CA Endevor® Software Change Manager

Automated Configuration Option Guide

Version 17.0.00

CA Technologies Product References

This document references the following CA Technologies products:

- CA Endevor® Software Change Manager (CA Endevor SCM)
- CA Endevor® Software Change Manager Automated Configuration (CA Endevor Automated Configuration)

Contact CA Technologies

Contact CA Support

For your convenience, CA Technologies provides one site where you can access the information that you need for your Home Office, Small Business, and Enterprise CA Technologies products. At http://ca.com/support, you can access the following resources:

- Online and telephone contact information for technical assistance and customer services
- Information about user communities and forums
- Product and documentation downloads
- CA Support policies and guidelines
- Other helpful resources appropriate for your product

Providing Feedback About Product Documentation

If you have comments or questions about CA Technologies product documentation, you can send a message to techpubs@ca.com.

To provide feedback about CA Technologies product documentation, complete our short customer survey which is available on the CA Support website at http://ca.com/docs.

Documentation Changes

The following documentation updates have been made since the last release of this documentation:

Note: In PDF format, page references identify the first page of the topic in which a change was made. The actual change may appear on a later page.

Version 16.0

- Summary of Levels Panel (see page 46)— Updated to change the version and level format to VVLL.
- Component List Panel (see page 47)— Updated to change the version and level format to VVLL.
- How to Tailor and Submit the Generated SCL for Execution (see page 92)— Updated to change the version and level format to VVLL.
- How to Perform Impact Analysis (see page 81), How to Propagate a Component Change to All Affected Programs (see page 83), and How to Validate a System for Consistent Use of Components (see page 87)— Updated with notes to state that the Autogen action option and the Validate action are more efficient methods of propagating components and validating components.
- Autogen Action Option (see page 85)— Updated to include Autogen Span options in the description for Autogen. Added a cross-reference to the scenario "How to Automatically Generate "Using" Elements, in the *Scenario Guide*.
- Validate Components (see page 90)— Updated to remove an obsolete statement about there being "no component support for long names."

Release 15.1

- How ACM Stores Component Lists (see page 40)—Updated to clarify that component information is recorded for all executed processor steps.
- How to Propagate a Component Change to All Affected Programs (see page 83)— Updated to add a cross reference to Autogen Action Option.
- Autogen Action Option (see page 85)— Added to describe this easier way to propagate a component change to all affected programs.
- How to Validate a System for Consistent Use of Components (see page 87)— Updated to add a cross reference to Validate Components.
- Validate Components (see page 90)— Added to describe this easier way to validate a system for consistent use of components"

Version 15.0

- Load Root and Cross-Reference Data Sets (see page 25)—Updated to add a note that this procedure can also be used to reload the data sets.
- Maintenance of Root and Cross-Reference Data Sets (see page 25)—Updated to add the back up option and rewritten for clarification. Retitled from Maintaining the Root and Cross-Reference Data Sets.
  - How to Reorganize Root and Cross-Reference Data Sets (see page 26)—Added to include and clarify information previously in the topic Maintaining the Root and Cross-Reference Data Sets.
  - How to Create Backup Copies of the ACM Query Data Sets (see page 27)—Added to describe the backup procedure.
  - How to Synchronize ACM Query Data with Inventory Data (see page 28)—Added to include and clarify information previously in the topic Maintaining the Root and Cross-Reference Data Sets.
- Set Build Action Generate (see page 69)—Updated to add the NoSource option to the diagram.

Contents

Chapter 1: Introduction

CA Endevor Automated Configuration (ACM) works with CA Endevor SCM Change Manager (CA Endevor SCM) to enable you to manage, through advanced automated technology, the interrelationships between software components.

This section contains the following topics:

What You Need to Know

To use CA Endevor Automated Configuration Manager (ACM), it is assumed that:

- The CA Endevor SCM system has been installed in accordance with the instructions provided in the *Installation Guide*.
- You have an understanding of the CA Endevor SCM environment in your organization.
- You have an understanding of software inventory management and application migration techniques.

Software Configuration Management

A typical software inventory is comprised of numerous application programs and the components—the common code (such as COBOL copybooks, Assembler macros, and CALLed subroutines)—which make up those programs. The discipline of managing the interrelationships between programs and their associated components is called *software configuration management*.

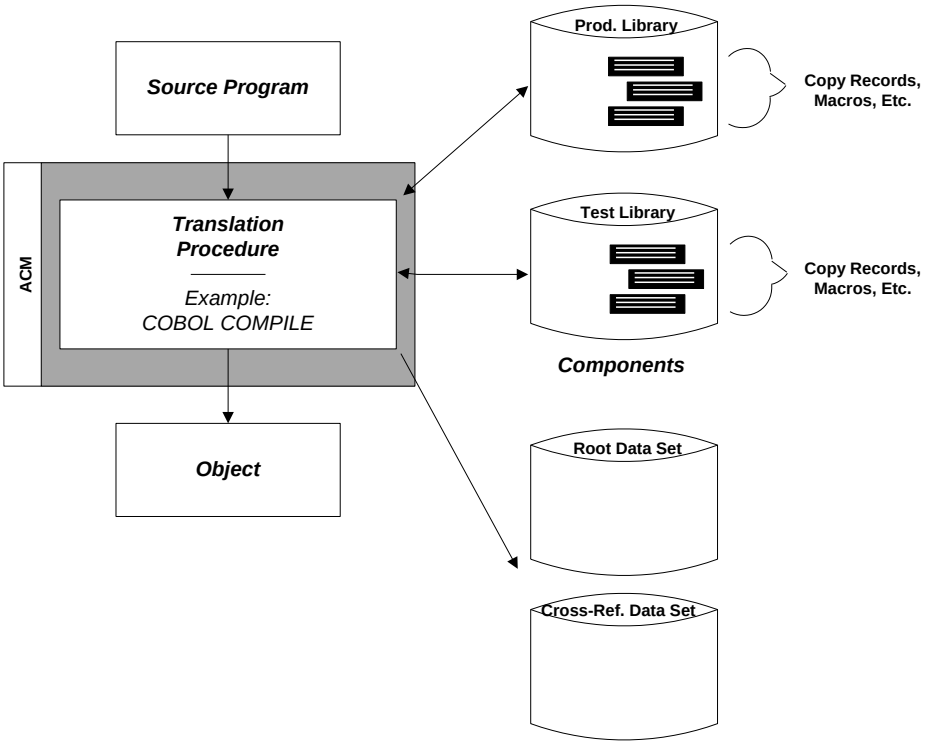
When a program is translated (for example, compiled, link-edited) from source to executable, all of the components which make up that program are extracted by language translators and/or linkage editors from their resident libraries, such as copylibs or maclibs. The major challenge is ensuring that the relationship between a program and its related components is accurate and up-to-date, at any given point in time. This synchronization is typically accomplished by retranslating a program each time one of its subordinate components is changed. A problem arises, however, when a component is changed but that modification is not propagated to all affected programs.

