# SCLM GUIDE TO TEAMCONNECTION TERMINOLOGY

Document Number TR 29.2147

Mike Petersen

TeamConnection Integration
IBM Software Solutions
Research Triangle Park, N.C.

(c) Copyright International Business Machines Corporation 1996.
All rights reserved.

# TRADEMARKS AND SERVICE MARKS

The following terms, denoted by an asterisk (*) on their first occurrence in this publication, are trademarks or service marks of the IBM Corporation in the United States or other countries:

CICS

DB2

IBM

OS/2

TeamConnection

The following terms, denoted by two asterisks (**) on their first occurrence in this publication, are trademarks or service marks of other companies:

| | |
|---|---|
| Object Store | Object Design, Inc. |

# ABSTRACT

This report introduces the experienced IBM(*) ISPF SCLM (Software Configuration and Library Manager) user to the terminology used in the IBM TeamConnection(*) product. This report may be useful in migrating from SCLM to TeamConnection, migrating from TeamConnection to SCLM, and in exploring possible interfaces between SCLM and TeamConnection.

**ITIRC KEYWORDS**

Architecture definition

Build agent

Build processor

Build server

Configuration Management

Drivers

ISPF

ISPF 4.2

Language definition

Library Management

Project definition

SCLM

SCLM migration

Software Configuration and Library Manager

TeamConnection

Work areas

# ABOUT THE AUTHOR

Mike Petersen is an IBM software developer for TeamConnection integration in Raleigh, North Carolina. Mike was an SCLM software developer for six years. He has worked in systems, application, and other product programming areas. He also has had assignments with world trade, IBM faculty loan program, technical education, and technical support for marketing.

Mike received a BS in mathematics with a physics minor in 1972 and an MS in computer science in 1974 from Purdue University. In 1979 he received an MBA from Marist College.

# CONTENTS

**Trademarks and Service Marks** . . . . . iii

**ABSTRACT** . . . . . v

**ABOUT THE AUTHOR** . . . . . vii

**BEFORE WE BEGIN** . . . . . 1

**Data Storage** . . . . . 3

**Library Structure and PROMOTE** . . . . . 5

**Language Definitions and BUILD** . . . . . 9

**Architecture Definitions and Build Trees** . . . . . 13

**Authorization Codes and Components** . . . . . 17

**Exits and Processes** . . . . . 21
- SCLM User Exits and Translators . . . . . 21
- TeamConnection User Exits . . . . . 22
- TeamConnection Subprocesses . . . . . 22
- TeamConnection User Notification . . . . . 25

**Other Topics** . . . . . 27
- Project Definition and Family . . . . . 27
- Versions and Releases . . . . . 27
  - Versions . . . . . 27
  - Releases . . . . . 29
- Serial and Concurrent Development . . . . . 29
- Integration Test and Acceptance Test . . . . . 30
- BUILD and PROMOTE by change code . . . . . 30
- Reports . . . . . 31
- Command Line Interface . . . . . 31

**GLOSSARY** . . . . . 35

# BEFORE WE BEGIN

This report is written for the experienced SCLM user who needs to understand the terminology used in TeamConnection. Experience should include constructing and modifying project definitions, language definitions, and architecture definitions. This report may serve as an introduction for programmers and project leaders involved in migrating an existing application from SCLM libraries to TeamConnection.

An experienced TeamConnection user may find this report useful as an introduction to SCLM. However, the report is written from the perspective of an experienced SCLM user and will not go into SCLM details. SCLM experience from a class, working with the SCLM samples, and/or "hands on" work may be needed to derive full benefit from this report.

The information in this report is based upon ISPF 4.2 SCLM and TeamConnection Version 1, Release 0. This information may apply in whole, or in part, to other versions and releases of these products.

The chapters are organized for quick reference to specific topics. Some information is presented based upon information in a previous chapter. It is therefore recommended that all of the chapters be read in the order presented as a first reading unless the reader is an experienced SCLM user and has completed a TeamConnection class.

# DATA STORAGE

SCLM is a software configuration and library management system that directly controls data stored in partitioned data sets on an MVS mainframe. SCLM may indirectly control DB2(*) plans and CSP MSLs. Source and object code for workstation exe files may be stored in partitioned data sets. The current release of an application developed under SCLM is found in a collection of members of partitioned data sets (such as load modules and documentation) and parts that cannot be easily stored in partitioned data sets (such as DB2 plans and exe work station files). While one may be able to directly access the SCLM controlled libraries, the partitioned data set members comprising the delivered application are usually copied into other data sets for execution in a production environment.

TeamConnection is a software configuration and data management system that directly controls data stored as objects in an ObjectStore(**) database on an OS/2 work station. TeamConnection may indirectly control DB2 plans and load modules stored on an MVS mainframe. Source and object code for a mainframe application may be stored as objects in the ObjectStore database. The current release of an application developed under TeamConnection is found in a collection of objects (such as exe files and documentation) in an ObjectStore database and parts that cannot be easily stored in the ObjectStore database (such as MVS load modules). Since one may not directly access the objects in the ObjectStore database, the objects are extracted from the ObjectStore database into files for execution in a production environment.

