@@FLMDSD) 273, 276  
 dynamic includes pointer  
 (@@FLMINC) 273, 277  
 extended CREF type  
 (@@FLMECR) 273, 276  
 extended type of source member  
 (@@FLMETP) 273, 276  
 function invocation date  
 (@@FLMFDT) 273, 276  
 function invocation time  
 (@@FLMFTM) 273, 276  
 group found (@@FLMGRF) 273, 276  
 group found data set name  
 (@@FLMDSF) 273, 276  
 include (@@FLM\$IN) 273, 278  
 include sets for includes  
 (@@FLM\$IS) 273, 278  
 language (@@FLM) 277  
 language (@@FLMLAN) 273  
 language version (@@FLMLVS) 273,  
 277  
 member version (@@FLMMVR) 273,  
 277  
 number of change codes  
 (@@FLMNCC) 273, 277  
 number of includes  
 (@@FLMNIN) 273, 277  
 number of noncomment lines  
 (@@FLMNCL) 273, 277  
 number of noncomment statements  
 (@@FLMNCS) 273, 277  
 number of user entries  
 (@@FLMNUE) 273, 277  
 OUT0 member name  
 (@@FLMOU0) 274, 277  
 OUT1 member name  
 (@@FLMOU1) 274, 277  
 OUT2 member name  
 (@@FLMOU2) 274, 277  
 OUT3 member name  
 (@@FLMOU3) 274, 277

SCLM variables (*continued*)

OUT4 member name  
 (@@FLMOU4) 274, 278  
 OUT5 member name  
 (@@FLMOU5) 274, 278  
 OUT6 member name  
 (@@FLMOU6) 274, 278  
 OUT7 member name  
 (@@FLMOU7) 274, 278  
 OUT8 member name  
 (@@FLMOU8) 274, 278  
 OUT9 member name  
 (@@FLMOU9) 274, 278  
 output member name  
 (@@FLMONM) 274, 277  
 predecessor date (@@FLMBD4) 274,  
 275  
 predecessor date (@@FLMBDT) 274,  
 275  
 predecessor time (@@FLMBTM) 274,  
 275  
 project (@@FLMPRJ) 274, 278  
 prolog lines (@@FLMPRL) 274, 278  
 promote date (@@FLMPD4) 274, 278  
 promote date (@@FLMPDT) 274, 278  
 promote time (@@FLMPTM) 274, 278  
 promote user ID (@@FLMPUS) 274,  
 278  
 SCLM internal data pointer  
 (@@FLMINF) 274, 277  
 SCLM version (@@FLMVER) 274,  
 278  
 static pointer (@@FLMSTP) 274, 278  
 sysprint DDNAME  
 (@@FLMDDO) 274, 276  
 system user ID (@@FLMUID) 274,  
 278  
 target group (@@FLMTOG) 274, 278  
 target group data set name  
 (@@FLMDST) 274, 276  
 top CU name (@@FLMCUN) 275  
 total lines (@@FLMTLL) 275, 278  
 total statements (@@FLMTLS) 275,  
 278  
 translator version (@@FLMTVS) 275,  
 278  
 user data entry (@@FLM\$UD) 275,  
 278  
 selection criteria 12  
 selection parameters 12  
 service  
 ACCTINFO 25  
 AUTHCODE 28  
 BUILD 32  
 character parameters 12  
 DBACCT 36  
 DBUTIL 38  
 DELETE 42  
 DELGROUP 44  
 DSALLOC 48  
 EDIT 51  
 END 54  
 EXPORT 55  
 FLMCMD interface 6  
 FREE 58  
 IMPORT 59  
 INIT 62

- service (*continued*)
  - interactive command processing 8
  - invocation from programs 5
  - LOCK 63
  - MIGRATE 68
  - NEXTGRP 71
  - notation conventions 5
  - PARSE 73
  - pointer parameters 13
  - PROMOTE 76
  - return code categories 20
  - RPTARCH 80
  - SAVE 83
  - START 87
  - STORE 89
  - UNLOCK 92
  - VERDEL 95
  - VERINFO 97
  - VERRECOV 100
- SREF statement, using 182
- START service 87
- static pointer
  - definition of 14
  - using 13
  - variable 274, 278
- statistical information
  - array 15
  - record field format 15
- STORE service
  - invoking 89
- sysprint ddname variable 274

## T

- title, on tailored report 40
- top CU name
  - variable 275
- translators
  - FLMCSPDB 199
  - FLMDTLC 202
  - FLMLPCBL 203
  - FLMLPFRT 206
  - FLMLPGEN 209
  - FLMLRASM 214
  - FLMLRC2 225
  - FLMLRC37 228
  - FLMLRCBL 218
  - FLMLRCIS 222
  - FLMLRDTL 232
  - FLMLRIPF 233
  - FLMLSS 236
  - FLMLTWST 240
  - FLMTBMAP 256
  - FLMTMSI 258
  - FLMTPRE 259
  - FLMTPST 261
  - FLMTXFER 263

## U

- UNLOCK service 92
- user data entries
  - array record 15
  - variable 275, 278
- utilities function, DBUTIL service 38

## V

- variable 273
- variables
  - CLIST 6
  - COBOL return code 21
  - description of 269
  - description of group 281
  - field names 271
  - FORTRAN 10
  - functions 271
  - initialize parameter 10, 12
  - ISPF, used by SCLM services 17
  - list of 270
  - Pascal 10
  - report 271
  - uses for 270
- VERDEL service 95
- verification
  - access key 65
  - authorization code 64
  - build output 65
  - group 64
  - predecessor 64
- VERINFO service 97
- VERRECOV service 100
- VSAM data set, specifying with
  - FLMCNTRL macro 152
- VSAM Record Level Sharing 155, 158
- VSAMRLS parameter 155, 158







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.

SC34-4818-01