The lack of automated, reliable methods for coordinating program-component relationships has created major obstacles to software development and maintenance. Manual definition of program-component relationships is error-prone and time-consuming by today's standards. In addition, manual methodologies do not yield the level of component detail (version/ level/ location) needed to accurately plan and stage application development activities.

Traditional approaches to software configuration management have been severely limited in their ability to produce accurate and reliable results. Source scanning techniques, while an improvement over manual definition, provide only a static "picture" of the program-component relationship. This picture may not relate to the actual state of the software at the time of program translation. Source scanning is a separate operation run independently of the compile process and, as such, is only accurate at the time of the scan. Furthermore, scanning neither stores historical information nor provides the level of detail needed to analyze and perform configuration management functions.

Building on advanced monitoring and data storage technology, ACM overcomes the shortcomings inherent in traditional approaches to change control and configuration management. ACM monitors the libraries that house program components and automatically captures related components during the translation process. It then stores this in formation using expert base/delta technology to help in determining changes to components from one translation to the next. Unlike source scanning, ACM works as an integral part of the translation procedure to automatically track program components over time.

The following shows how ACM works to automatically track program components over time.



As previously shown, when activated, ACM forms an "envelope" around the translation process (for example, compiler or linkage editor). When the program or module is executed under an CA Endevor SCM processor, ACM tracks all requested change activity and application program-component relationships. Module interrelationships are automatically captured as part of the translation procedure. Optionally, this information can also be automatically stored in Root and Cross-reference data sets for "where-used" analysis.

ACM Facilities

The following facilities are supplied with CA Endevor SCM ACM:

- Component Monitor
- Component List
- Displaying Component Data and Batch Reporting
- Software Control Language (SCL) Enhancements
- ACM Query Facility

Component Monitor

The ACM Component Monitor is invoked by adding a special "MONITOR=COMPONENTS" keyword to DD statements in an CA Endevor SCM processor. Monitoring is performed on a data set basis. Once initiated, components used from the specified data sets are automatically tracked by ACM's Component Monitor.

The processor keyword MONITOR cannot be specified on DD statements that refer to HFS path and file names.

Component List

The CA Endevor SCM ACM Component List provides a detailed "snapshot" of all the components from monitored data sets at the time of program translation. The Component List provides five types of information: element, processor, symbol, input, and output.

- The element information describes the program being generated.
- The processor information describes the CA Endevor SCM processor used to generate or move the element.
- The symbol information identifies the user-defined symbolics used in the CA Endevor SCM processor, and their values.
- The input components identify (by step, DDname, dsname, and volume) the component items referenced and any footprints that existed in a monitored data set.
- The output components identify (by step, DDname, dsname, and volume) the members that were created during processor execution.

The first time ACM monitoring is enabled for an element, a "base" component list is stored. This component list contains all of the element, processor, input, and output information mentioned above. Subsequent generations produce new component lists which ACM internally compares to produce "delta levels." (A new delta level is stored only when one or more items change in the component list.) This base/delta architecture makes it possible to compare component list changes from compile to compile. With this information, you can quickly determine what has changed since the last compile, as well as greatly reduce debug time.

Component List Level Numbers

The level numbers assigned to a component list (component levels) are independent of the level numbers assigned to an element (element levels). Since an element may be recompiled many times due to copy or macro changes, there may be more component levels than element levels. In addition, monitoring may be initiated at any time and at any element level. For example, an element at level 77 will have a component list at level 0 when it is first enabled for monitoring by ACM.

ACM remembers up to 97 generations of a component list by default. When level 97 is reached, component levels 50-96 are re-numbered as 0-49, and the previous (old) levels 0-49 are deleted. In this way, the component list functions like a circular storage vehicle. You are guaranteed that the last 50 "translations" are remembered by ACM if you use the default. You can set the number of generations remembered by ACM to a number lower than 97.

Component Data and Batch Reporting

CA Endevor SCM's element display facility enables you to view configuration information online. Using this facility, you can view the current level of a program and its related components, the component changes from one compile to the next, and the complete change history of a program and its components over time. All Foreground Query options (Component Summary, Component Browse, Component Changes, Component History) are also available in batch through the SCL PRINT command.

Software Control Language (SCL) Enhancements

Expanding on CA Endevor SCM's batch processing language, SCL, ACM enables you to "implode" and "explode" information in a component list through a special LIST command. This command produces valid syntax which can further be used to perform CA Endevor SCM actions such as GENERATE, MOVE, and ADD. The LIST command performs the function of cross-referencing a low-level component with all of its higher level "owners."

ACM Query Facility

The ACM Query Facility provides the capability to quickly perform "where-used" queries against ACM component data. This facility utilizes the information stored in the Root and Cross-reference data sets to provide this analysis. This data will be dynamically updated during processor execution or loaded at regular intervals by your system administrator.

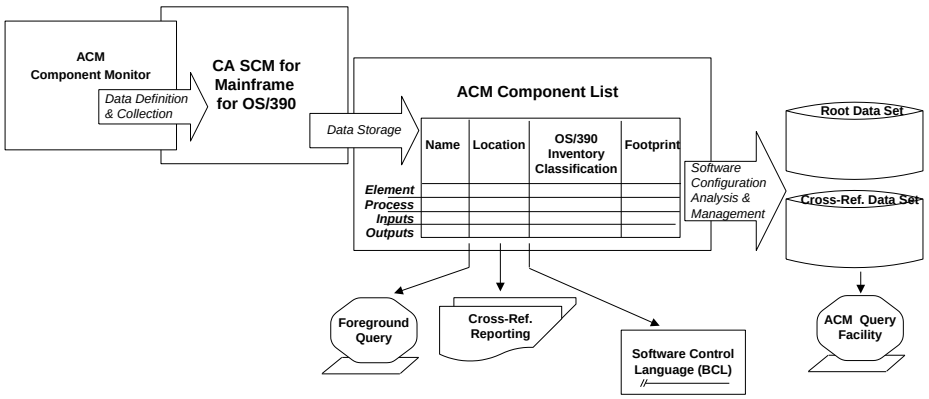
Note: For more information about the ACM Query Facility, see Getting Started with the ACM Query Facility (see page 61).

ACM Operational Areas

Operationally, ACM can be divided into three distinct areas:

- Data collection
- Data storage
- Software configuration analysis and management

The following illustrates how the different areas work together within ACM.



In these areas, you can see how data definition and collection starts in the ACM Component Monitor, data storage continues in the ACM Component List, software configuration analysis and management flows to the Root and Cross Reference Data Sets, and finally to the ACM Query Facility.

Data Collection

Data Collection begins when you invoke ACM through the MONITOR=COMPONENTS parameter in an CA Endevor SCM processor. With this parameter, you specify the libraries and data sets that are to be monitored automatically by ACM. Then, the Component Monitor tracks all change activity and program-component relationships for the data sets you earmarked for monitoring.