Table 1. Data Storage Summary

| **SCLM** | **TeamConnection** |
|---|---|
| MVS product | OS/2 product |
| data stored as members of partitioned data sets | data stored as objects in ObjectStore database |
| may indirectly control DB2 plans | may indirectly control DB2 plans |
| source and object code for work station applications may be stored in SCLM controlled partitioned data sets | source and object code for mainframe applications may be stored as objects in the ObjectStore database |
| work station exe files are not easily stored in the SCLM controlled libraries | mainframe load modules are not easily stored in the ObjectStore database |

# LIBRARY STRUCTURE AND PROMOTE

SCLM library management uses a hierarchy of concatenated data sets. The top group in the hierarchy contains all of the members that are related to the current release of an application. The development groups at the bottom of the hierarchy contain the members that are being developed for the next version of the release. The concatenation of data sets from a development group up through the groups in the hierarchical path represents a version of the release that is under development.

An SCLM developer uses SCLM EDIT to lock an existing member in the hierarchical path and place a copy of the member into a development group data set. New members are created and locked using SCLM EDIT. The developer uses SCLM PROMOTE to move the members up to the next group in the hierarchy. SCLM PROMOTE unlocks the members for use by other developers.

TeamConnection groups the parts for an application into a release. A release is a logical organization of all parts that are related to an application. A work area is a window into the release and is used by developers to modify existing parts and create new parts for the next version of the release.

A TeamConnection developer checks out a part to a work area which results in a copy being placed in an OS/2 file. The developer modifies the part and checks it in to the work area. The check out and check in may be handled automatically through the use of the TeamConnection graphical user interface (GUI) with a specified editor. In TeamConnection's serial development the part will be automatically locked when it is checked out.

A new part is created by using a work area and the TeamConnection part create command.

The release may be updated with the new and changed parts in one of two ways:

1. integrate the work area into the release, or
2. add the work area as a member to a driver, commit the driver to the release, and complete the driver.

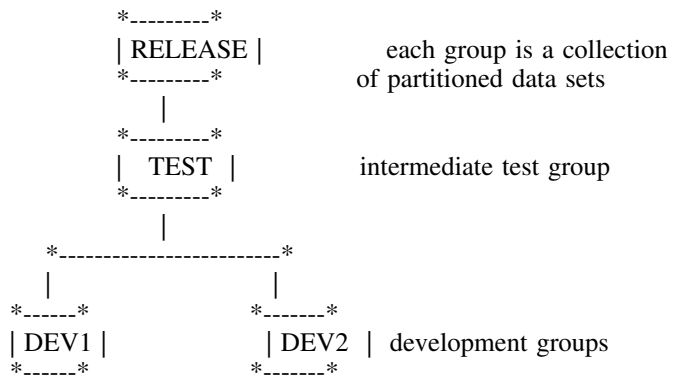
Both of these techniques will unlock the part as well as make the new and changed parts visible to other developers.

Work area and driver names may not be reused after integration and commit activities.

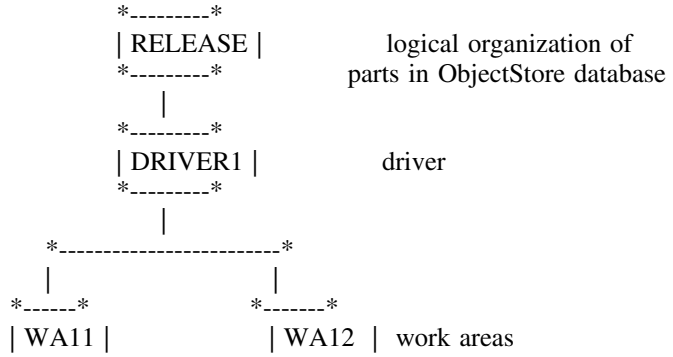
Table 2. Library Structure and PROMOTE Summary

| SCLM | TeamConnection |
|---|---|
| release group (top group in hierarchy) | release |
| development group (bottom group in hierarchy) | work area |
| intermediate test group (middle group in hierarchy) | driver |
| PROMOTE a development group directly to the release group | integrate a work area into a release |
| PROMOTE a development group to a test group | add a work area as a member of a driver |
| PROMOTE a test group to the release group | commit and complete a driver |
| SCLM EDIT | TeamConnection GUI with a specified editor |
| draw down a member | check out a part to a work area |
| group names do not change after PROMOTE | work area and driver names may not be reused after integrate and commit activities |

EXAMPLE: SCLM Library Hierarchy




EXAMPLE: TeamConnection Database



*------*  *-------*

# LANGUAGE DEFINITIONS AND BUILD

SCLM uses language definitions to identify translators to be used for a BUILD of a member and the outputs to be produced from a BUILD. The creation of a part involves identifying the language of the part. This language value is stored in the part accounting record. The language definition with a matching LANG keyword value is used for the BUILD of the part.

Language definitions also identify parsers which are translators that identify dependencies when a member is created or changed. The dependencies are stored in the accounting record of the member.

During a BUILD, SCLM passes control to load modules, TSO commands, REXX execs, and CLISTs. These translators may then execute other tools that result in the compilation of source code on MVS or a work station, DB2 plan binds, link edits of object code, and other transformations of data.

The language attribute of a member may be changed during SCLM EDIT. The next BUILD will build the member with the new language definition even if the member itself was not changed. Similarly, the VERSION keyword in a language definition may be changed so that all members of that language will be built during the next BUILD even if the members were not changed. In each situation the members involved will be automatically rebuilt during the next BUILD with the new language definition.