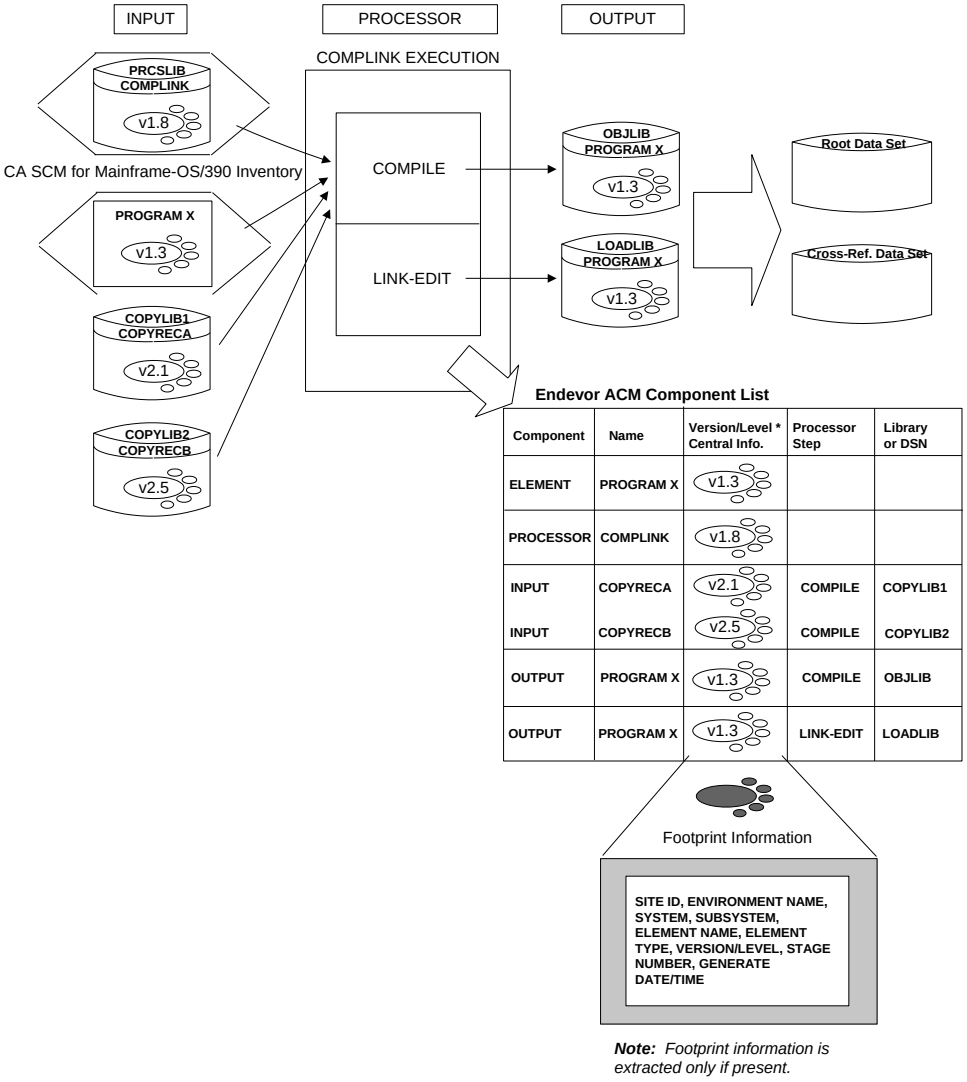
The processor keyword MONITOR cannot be specified on DD statements that refer to HFS path and file names.

Data Storage

ACM produces a component list during program translation (compilation), thus beginning the data storage process. The component list is an internal data structure which assembles all program-component information, along with an audit stamp or "footprint" for each component, at the time of each compile. Component list "levels" (differences in component lists from one translation to the next) are stored in CA Endevor SCM base/delta format. The component list provides a "snapshot" of a program at compilation—identifying an element’s components, where they originated, their version and level, and the output created as the result of the translation.

Example

The following illustrates how ACM stores and remembers the components used to create the outputs for a particular generate date and time, version, and level of a typical COBOL program—PROGRAM X—as transformed by the CA Endevor SCM processor COMPLINK.



Note: *Footprint information is extracted only if present.*

The ACM Component Monitor has captured all of the components of PROGRAM X, along with the CA Endevor SCM footprint information associated with those components. The resulting component list provides an integrated view of all configuration information relating to PROGRAM X at the time of the translation.

Software Configuration Analysis and Management

Through the component list and the information stored in the Root and Cross-reference data sets, it is possible to initiate CA Endevor SCM actions and make online inquiries. When viewed over time, this information provides a historical audit trail of component changes which serves as the foundation for all software configuration analysis and management activities, including:

- Analyzing system behavior
- Propagating a component change to all affected programs
- Validating a system for consistent use of components
- Recreating past program versions
- Moving related source components during promotion to production
- Providing "where-used" analysis

Name Masking

To help you more easily find information and process requests, you can use name masking. By substituting a name with the asterisk wildcard character (*), a character with the percent sign placeholder (%), or by using both together, it is much faster and easier to find information and process requests.

Note: For more information about name masking and valid uses for name masking, see the *User Guide.*

SCL Statement Syntax Convention

For information about syntax, how you code syntax, and sample syntax diagrams, see the *SCL Reference Guide*.

Chapter 2: Using ACM

This section contains the following topics:

How ACM Works

CA Endevor SCM ACM functions can be separated into the following four categories:

1. Enable ACM — Manual setup function performed by CA Endevor SCM administrators.
2. Activate the ACM Query Facility — Manual setup function performed by CA Endevor SCM administrators.
3. Monitor and Collecting Data — Performed automatically by the ACM Component Monitor.
4. Store and View Configuration Information — Capabilities provided by ACM through Component Lists and base/delta technology.
5. Analyze and Manage Configuration Information — Performed by users, with configuration information automatically monitored and stored by ACM.

Enable ACM and the ACM Query Facility

CA Endevor SCM ACM is shipped on the CA Endevor SCM installation tape and must be activated before you can use it. To enable ACM and the ACM Query facility, enter **Y** in the ASCM field in the CA Endevor SCM C1DEFLTS Table as shown below. Because ACM configuration data is collected during processor execution, you must also enable processors by coding the PROC=Y prior to running ACM.

```
C1DEFLTS TYPE=MAIN,                                          X
         ...
         ACMROOT=uprfx.uqual.root, ACM INDEX ROOT DATA SET NAME   X
         ACMXREF=uprfx.uqual.xref, ACM INDEX XREF DATA SET NAME   X
         ...
         ASCM=Y,                   ASCM CONTROL OPTION (Y/N)      X
         ...
         PROC=Y,
```

Note: To ensure that you have successfully enabled ACM, select option **1** (DISPLAY) on the Primary Options menu and press Enter. Select option **3** (SITE) and press Enter. On the Site Information panels, check the ASCM and PROC OPTIONS. If ACM iand processors are correctly enabled, a Y displays in these fields.