TeamConnection uses a build script to identify the tools to be used for a BUILD of a member. The build script may be a command file, a JCL stream, or a simple command. The build script is part of a builder object. Each builder has a name and this name is attached to an output part such as object code or an exe. This is a significant difference from SCLM which associates a language with an input part. This also means that one must create empty output parts and attach builder names before the first BUILD.

TeamConnection uses parsers to identify dependencies of a part. Parsers are created in a fashion similar to the creation of builders. Unlike a builder that is attached to an output part, a parser is attached to an input part. The actual parse of the part occurs at BUILD and the dependencies are added to the build tree (similar to an SCLM architecture definition) for the application at that time.

A TeamConnection BUILD:

1. build tree is examined for all parts that have changed since the last build or will be affected by the build
2. build event for each part needing to be built is put on a job queue in the ObjectStore database

3. software called build agents poll the job queue and remove the build events
4. build agent sends the build script and the needed parts to other software called a build processor
5. build processor executes the build script together with the input parts received from the build agent and sends the output parts back to the build agent
6. build agents store the output parts in the ObjectStore database

Build agents and build processors always exist as pairs. A build agent/build processor pair is called a build server. It is possible for the build agents and build processors to be on different machines in a client/server network. In particular, the build processor for an MVS build is a job that is running on an MVS server machine awaiting the arrival of a build script and input parts from a build agent that is running on an OS/2 machine.

SCLM may pass parameters to translators in four ways:

1. FLMTRNSL OPTIONS keyword with variables
2. FLMTOPTS macro may be used to append or replace the parameter string created by the OPTIONS value.
3. PARM architecture definition statement parameter string to be appended to the end of the FLMTRNSL OPTIONS keyword value
4. PARMx architecture definition statement parameter string to be appended to the end of the parameter string created by the concatenation of the OPTIONS keyword value and the PARM architecture definition statement parameter string

TeamConnection may pass parameters to the build script in five ways:

1. Parameters attribute of the builder with variables (similar to the SCLM FLMTRNSL OPTIONS keyword value)
2. As attributes of parts in the build tree (similar to the SCLM PARM and PARMx architecture definition statements) which replace the value of the Parameters attribute in the builder
3. Parameters specified for any part in a build tree that is an output of a build event will be applied to all outputs of the build event
4. Substitute the builder "Parameters" field value into the file parameters by using the variable $(BIUILDERPARMS) in the teamc part -build command
5. Set environment variables in the teamc part -build command that may be retrieved by the build scripts.

Both SCLM and TeamConnection have similar BUILD modes with the same results:

1. CONDITIONAL(SCLM) and NORMAL(TeamConnection)

2. UNCONDITIONAL
3. FORCED
4. REPORT

The success of a BUILD is determined by the return code of a translator in SCLM. The SCLM FLMTRNSL GOODRC keyword value is the maximum permitted return code for the related translator. The TeamConnection builder has a comparison operator attribute (LT, LE, EQ, GE, GT, NE) as well as a return code attribute. Use of the COND EXEC statement keyword parameter is the TeamConnection counterpart of the SCLM FLMTCOND macro.

Table 3. Language Definition and BUILD Summary

| SCLM | TeamConnection |
|---|---|
| language definition | build script |
| FLMLANGL macro LANG keyword | builder name |
| language name associated with a member to be transformed into one or more outputs (such as a source part) | builder name associated with an output part (such as an obj part) |
| output members will be created as needed during BUILD | output members must exist prior to BUILD and may be empty prior to the first BUILD |
| parsers identified in language definitions | PARSERs created like BUILDERs and attached to input parts |
| members parsed when migrated into SCLM or saved during SCLM EDIT | members parsed as part of BUILD |
| FLMLANGL VERSION keyword may be changed to cause a rebuild of all members with a language value equal to the LANG keyword value | Modification of the builder will result in parts with the builder attached to be rebuilt |
| Modification of the translators will not result in a rebuild of the members with the associated language name | Modification of the build script will not result in a rebuild of the parts with the attached builder name |
| FLMTRNSL OPTIONS keyword and FLMTOPTS macro | Parameter attribute of the builder |
| PARM and PARMx architecture definition statements | attributes of parts in a build tree |
| CONDITIONAL, UNCONDITIONAL, FORCED, and REPORT BUILD modes | NORMAL, UNCONDITIONAL, FORCED, and REPORT BUILD modes |
| FLMTRNSL GOODRC keyword for each translator | Return code and Comparison operator attributes for each builder |
| FLMTCOND macro for conditional execution of translators | COND EXEC statement keyword parameter for conditional execution of translators |

# ARCHITECTURE DEFINITIONS AND BUILD TREES

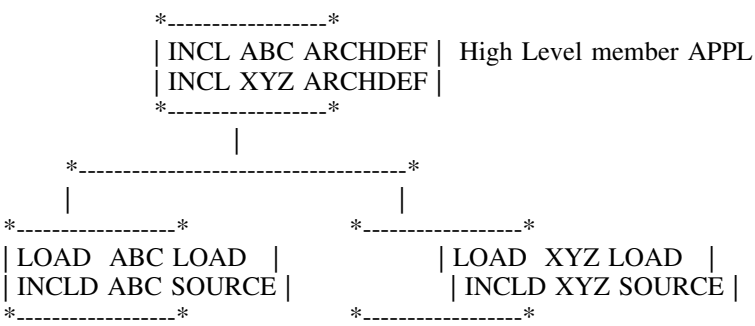
SCLM uses architecture definitions to describe the configuration of an application. Typically one builds a high level architecture definition that references one or more subsets of an application or the entire application. The architecture definition itself is controlled by SCLM and as such any changes to it will result in a build of the affected members. Each architecture definition member has a language attribute (usually ARCHDEF).

TeamConnection uses build trees to describe the configuration of an application. Typically one builds a collector part that references one or more subsets of an application or the entire application. The collector part is the final output of the build and as such it requires a builder. The null builder is used as there is no build script to be executed for a collector part.

Table 4. Architecture Definitions and Build Tree Summary

| **SCLM** | **TeamConnection** |
|---|---|
| high level member | collector part |
| source code dependencies are found in accounting records | source code dependencies are found in build trees |
| source code dependencies are identified by a parser | source code dependencies are identified by a parser |
| source code dependencies may not be manually updated in the accounting record | source code dependencies may be manually updated in the build tree |
| INCL statements used to create dependencies between architecture definition members | TeamConnection GUI or teamc part -connect command used to connect parts to parents in the build tree |

EXAMPLE: SCLM Architecture Definition




Assume: SOURCE member ABC has a dependency upon SOURCE member COPYABC
NOTE: COPYABC does not appear in any architecture
definition statements

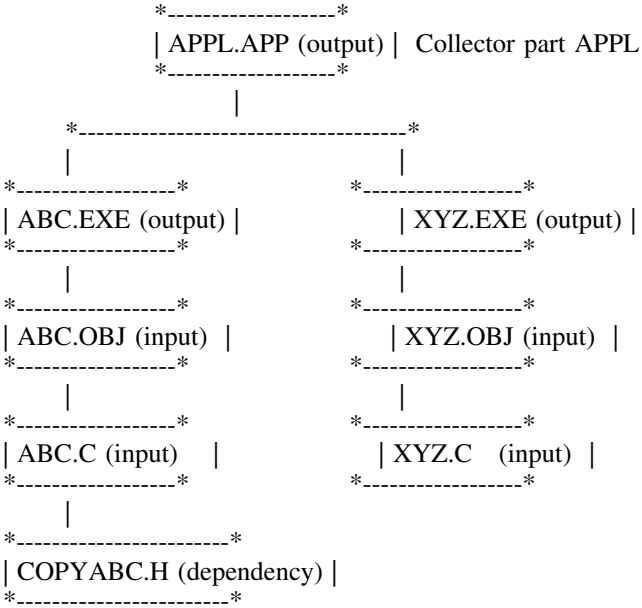
language definition for SOURCE members ABC and XYZ has a build
translator that produces object code output OBJ members ABC, XYZ

SOURCE members ABC and XYZ have been modified since last BUILD

A BUILD of architecture definition APPL will produce object code
OBJ members ABC and XYZ as well as load modules LOAD members ABC and XYZ.

A BUILD of only SOURCE member ABC will result in the translation of only
SOURCE member ABC which will produce object code output OBJ member ABC.

EXAMPLE: TeamConnection Build Tree




Assume: ABC.C and XYZ.C have been modified since the last build
NOTE: The parse of ABC.C results in the addition of COPYABC.H
to the build tree.

A BUILD of collector part APPL.APP will first parse ABC.C and XYZ.C. Build scripts will then invoke tools to produce ABC.OBJ and XYZ.OBJ. Finally build scripts will produce ABC.EXE and XYZ.EXE. ABC.OBJ, XYZ.OBJ, ABC.EXE, and XYZ.EXE will be stored in the ObjectStore database. They must be extracted from the ObjectStore database in order to be tested.

A BUILD of only part ABC.OBJ will result in a parse of part ABC.C followed by the execution of the build script for part ABC.C and creation of output part ABC.OBJ.

# AUTHORIZATION CODES AND COMPONENTS

SCLM does not have a built-in system for restricting usage of commands other than that provided by authorization codes. The purpose of authorization codes is to restrict promotions. In TeamConnection, permitted actions are specified in access authority groups. The counterpart of the SCLM authorization code is the TeamConnection access authority group.

A TeamConnection access authority group is a set of TeamConnection actions that will be permitted for a user assigned to that access authority group. An access list is a list of TeamConnection users and their assigned access authority groups. TeamConnection provides access authority groups that may be used "as is" and/or modified. Additional access authority groups may be created to meet the needs of the organization. The TeamConnection access list is part of a more extensive control structure than the SCLM group and associated authorization codes.

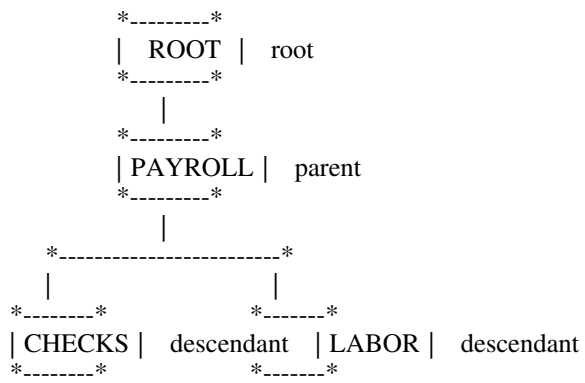
We must first discuss components before we can examine the full use of access lists. TeamConnection uses components to organize development data, control access to the data, and notify users when certain actions occur. Each component has an owner who is responsible for managing data related to it. This data includes defects, features, and the component access list.