Activate the ACM Query Facility

To activate the ACM Query Facility you must perform the following steps:

1. Modify the C1DEFLTS table
2. Estimate Root and Cross-reference data sets space requirements
3. Define and initialize Root and Cross-reference data sets
4. Load Root and Cross-reference data sets

Modify the C1DEFLTS Table

The first step to activating the ACM Query facility is to modify the C1DEFLTS table.

To modify the C1DEFLTS table

1. Update the C1DEFLTS Table with the ACM Root and Cross-reference data sets.

```
C1DEFLTS TYPE=MAIN,
      ...

      ACMROOT=uprfx.uqual.root,      ACM INDEX ROOT DATA SET NAME
      ACMXREF=uprfx.uqual.xref,      ACM INDEX XREF DATA SET NAME
```

2. Assemble and link your C1DEFLTS.

   Note: To ensure that you have successfully enabled ACM, select option **1** (DISPLAY) on the Primary Options menu and press Enter. CA Endevor SCM displays the Display Options Menu. Check the ASCM field under OPTIONS. If ACM is enabled, you will see a Y in this field. The panel displays the Root and Cross-reference data set names.

   Note: If you are implementing CA Endevor SCM for the first time, you must estimate the number of elements to add to CA Endevor SCM, and then calculate space requirements.

Estimate Root and Cross-reference Data Sets Space Requirements

The ACM Query Facility's Root and Cross-reference data sets consist of the following:

- The Root data set contains the names of each CA Endevor SCM element and all related components.
- The Cross-reference data set contains a record for each component relationship.

To estimate the size of the Root and Cross-reference data sets

1. Execute the SYSTEM INVENTORY SUMMARY REPORT (CONRPT02) to get an overall count of CA Endevor SCM elements. Increase this value by 50% (that is, if the report determines that there are 4000 elements, the estimated total elements should be increased to 6000 to include related components which are not CA Endevor SCM elements).

   Note: You must have already assembled the C1DEFLTS Table with the ACM Root and Cross-reference data sets prior to the execution of this JCL.

2. A 3390 cylinder can hold approximately 6000 components. Using the value obtained above, allocate enough cylinders to accommodate your current inventory and future expansion. This value represents the primary and secondary allocation of the Root file.

3. Multiply the composite element total from the first step by 10 to estimate the total number of relationships to be added in the XREF file (i.e., 6000 * 10 = 60,000 estimated relationships).
4. A 3390 cylinder can hold approximately 120,000 relationships. Using the value obtained in the previous step, allocate enough cylinders to accommodate your current inventory and future expansion. This value represents the primary and secondary allocation of the XREF file.

Define and Initialize Root and Cross-Reference Data Sets

In this step, you define and initialize the Root and Cross-reference data sets.

Note: If you are an existing ACM user and want to populate Root and Cross-reference files, continue to the Load Root and Cross-reference data sets step.

To define and initialize the Root and Cross-reference data sets

1. Use member BC1JACMD, located in iprfx.iqual.CSIQJCL, to define (create) the Root and Cross-reference data sets.
2. Edit member BC1JACMD to add the space requirement information that you previously calculated.
3. Change the following variables:
   - iprfx — Highest-level qualifier used when assigning data set names for installation and execution data sets.
   - iqual — Second-level qualifier used when assigning data set names for installation and execution data sets.
   - vvolser — Volume serial number of the disk on which the MCF and package data sets will be allocated.
4. Follow the instructions in BC1JACMD to further tailor the JCL. After you have tailored the JCL, make sure you have a valid JOBCARD, and submit the JOB to create and initialize the Root and Cross-reference data sets.

Load Root and Cross-Reference Data Sets

In this step, you load Root and Cross-reference data sets.

Note: You can also use this step to reload Root and Cross-reference data sets.

1. Use member BC1JACML, located in iprfx.iqual.CSIQJCL, to extract existing ACM component information from CA Endevor SCM and load the ACM Root and Cross-reference data sets. You must modify the SCL control statements for Step 1 to specify all the environment names in your environment map.

   Note: Because this job can be time consuming, you may elect to run it in multiple parallel jobs, for each job selecting a different environment, system, subsystem, and type or stage (or both type and stage).

2. Edit member BC1JACML to make the above modifications. Follow the instructions in BC1JACML to further tailor the JCL and SCL. After you have tailored the JCL and SCL, make sure you have a valid JOBCARD, and submit the job to load the Root and Cross-reference data sets.

3. Change the following variables:

   iprfx

   Highest-level qualifier used when assigning data set names for installation and execution data sets.

   iqual

   Second-level qualifier used when assigning data set names for installation and execution data sets.

   tdisk

   Unit label for temporary disk data sets.

Note: You can use JCL member BC1JACMQ, located in iprfx.iqual.CSIQJCL, to run a quick validation that the ACM extended query data sets have been loaded correctly. For more information, see the chapter Using the ACM Query Utility (see page 61).

Maintenance of Root and Cross-Reference Data Sets

The performance of CA Endevor SCM depends on well maintained ACM Query files. The batch job BC1JACMO is provided to maintain the ACM Query files. The JCL for BC1JACMO can be found in *iprfx.iqual*.CSIQJCL. The job BC1JACMO invokes program BC1PACMO to reorganize the root and cross-reference data sets. Reorganization removes records that are no longer used and merges newly inserted records into the sorted areas of the files.

We recommend the following regular maintenance:

- **Reorganization**—Run the BC1PACMO utility regularly to reorganize the root and cross-reference data sets. For more information, see How to Reorganize Root and Cross-Reference Data Sets (see page 26).
- **Backup**—Edit the BC1PACMO utility so that it creates backup copies of the data sets each time the data sets are reorganized. For more information, see How to Create Backup Copies of the ACM Query Data Sets (see page 27).

The following additional functions can be used, when needed:

- **Synchronization**—Use the Validate parameter in BC1PACMO to synchronize ACM Query data with inventory data to remove orphan ACM Query entires. For more information, see How to Synchronize ACM Query Data with Inventory Data (see page 28).
- **Replacement**—Reload the ACM Query data to refresh existing ACM Query references. Program BC1PACMO is used by job BC1JACML to load new ACM Query data or to refresh existing ACM Query references with new data from the ACMCOMP DD. For more information, see Load Root and Cross-Reference Data Sets (see page 25).

Although the BC1JACMO job can run concurrently with other CA Endevor SCM jobs, we recommend that you run it outside of peak hours, to avoid waits in the jobs that update the ACM Query data sets. BC1PACMO always reserves the ACM Query ROOT and XREF data sets for exclusive use, thus blocking any other CA Endevor SCM access to ACM Query data for the duration of the job step. Typically, the execution of BC1PACMO takes less than one minute. However, the Validate parameter greatly slows down BC1PACMO execution. We strongly recommend that you use the Validate parameter only during hours of low CA Endevor SCM activity, because the exclusive access of BC1PACMO will block most CA Endevor SCM activities for the duration of the run.