Components are organized in a hierarchy. Parts such as source code are controlled by components. The access list for a component determines the explicit authority users will have to perform actions upon parts controlled by the component and its descendants.

At first glance, a TeamConnection component hierarchy resembles an SCLM library hierarchy. THEY ARE NOT THE SAME THING. There is no SCLM counterpart to a TeamConnection component hierarchy.

Components control access to data. In a sense a source code part belongs to one and only one component. The component hierarchy may be organized so that the components control data by subapplication, application development team, or other logical combinations. An organization by subapplication is useful for situations in which all of the parts for a subapplication need to be controlled with restricted access such as a payroll application. An organization by application development team is useful for situations in which different processes are needed for different teams such as emergency fix, test, development, and prototype.

EXAMPLE: TeamConnection Component Hierarchy Organized by Subapplications

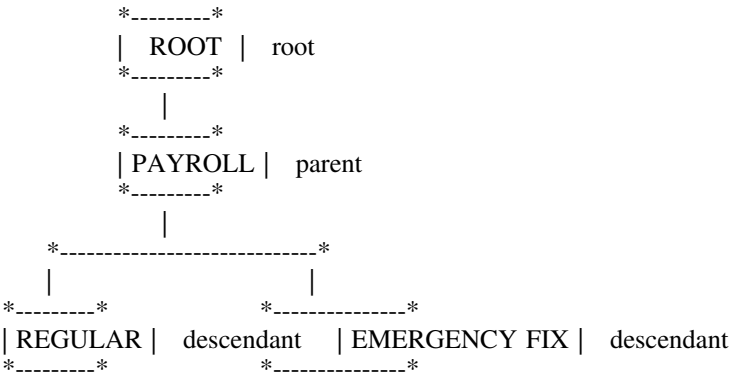



In this component hierarchy, users who work on the LABOR subapplications would appear in the access list for the LABOR component. All parts for the LABOR subapplication would be controlled by the LABOR component. Similarly, users who work on the CHECKS subapplications would appear in the access list for the CHECKS component. All parts for the CHECKS subapplication would be controlled by the CHECKS component.

A few users (such as project managers, team leaders) would need explicit access to parts controlled by both CHECKS and LABOR. For these users they would appear in the access list for the PAYROLL component which, by the nature of the component hierarchy, would also give them inherited access to the descendant components (CHECKS and LABOR).

In a more complex structure one may restrict access for a user in a descendant component so that the user could have inherited access to all but a few components.

EXAMPLE: TeamConnection Component Hierarchy Organized by Development Team




In this component hierarchy the REGULAR component would have strict TeamConnection development processes (to be discussed later) so that defects and features are addressed in a controlled fashion. The EMERGENCY FIX component would have only a few TeamConnection development processes so that emergency fixes may be quickly addressed.

Users may have access to PAYROLL, REGULAR, and EMERGENCY FIX as required. Users in the PAYROLL component access list will have inherited access to the data controlled by the REGULAR and EMERGENCY FIX components.

# EXITS AND PROCESSES

## SCLM USER EXITS AND TRANSLATORS

SCLM provides user exits that may be used to perform additional processing as part of the application development process. These exits provide a means of additional validation and approvals before SCLM proceeds to the next step in EDIT, BUILD, or PROMOTE. The exit routines must be provided by the customer and may be used to perform logging functions, additional verification, notification, or coordinate processing with non-SCLM tools. SCLM exits and possible use in the application development process are:

change code verification routine

A change code verification routine is called by SCLM when a developer attempts to update a member. As a minimum, an IEFBR14 type of program will require the developer to enter a change code. A more sophisticated routine may verify that the change code provided by the developer is correct, create a defect or feature record in an information database, and/or notify other users of the change activity.

BLDEXT1

This exit routine is called at the end of the BUILD. It may be used for logging information and/or notifying others of the BUILD activity. It may return any positive integer value without disrupting normal processing.

PRMEXT1

This exit routine is called after the PROMOTE verification phase. It may be used for saving members in the "to-group" to a private data set before the copy operation, stopping the PROMOTE at this point, and/or notifying other users of the PROMOTE activity.

PRMEXT2

This exit routine is called after the PROMOTE copy operation. It may be used for logging information about the copied members, stopping the PROMOTE at this point, and/or notifying other users of the PROMOTE activity.

PRMEXT3

This exit is called after the PROMOTE purge activity. It may be used for logging information and/or notifying others of the PROMOTE activity. It may return any positive integer value without disrupting normal processing.

Additional process control may utilize user supplied translators to be called during the PARSE, BUILD, VERIFY, COPY, and PURGE functions.

TeamConnection provides user exits for additional actions and subprocesses for the application development environment. The TeamConnection subprocesses provide built-in functions that would require extensive use of user written exits in SCLM. TeamConnection also has a built-in notification function.

## TEAMCONNECTION USER EXITS

All developer activities in TeamConnection are accomplished through TeamConnection actions. The TeamConnection GUI is a front-end to a process that builds and executes commands for the TeamConnection actions. These commands may be executed from a command line, a command file, or the TeamConnection GUI. A user exit is a user-defined program that the TeamConnection server will call during the processing of TeamConnection actions.

The user exits may be a command file or an executable file. The exit may be used to provide additional verification or start non-TeamConnection activities. Most TeamConnection actions support user exits.