How to Reorganize Root and Cross-Reference Data Sets

We recommend that you run the batch job BC1JACMO at least once a day or set it to run after a threshold of 5,000 updates. Infrequent reorganization of the ACM Query files will lead to bottlenecks and throughput problems.

To automate the batch job to run after a minimum number of 5,000 file inserts have been registered, activate the following setting in the Optional Features Table (ENCOPTBL):

```
AUTO_ACMQ_FILE_REORG=(on,5000)
```

If you would rather receive a console message when the threshold is met, instead of automatically invoking the batch job, then you must *also* activate the following ENCOPTBL option:

```
ACMQ_REORG_BY_WTO=ON
```

For more information, see the comments in the ENCOPTBL table file.

How to Create Backup Copies of the ACM Query Data Sets

Although ACMQ data can be rebuilt by running BC1JACML, we recommend that you make daily copies of the data sets. You can use BC1PACMO to make copies of the data sets that are synchronized so that the XREF file content matches ROOT file content.

Note: We recommend that you create backup copies of ACM files using BC1PACMO. Although IDCAMS REPRO and other programs can make copies of the files, the copies created by those programs may not be synchronized. Therefore, the copies cannot be used to properly back up the ACMQ data sets.

The BC1PACMO program can back up ROOT and XREF data as well as reorganize the data set contents. To create backups, add the following DD statements to BCIPACMO:

```
//ROOTCOPY DD DISP=(NEW,CATLG),DSN=your.root.copy(+1),
// LRECL=4096,RECFM=FB,DSORG=PS
//XREFCOPY DD DISP=(NEW,CATLG),DSN=your.xref.copy(+1),
// LRECL=4096,RECFM=FB,DSORG=PS
```

When these two DD statements are present, the elapsed time of BC1PACMO will increase by a few seconds.

How to Synchronize ACM Query Data with Inventory Data

If you suspect that there are synchronization problems between the inventory data of the component lists, MCF, and ACM Query data sets, you can run the BC1PACMO job to remove orphan ACM Query entries. To synchronize your ACM Query data, run BC1PACMO with the parameter Validate as follows:

```
EXEC PGM=NDVRC1,PARM='BC1PACMOVALIDATE'
```

VALIDATE

Verifies that each parent in the ACMQ ROOT file is defined in your inventory (MCF/ELMCATL). If a parent record is not defined in the inventory, the entry is deleted from ACMQ and the program returns RC=4.

Note: The Validate parameter is ignored if the ACMCOMP DD statement is present.

Note: The Validate parameter will considerably slow down the BC1PACMO execution. For more information, see Maintenance of Root and Cross-Reference Data Sets (see page 25).

How to Monitor and Collect Data

Once CA Endevor SCM ACM is enabled, you are ready to begin automatic monitoring of components during CA Endevor SCM processor execution. Generate and move processors can be monitored.

The Component Monitor

ACM has been built to be compatible with source translators (COBOL compilers, assemblers, PL/1 compilers, and linkage editors) without modification. These translators resolve element-to-component relationships by reading in components—called "input components"—from specific libraries. Utilities such as the linkage editor, CONLIST, create and write members—called "output components"—to various libraries.

The ACM Component Monitor automatically and transparently captures relationships by monitoring selected library data sets. You select the libraries you want monitored by ACM through a processor.

The information captured by the Component Monitor includes the step name, DD statement, volume, data set name, PDS or PDS/E directory and footprint information (if it exists).

Optionally, ACM provides the CONSCAN processor utility to add additional component information.

Note: For more information about using this utility, see the *Extended Processors Guide*.

Program Object Support

The IBM program product DFSMS/MVS 1.1 (now known as DFSMS z/OS) introduced a new executable storage format called program objects, which are stored in PDS/E data sets with undefined record format (otherwise known as 'PDS/E load libraries'). Program objects are functionally similar to load modules, but relieve many of the restrictions of conventional load modules. They are created by a utility which replaces the linkage editor, called the binder.

With DFSMS/MVS 1.1 installed, processor steps which execute IEWL—the old linkage editor—actually execute the binder. Note that component monitoring during execution of the binder and any other utilities which manipulate program objects also requires the use of the IEWBIND programming interface or the Directory Entry Services (DESERV) programming interfaces in DFSMS/MVS 1.3.

Activate the ACM Component Monitor

When activated, CA Endevor SCM ACM's Component Monitor automatically tracks and captures program-component relationships as specified in the processor definition. To activate the Component Monitor, you simply add the keyword **MONITOR=COMPONENTS** onto the CA Endevor SCM processor DD statements that you want to monitor. The processor keyword MONITOR cannot be specified on DD statements that refer to HFS path and file names.

The MONITOR=COMPONENTS keyword can be coded on any DD statement within an CA Endevor SCM generate or move processor. Usually, you would not code MONITOR=COMPONENTS on STEPLIBs, SYSUTn, or temporary data sets. You cannot code MONITOR=COMPONENTS on DDs that specify a path or HFS file.

How to Monitor Components in Dynamically Allocated Data Sets

There are cases when a user program in a processor dynamically allocates datasets. Typical examples are IKJEFT01 and EDCPRLK. When these programs read from or write to these data sets, you do not have control over the following parameters:

- MONITOR=COMPONENTS
- BACKOUT=N
- FOOTPRINT=CREATE

As a result, CA Endevor SCM chooses the default values for parameters, which means that by default:

- No components will be monitored
- Backouts will be written
- No footprints will be written

In order to modify these parameters, you can specify a DD name starting with 'EN$DYN' to define one or more data set names (in a concatenation) with their monitor/backout/footprint specifications.

Note: CA Endevor SCM does not support FOOTPRNT=CREATE on a concatenated data set. In case this is required, more than one EN$DYNXX statement will be required.

In the following example, no backout records will be written to file &SYSLIB1, even if IKJEFT01 allocates it and writes members to &SYSLIB1. Similarly, all members created in SYSLIB2 will be footprinted.

```
//STEP1EXEC PGM=IKJEFT01
//EN$DYN00 DD DISP=SHR,DSN=&SYSLIB1.,BACKOUT=NO
//EN$DYNM  DD DISP=SHR,DSN=&SYSLIB2.,FOOTPRNT=CREATE
```

In the next example, components will be monitored from all files in the SYSLIB concatenation, even if EDCPRLK reallocates each file separately under another DD name.

```
//PRELINK EXEC PGM=EDCPRLK
//SYSLIB   DD DISP=SHR,DSN=&SYSLIB1.,MONITOR=COMPONENTS
// DD DISP=SHR,DSN=&SYSLIB2.,MONITOR=COMPONENTS ETC.
//EN$DYNDS DD DISP=SHR,DSN=&SYSLIB1.,MONITOR=COMPONENTS
// DD DISP=SHR,DSN=&SYSLIB2.,MONITOR=COMPONENTS ETC.
```

In the previous example, the MONITOR=COMPONENTS statements can be removed from the SYSLIB concatenation, as their presence on the EN$DYNDS DD statement will cause all their components to be monitored regardless of the DD name under which the file is eventually (re-) allocated.