User exits may be called at the following times for an action:

prior to initialization and access checking (exit ID 0)

after all checks are made and before command processing (exit ID 1)

after action has completed

after an unsuccessful user exit with exit ID 0 or exit ID 1 or after an unsuccessful action (this user exit may be used to clean up what the other two exits started)

In addition to a greater number of possible user exits as compared to SCLM, TeamConnection also provides a greater number of parameters to the user exit for even more flexibility.

## TEAMCONNECTION SUBPROCESSES

TeamConnection provides preconfigured processes for application development that may be turned on and off at any time. In practice, one may start without any controls for developing a prototype and then increase the controls by enabling additional processes as the team moves into formal development of the application. There are no routines to write to use these processes.

TeamConnection preconfigured processes are combinations of different sets of subprocesses. These subprocesses are for the control of application development activities. The processes may be modified and new processes configured in order to meet the needs of the organization.

TeamConnection provides a mechanism for logging defects and features. Defects are for fixing problems and features are for adding new functions. These may be used without any

processes turned on and provide a suitable means for manually keeping track of what changes are being made. One merely opens a defect or feature at the beginning of the development process and closes it at the end. Of course in a manual system one may forget to open or close these objects.

The TeamConnection subprocesses may be used to tie the defects and features to the work areas as well as introducing approval, fix, verification, and test records at appropriate times for electronic signatures. This will help to remind people to open and close defects and features, verify fixes, and perform tests. It will also provide automatic notification of various development activities to the appropriate team members (notification will be discussed later).

The TeamConnection subprocesses provide a much greater degree of control over the application development environment than the SCLM exits.

Table 5 (Page 1 of 2). TeamConnection Subprocesses for Components

| **Component Subprocess** | **Description** |
|---|---|
| dsrDefect | Design, size, and review fixes to be made for defects.<br>design text must be entered before the defect may be moved to the size state<br>sizing record must be created for each release that requires part changes for the defect<br>review design text and sizing records |
| dsrFeature | Design, size, and review enhancements to be made for features.<br>design text must be entered before the feature may be moved to the size state<br>sizing record must be created for each release that requires part changes for the feature<br>review design text and sizing records |
| verifyDefect | TeamConnection creates a verification record when the defect is accepted. The defect cannot be closed until the originator of the defect:<br>accepts the fix<br>rejects the fix<br>abstains from making a decision - tacit approval<br>A new defect will need to be opened if the defect fix is rejected. |

Table 5 (Page 2 of 2). TeamConnection Subprocesses for Components

| Component Subprocess | Description |
|---|---|
| verifyFeature | TeamConnection creates a verification record when the feature is accepted. The feature cannot be closed until the originator of the defect:<br>accepts the new function<br>rejects the new function<br>abstains from making a decision - tacit approval<br>A new feature will need to be opened if the feature is rejected. |

Table 6. TeamConnection Subprocesses for Releases

| Release Sub-process | Description |
|---|---|
| track | A work area must be associated with a specific defect or feature. A work area cannot be created without first opening a defect or feature. |
| approval (must be used with track) | TeamConnection creates an approval record for each user in the approver list for the release. Each approver:<br>accepts continued work<br>rejects continued work<br>abstains from making a decision - tacit approval<br>The work area will move to the fix state after all approvers have accepted or abstained to continue work. Work cannot begin in the work area until the work area is in the fix state. |
| fix (must be used with track) | TeamConnection creates fix records when parts are checked in (created or checked out and modified). A work area may not be integrated to the release or added to a driver until all fix records are complete. |
| driver (must be used with track) | The driver subprocess allows one to create drivers. Work areas may be added to drivers incrementally so that their impact can be evaluated and verified before additional changes are incorporated. |
| test (must be used with track) | TeamConnection creates test records. Work areas will move to the test state when they are integrated to a release or when their driver is completed. The test records must be accepted before the work area will move to the complete state. |

## TEAMCONNECTION USER NOTIFICATION

TeamConnection has over 150 actions. Actions such as CompCreate (component create), TestReady, and VerifyReady will automatically notify the respective owners by sending a notification message. This message is sent to an electronic mailing address that was specified when the owner's TeamConnection userid was created. This automatic notification is called implicit notification.

TeamConnection users may be interested in actions such as DefectOpen, FixComplete, and PartDelete. TeamConnection provides interest notification groups that are sets of actions that may be of interest to managers, developers, builders, and testers. The interest notification groups may be used "as is" or may be modified. Additional interest notification groups may be added to meet the needs of the organization.

Each component has a notification list that controls who is notified of what actions. This list contains the users and their related interest notification group. When a user is to be notified that a specific action occurred within one component, that user will also be notified when that action occurs in any of the descendant components.

The mechanism for sending the notification message to the electronic address associated with a user is through a notification server that must be running. The mail exit routine to be used for handling mail requests is specified when the notification server is started.

# OTHER TOPICS

## PROJECT DEFINITION AND FAMILY

The SCLM project definition defines the library hierarchy, the language definitions, and the translators to be used. Usually the project definition is stored in libraries separate from the application data and architecture definitions. One may use several project definitions (called alternate project definitions) to work with the same application data and architecture definition data sets.

The SCLM project definition is usually self-contained and the libraries are not shared among project definitions that are not alternates. At times it is difficult to determine how many project definitions are needed for an organization. Once two separate project definitions are under way it is difficult to combine them into one project definition and one set of libraries.