How to Monitor Input Components

Typical input components would be INCLUDE object modules, load modules, program objects, copybooks, and CALLed modules. COBOL compilers, for example, read in copybooks from a SYSLIB DD statement. Thus, for each data set in the SYSLIB concatenation, you would code MONITOR=COMPONENTS, as shown in the following example:

```
//SYSLIB DD DSN=BST.C1DEMO.COPYLIB1,DISP=SHR,MONITOR=COMPONENTS
//        DD DSN=BST.C1DEMO.COPYLIB2,DISP=SHR,MONITOR=COMPONENTS
```

How to Monitor Output Components

Typical outputs would be load modules, program objects, object decks, and listings. The linkage editor, for example, writes load modules to a SYSLMOD DD statement. For each output data set you want to monitor, code MONITOR=COMPONENTS, as shown in the following example:

```
//SYSLMOD DD DSN=BST.C1DEMO.LOADLIB1,DISP=SHR,MONITOR=COMPONENTS
```

How to Monitor Components in a Generate Processor

Any data set defined to a program within a processor can be monitored by the Component Monitor. The following programs are typically used in a generate processor. This sampling represents DD statements containing data sets that would normally warrant monitoring.

Note: The Component Monitor does not support programs that use EXCP (such as IEBCOPY) to perform I/O operations.

You are not restricted to the following programs, identified according to program, the Ddname and descriptions.

CONWRITE

ddname—CONWRITE writes to the first DDname after the EXEC statement. To monitor components expanded by CONWRITE (namely INCLUDEs), code MONITOR=COMPONENTS on the DDname. If MONITOR=COMPONENT is coded, make sure the parameter is set at PARM=EXPINCL(Y). ACM will then monitor for input components.

IGYCRCTL

SYSLIB—Input components (usually copylibs not COBOLlibs).

SYSLIN—Output components. Code only if writing to permanent data set.

SYSPRINT—Output components. Code only if writing to a permanent data set.

IEWL

SYSLIB—Input components (usually utilities not system libraries).

SYSLMOD—Output components.

SYSLIN—Input components. Code only if reading from a permanent data set.

ddname—Any data set that may be referenced by linkage editor control cards.

CONLIST

C1LLIBO—Output components.

The MONITOR=COMPONENTS keyword can be coded on any DD statement within an CA Endevor SCM generate or move processor. As a general rule, you would not code MONITOR=COMPONENTS on STEPLIBs, SYSUTn, or temporary data sets.

Example (Generate Processor + MONITOR=COMPONENTS)

In the following example, we've decided to monitor several data sets in the CA Endevor SCM generate processor GCIINBL; therefore, we have added the keyword MONITOR=COMPONENTS after those data sets. An explanation describing why each data set was selected for monitoring is included below each example.

```
//*******************************************************************
//**                                                               **
//**    COBOL COMPILE AND LINK-EDIT PROCESSOR                      **
//**                                                               **
//**                                                               **
//**                                                               **
//*******************************************************************


//**
//GCIINBL PROC COBLIB='CEE.SCEERUN',
//              COBSTPLB='IGY.SIGYCOMP',
//              CSYSLIB1='uprfx.uqual1.COPYLIB',
//              CSYSLIB2='uprfx.uqual2.COPYLIB',
//              EXPINC=N,
//              LISTLIB='uprfx.uqual1.LISTING',
//              LOADLIB='uprfx.uqual1.LOADLIB',
//              LSYSLIB1='uprfx.uqual1.LOADLIB',
//              LSYSLIB2='uprfx.uqual2.LOADLIB',
//              MEMBER=&C1ELEMENT.,
//              PARMCOB='LIB,NOSEQ,OBJECT,PMAP,DMAP',
//              PARMLNK='LIST,MAP,SIZE(9999K)',
//              SYSOUT=A,
//              WRKUNIT=tdisk
//**


//*********************************************************************
//*   ALLOCATE TEMPORARY LISTING DATA SETS                            *
//*********************************************************************
//*
//INIT      EXEC PGM=BC1PDSIN,MAXRC=0
//C1INIT01   DD DSN=&&COBLIST.,DISP=(,PASS,DELETE),
//              UNIT=&WRKUNIT.,SPACE=(CYL,(1,2),RLSE),
//              DCB=(RECFM=FBA,LRECL=121,BLKSIZE=6171,DSORG=PS)
//C1INIT02   DD DSN=&&LNKLIST.,DISP=(,PASS),
//              UNIT=&WRKUNIT.,SPACE=(CYL,(1,2),RLSE),
//              DCB=(RECFM=FBA,LRECL=121,BLKSIZE=3630,DSORG=PS)
//*
```

Unlike the COBOL compiler or Linkage Editor steps, the CONWRITE step may not contain DD statements for input. This is because CONWRITE can reference the INCLUDE libraries as specified in the TYPE definition and searches for members when expanding those INCLUDE statements. Therefore, in order to instruct CONWRITE to monitor inputs, we've specified "MONITOR=COMPONENTS" in the output statement (ELMOUT). This instructs the Component Monitor to monitor CONWRITE as it expands INCLUDEs.

```
//*********************************************************************
//* GET THE SOURCE FROM THE BASE/DELTA LIBRARIES                      *
//*********************************************************************
//*
//CONWRITE EXEC PGM=CONWRITE,PARM='EXPINCL(&EXPINC)'
//ELMOUT    DD DSN=&&ELMOUT.,DISP=(,PASS),
//          SPACE=(TRK,(1,1),RLSE),UNIT=&WRKUNIT.,
//          DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120),
//          MONITOR=COMPONENTS
//**
//*****************************************************************
//**    COMPILE THE ELEMENT                                       **
```

The data sets specified by &CSYSLIB1 and &CSYSLIB2 are being monitored because they are located where the COBOL compiler reads in copybooks. In this example, we are monitoring (per the PROC statement) uprfx.uqual1.COPYLIB (Stage 1) and uprfx.uqual2.COPYLIB (Stage 2). You do not want to monitor the &COBLIB DD because this calls extraneous COBOL subroutines.

```
//*****************************************************************
//**
//COMPILE EXEC PGM=IKFCBL00,COND=(0,NE),MAXRC=4,
// PARM='&PARMCOB.,BUF=512K,SIZE=1024K'
//STEPLIB DD DSN=&COBSTPLB.,DISP=SHR
//SYSLIB   DD DSN=&CSYSLIB1.,
//             MONITOR=COMPONENTS,DISP=SHR
//         DD DSN=&CSYSLIB2.,
//             MONITOR=COMPONENTS,DISP=SHR
//SYSIN    DD DSN=&&ELMOUT.,DISP=(OLD,DELETE)
//SYSLIN   DD DSN=&&SYSLIN.,DISP=(,PASS,DELETE),
//             UNIT=&WRKUNIT.,SPACE=(TRK,(3,5),RLSE),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120),
//             FOOTPRNT=CREATE
//SYSUT1   DD UNIT=&WRKUNIT.,SPACE=(CYL,(1,1))
//SYSUT2   DD UNIT=&WRKUNIT.,SPACE=(CYL,(1,1))
//SYSUT3   DD UNIT=&WRKUNIT.,SPACE=(CYL,(1,1))
//SYSUT4   DD UNIT=&WRKUNIT.,SPACE=(CYL,(1,1))
//SYSUT5   DD UNIT=&WRKUNIT.,SPACE=(CYL,(1,1))
//SYSPRINT DD DSN=&&COBLIST.,DISP=(OLD,PASS)
//**
//*****************************************************************
//**    LINK EDIT THE ELEMENT                                     **
//*****************************************************************
```