A TeamConnection family represents a complete and self-contained collection of TeamConnection users and development data. One family cannot share data with another. At times it is difficult to determine how many families are needed for an organization. Once two families are under way it is difficult to combine them into one family.

The TeamConnection family is controlled by a family server that runs on a machine sometimes known as the TeamConnection server. The ObjectStore database is usually located on the same machine. Unlike SCLM where a project is represented by a collection of partitioned data sets, a TeamConnection family has everything located in one ObjectStore database. The name of the database is family_name.tcd. This database is easily saved for backup purposes or for preserving a starting point for test and demonstration purposes.

## VERSIONS AND RELEASES

### Versions

The SCLM versioning capability stores versions of data in partitioned data sets. Previous versions of source members may be retrieved to a sequential data set, a member of a partitioned data set, or a member of a partitioned data set controlled by SCLM. In order to recreate a previous version of a load module one must be sure to retrieve the proper versions of all members (including architecture definitions) related to the BUILD of the load module as they existed at that time in the past. The BUILD may not produce the same load module if the language definitions and/or translators have changed since that time in the past.

SCLM uses a concatenation of data sets as found in the hierarchical path. As other developers PROMOTE into groups in the path, the concatenation automatically reflects the changes. It is possible for a developer to have an SCLM BUILD involve more members than those affected by his changes if other developers have promoted other members that are

affected by the changes made by the developer. It is a good idea to issue a BUILD report prior to a BUILD in order to determine all of the members that have changed since the last BUILD.

TeamConnection has a built-in versioning facility. Each time a work area is integrated to a release or a driver committed to a release, TeamConnection will create a version of the release. TeamConnection appends a ":n" to the release name starting with the second version of the release. Thus if the first version of the release is "xyz" then the second version will be "xyz:2", the third version "xyz:3", and so on. However, this convention is only used for retrieving past versions. The version name "xyz" in the TeamConnection GUI and commands is used to refer to the current version.

At times a developer will reach a point in a work area where the changes made up to that point are stable but additional work remains. The developer may "freeze" the work area before making additional changes. This action will create a version of the work area so that the developer may return to it if the additional changes do not work out. This technique will assist the developer in returning to a known point of time in the work area without having to back out changes. The same naming convention is used for work area versions as that used for release versions.

In a similar fashion to SCLM, retrieval of a previous version of a source code part involves extracting the previous version, checking out the current version, replacing the current version with the extracted part, and then checking the part back in to the work area. Parts may be retrieved from previous versions of releases and from previous work areas within those versions. The versions of a part within a work area will go away if the release is pruned or if autopruning is on at the time of the work area integration. Usually the version of a part at the time of the work area integration will be sufficient for recovery as the versions within a work area are usually intermediate results that do not contain a complete fix.

In TeamConnection a work area is a window into the release. This view of the release does not change as other developers integrate their work areas (or commit drivers) into the release. A BUILD may involve an old version of the release. In order to update the view of the release as seen by the work area, the developer may perform a work area "refresh". This action will freeze the work area if it has not already been frozen and update the view so that the current versions of the parts are visible. It is a good idea to refresh a work area before a BUILD in order to be building with the current version of other parts in the release. A work area cannot be integrated into a release if it was built with an old version of the release.

Drivers may also be refreshed prior to a BUILD of a driver.

### Releases

It is difficult in SCLM to start a new release within the same project definition. One way of doing this is to make the top group a non-key group and then add another group on top for the next release. Additional test and development groups would need to be added for the new release. The need to update a member in the new release while leaving it unchanged in the old release would result in a new name for the member that is no longer identical to both releases. Language definitions also change and the use of an alternate project definition may be required.

A new release may be handled in SCLM by creating a new project definition that starts out identical to the old with the same data. By not sharing data between the two project definitions, one may maintain the same member names. However, a change to a member in the new release may result in the need for an identical change in the old release when one needs to fix a defect that is common to both releases. There is some manual control needed.

In TeamConnection the new release may be created in the same family as the old release. By creating an initial work area in the new release and linking it to the old release one may start out with the new release equal to the old. The development data, component hierarchy, and related lists stay the same. A change to a part for the old release will be reflected in the new release and vice versa.

At some point a change will be required to a part that affects only one release. It is at that point that one may break the link of the part to both releases. The part will exist by the same name in both releases, but will have different contents due to the changes for defects and features. One may then maintain an old release and develop a new release without changing the names of parts and without making duplicate changes to the parts that are common to both releases.

## SERIAL AND CONCURRENT DEVELOPMENT

SCLM provides a merge tool so that two or more developers may work on the same member at the same time. One developer draws down the member with an authorization code that permits a PROMOTE while the other developers draw down the same part with an authorization code that prevents a promote. The various members in the development groups are later merged together. This is concurrent development.

Serial development is accomplished by not permitting two or more developers to draw down the same part at the same time. This is accomplished by using a library hierarchy that has no extra authorization codes in the development groups.

The SCLM library hierarchy may be constructed such that some development groups may be used for concurrent development while others are restricted to serial development.

A TeamConnection release is created with serial or concurrent development enabled. It is not possible to change back and forth within a release. As with SCLM there is a merge tool that a developer may use to merge his changes with those of another developer. Unlike SCLM, in concurrent development the developers may not be aware that other developers are changing the same member until such time that they refresh their work area, integrate their work area into a release, or add their work area to a driver. (In SCLM one must at least specify the authorization code before drawing down a member that is locked by another developer.)