The data set indicated by &LOADLIB (uprfx.uqual1.LOADLIB in this example) is monitored because it is where the Linkage Editor or binder stores the output executable (load module or program object).

```
//**
//LKED     EXEC PGM=IEWL,COND=((0,NE,CONWRITE),(4,LT,COMPILE)),
//             PARM='PARMLNK',MAXRC=4
//SYSLIN   DD DSN=&&SYSLIN.,DISP=(OLD,DELETE)
//SYSLMOD DD DSN=&LOADLIB(&MEMBER).,
//             MONITOR=COMPONENTS,DISP=SHR
//SYSLIB   DD DSN=&LSYSLIB1.,
```

SYSLIB is where the Linkage Editor or binder resolves CALL or INCLUDE statements, based on inputs. In this example, we are monitoring the data sets indicated by &LSYSLIB1 (uprfx.uqual1.LOADLIB - Stage 1) and &LSYSLIB2 (uprfx.uqual2.LOADLIB - Stage 2).

```
//             MONITOR=COMPONENTS,DISP=SHR
//         DD DSN=&LSYSLIB2.,
//             MONITOR=COMPONENTS,DISP=SHR
//         DD DSN=&COBLIB.,
//SYSUT1   DD UNIT=&WRKUNIT.,SPACE=(CYL,(1,1))
//SYSPRINT DD DSN=&&LNKLIST.,DISP=(OLD,PASS)
//**
//*****************************************************************
//*     STORE THE LISTINGS IF: &LISTING=LISTING. LIBRARY NAME       *
//*****************************************************************
```

The data set indicated by &LISTLIB (uprfx.uqual1.LISTING, in this example) is being monitored because "C1LLIBO" is where CONLIST writes its output listings.

```
//*
//CONLIST EXEC PGM=CONLIST,MAXRC=0,PARM=STORE,
//           EXECIF=(&LISTLIB.,NE,NO)
//C1LLIBO DD DSN=&LISTLIB.,DISP=SHR,
//             MONITOR=COMPONENTS
//C1BANNER DD UNIT=&WRKUNIT.,SPACE=(TRK,(1,1)),
//             DCB=(RECFM=FBA,LRECL=121,BLKSIZE=6171)
//LIST01   DD DSN=&&COBLIST.,DISP=(OLD,DELETE)
//LIST02   DD DSN=&&LNKLIST.,DISP=(OLD,DELETE)
//*
```

Since MONITOR=COMPONENTS has been specified after certain data sets within GCOBNBL, ACM's Component Monitor is activated and automatically collects changes to the components each time the generate processor is executed. The change information for those data sets will be further stored as ACM Component Lists in base/delta format.

How to Monitor Components in a Move Processor

Move processors are used during a MOVE action to create the appropriate outputs at the target stage. This can be accomplished by copying the outputs (load modules and listings) from the source to the target stage.

Note: Do not re-create load modules at the target stage by coding a compile and link step in a move processor. This causes the load module footprint to become out of synchronization with Master Control File Generate information for the element.

Typically, the move processor is coded to perform two generic functions: 1) to perform a translation (compile and link-edit), and 2) to move a load module and any other entry stage members created by the entry stage generate processor.

- If you code a move processor to perform a translation (for example, compile, link-edit), code MONITOR=COMPONENTS as described in Monitoring Components in a Generate Processor.
- If you code a move processor in order to move load modules, listings, and so on from the entry stage to the next mapped stage, you might also want to move the current (version/level) of the Component List. You can do this using a special processor utility BC1PMVCL, which is provided with ACM. To execute this program, code a step in the CA Endevor SCM move processor that specifies the EXEC statement with a MAXRC=0, as illustrated below:

```
//MOVE EXEC PGM=BC1PMVCL,MAXRC=0
```

Now when the element in the entry stage is moved to the next mapped stage, its corresponding Component List (containing all the element-to-component relationships) will also be moved to the next mapped stage.

If you want a move processor to replace the OUTPUT components of the existing component list with the new output components that are the result of the move processor then MONITOR=COMPONENTS should be coded only on the OUTPUT files of the steps in the processor. When BC1PMVCL gets executed as the last step of the processor, it will automatically replace the existing OUTPUT components with the newly captured output components. If you do not want this to happen then simply do not code MONITOR=COMPONENTS in your move processor.

Important! Coding MONITOR=COMPONENTS in a move processor will lead to a new delta level in the component list in each of the following cases:

1. The processor contains MONITOR=COMPONENTS parameters and INPUT components are being monitored.
2. The move processor makes use of the CONRELE utility.
3. The utility program BC1PMVCL is not the last step in the move processor and components are being monitored in any step following BC1PMVCL.

Example (Sample Move Processor MLODNNL)