## INTEGRATION TEST AND ACCEPTANCE TEST

The SCLM library hierarchy may be structured so that development groups PROMOTE to an integration test group and integration test groups PROMOTE to a system test group. The system test groups would then PROMOTE to the release group at the top of the hierarchy. SCLM permits a finitely large number of groups in any hierarchical path.

TeamConnection may be used to mirror this environment through the use of drivers. One may have a set of drivers for integration test and another set of drivers for system test. The application development process could then involve:

1. developers add their work areas as members to an integration driver
2. after integration test, work areas are added to a system test driver and removed from an integration test driver
3. after system test, the system test driver is committed to the release and a new system test driver is created for the next cycle (recall that driver names of completed drivers may not be reused)

A REXX exec may be written to move the work areas from one driver to the next. Just as one may have a finitely large number of SCLM groups and SCLM hierarchical paths, one may have a finitely large number of drivers.

## BUILD AND PROMOTE BY CHANGE CODE

In SCLM one may use a change code to indicate the defect or feature being addressed by the changes. At some point one may decide that the changes for a subset of the defects and features are ready for a BUILD or PROMOTE. SCLM provides a facility for specifying by change code the desired changes for a BUILD or PROMOTE so that not all changes are involved.

In TeamConnection one may use the defect or feature name as the name of a work area. The use of drivers permits one to add or remove work areas as desired from a driver. Unlike SCLM, there is no automatic facility for specifying the defects and features of interest in a driver BUILD or COMMIT. However, one may be able to write a REXX exec to accomplish this task. The key is to add only those work areas of interest to the driver.

## REPORTS

SCLM provides a report facility for producing customized reports. One may specify patterns to act as filters in order to produce a report that focuses on a subset of the information.

TeamConnection has an extensive report facility that uses the Standard Query Language (SQL). One may use the TeamConnection GUI to generate a report and then modify the SQL query that is displayed at the bottom of the report window as well as using the command line and typing in the desired SQL query. These queries provide a filter so that one may focus on a subset of the information.

## COMMAND LINE INTERFACE

Both SCLM and TeamConnection have a command line interface. The difference is that in TeamConnection, use of the GUI results in TeamConnection commands being executed. These commands are stored in a log file. One may use the TeamConnection GUI to complete a set of tasks and then create a command file from the log file that may be used in place of the GUI in the future. This is useful when many iterations of a sophisticated task are required.

The following examples are based upon the examples in chapter 3 "Library Structure and PROMOTE" and in chapter 5 "Architecture Definitions and Build Trees".

EXAMPLE: commands for SCLM and TeamConnection BUILD

```
FLMCMD BUILD,PROJECT,,DEV1,ARCHDEF,APPL
teamc  part -build APPL.APP -family FAMILY -workarea WA11 -pool pool1 -release RELEASE
```

EXAMPLE: commands for SCLM and TeamConnection part delete

```
FLMCMD DELETE,PROJECT,,DEV1,SOURCE,ABC,ACCESKEY
teamc  part -delete ABC.C -family FAMILY -workarea WA11 -release RELEASE
```

EXAMPLE: commands for SCLM and TeamConnection part lock

```
FLMCMD LOCK,PROJECT,,DEV1,SOURCE,ABC,,ACCESKEY
teamc  part -lock ABC.C -family FAMILY -workarea WA11 -release RELEASE
```

EXAMPLE: commands for SCLM PROMOTE and TeamConnection driver commit

```
FLMCMD PROMOTE,PROJECT,,TEST,ARCHDEF,APPL
teamc  driver -commit DRIVER1 -family FAMILY -release RELEASE
```

# GLOSSARY

**Access Authority Group**. An access authority group is a set of TeamConnection actions. An access list is a list of TeamConnection users and their related access authority groups.

**Access List**. An access list is an object controlled by a component. An access list is a list of TeamConnection users and their related access authority groups. The access list describes the actions a user may perform upon data controlled by the component or any descendant components. A user has explicit authority to perform these actions.

**Build Agent**. A build agent is software on an OS/2 workstation that polls the TeamConnection job queue for build events. A build agent removes build events from the job queue, sends the build script and input files to a build processor, and stores the returned outputs in the TeamConnection ObjectStore database.

**Build Processor**. A build processor is software on a machine in a client/server network that waits for a build script and input files from a build agent. A build server executes the build script and sends the outputs back to the build agent.

**Build Server**. Build agents and build processors always exist in pairs. A build agent/build processor pair is called a build server.

**Builder**. A builder is a TeamConnection object that identifies a build script. A TeamConnection builder plays a role similar to that of an SCLM language definition.

**Component**. A TeamConnection component organizes development data, controls access to the data, and notifies users when certain actions occur.

**ISPF**. Interactive System Productivity Facility

**JCL**. Job Control Language

**Language Definition**. A 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. An SCLM language definition plays a role similar to that of a TeamConnection builder.

**PDF**. The Program Development Facility of ISPF.

**project definition**. A 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.

**SCLM**. The Software Configuration and Library Manager is a component of the ISPF/PDF product. SCLM was first introduced in ISPF/PDF Version 3 Release 1.

**Translator**. A translator is 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.