In the following sample move processor (MLODNNL), we've specified that when the load module is moved from Stage 1 to Stage 2, its associated Component List information is also moved and updated.

```
//*********************************************************************
//*                                                                   *
//* COPY LOAD MODULES FROM STAGE 1 TO STAGE 2 AND THEIR ASSOCIATED    *
//* COMPONENT LIST AND LISTINGS.                                      *
//*                                                                   *
//*********************************************************************


//*
//MLODNNL  PROC  LISTLIB='YES',
//               LISTLIB1='&PROJECT..&GROUP.&STG1..LISTLIB',
//               LISTLIB2='&PROJECT..&GROUP.&STG2..LISTLIB',
//               LOADLIB1='&PROJECT..&GROUP.&STG1..LOADLIB',
//               LOADLIB2='&PROJECT..&GROUP.&STG2..LOADLIB',
//               PROJECT='IPRFX.IQUAL',
//               GROUP='SMPL',
//               STG1='&C1SSTAGE.',     CURRENT STAGE
//               STG2='&C1STAGE.',      TO STAGE
//               MONITOR=COMPONENTS,
//               SYSOUT=*,
//               WRKUNIT=TDISK


//*********************************************************************
//*   ALLOCATE TEMPORARY LISTING DATASETS                             *
//*********************************************************************
//INIT      EXEC  PGM=BC1PDSIN
//C1INIT01 DD DSN=&&COPYLIST.,DISP=(,PASS,DELETE),
//            UNIT=&WRKUNIT.,SPACE=(CYL,(1,2),RLSE),
//            DCB=(RECFM=V,LRECL=121,BLKSIZE=125,DSORG=PS)
//*********************************************************************
//* COPY THE LOAD MODULE                                              *
//*********************************************************************
//BSTCOPY  EXEC PGM=BSTCOPY,MAXRC=04
//SYSPRINT  DD  DSN=&&COPYLIST.,DISP=(OLD,PASS)
//SYSUT3    DD  UNIT=&WRKUNIT.,SPACE=(TRK,(1,1))
//SYSUT4    DD  UNIT=&WRKUNIT.,SPACE=(TRK,(1,1))
//INDD      DD  DSN=&LOADLIB1.,DISP=SHR
//OUTDD     DD  DSN=&LOADLIB2.,DISP=SHR,MONITOR=&MONITOR.
//SYSIN     DD  *
  COPY O=OUTDD,I=INDD
  SELECT MEMBER=((&C1ELEMENT.,,R))
```

```
//*******************************************************************
//*     COPY & STORE THE LISTINGS IF:  &LISTING=LISTING LIBRARY     *
//*******************************************************************
//COPYLIST EXEC PGM=CONLIST,MAXRC=0,PARM=COPY,COND=EVEN,
//         EXECIF=(&LISTLIB.,EQ,YES)
//C1LLIBI  DD DSN=&LISTLIB1.,DISP=SHR
//C1LLIBO  DD DSN=&LISTLIB2.,DISP=SHR,MONITOR=&MONITOR.
//C1BANNER DD DSN=&&BANNER.,DISP=(,PASS,DELETE),
//            UNIT=&WRKUNIT.,SPACE=(TRK,(1,1)),
//            DCB=(RECFM=FBA,LRECL=121,BLKSIZE=6171,DSORG=PS)
//LIST01   DD DSN=&&COPYLIST.,DISP=(OLD,DELETE)


//*******************************************************************
//*     PRINT THE LISTINGS IF:  &LISTING=NO.                        *
//*******************************************************************
//PRNTLIST EXEC PGM=CONLIST,MAXRC=0,PARM=PRINT,COND=EVEN,
//         EXECIF=(&LISTLIB.,EQ,NO)
//C1BANNER DD UNIT=&WRKUNIT.,SPACE=(TRK,(1,1)),
//            DCB=(RECFM=FBA,LRECL=121,BLKSIZE=6171,DSORG=PS)
//C1PRINT  DD SYSOUT=&SYSOUT.,
//            DCB=(RECFM=FBA,LRECL=133,BLKSIZE=1330,DSORG=PS)
//LIST01   DD DSN=&&COPYLIST.,DISP=(OLD,DELETE)
//*******************************************************************
//* UPDATE THE COMPONENT LIST WITH THE NEW OUTPUT COMPONENTS          *
//* AND MOVE THE COMPONENT LIST TO THE NEXT STAGE
//*******************************************************************
//MOVECL   EXEC PGM=BC1PMVCL,COND=(0,NE)
//*
```

How to Monitor Components in a Delete Processor

Important! The delete processor runs as the result of a DELETE action. When the DELETE action is executed, both the element and its associated Component List are deleted. Therefore, you should not monitor a delete processor.

Changing processor groups invokes the delete processor.

How to Store Configuration Information

The components which are monitored and collected by the ACM Component Monitor are stored in the ACM component list. As the repository for configuration information, this component list provides a "snapshot" of a program—and the components which make up that program—each time a monitored CA Endevor SCM processor is executed.

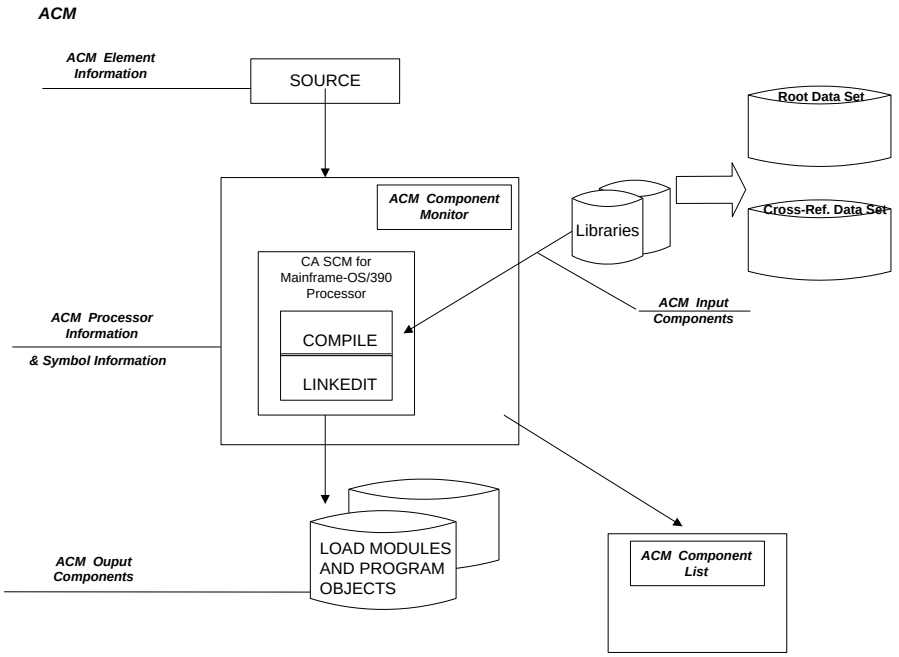
The Component List

The ACM component list comprises six internal component categories: element information, processor information, symbol information, input components, output components and related data.

Each of these component categories provides an essential piece of the in-depth information needed to analyze and manage software configurations.

- Element Information — The CA Endevor SCM source for which an CA Endevor SCM processor is being executed. It tells you what the originating CA Endevor SCM element source was when the component list was created.
- Processor Information — The CA Endevor SCM processor that is being executed for this particular element. It contains the MONITOR=COMPONENTS keyword for specific data sets, and tells you the specific CA Endevor SCM processor used when the component list was created.
- Symbol Information — The symbolics that have been defined by the user for this processor. It tells you the symbolic, the value that will be substituted for it when the processor is run, and where that value has been defined — either directly in the PROC statement of the processor or as overridden through the Processor Group Symbolics panel.
- Input Components — The members in the monitored data sets which are read by programs (such as the compiler or linkage editor) in the CA Endevor SCM processor when it collects components to build composite modules. It tells you the related "pieces" of the element and where they came from.
- Output Components — The monitored data sets which are written to by programs (such as the linkage editors or CA Endevor SCM utilities) when composite modules are created. It tells you the members created during processor execution.

The following diagram illustrates where the different pieces of component list information are derived.



As previously illustrated, each piece of Component List information is derived from a different source.

How ACM Stores Component Lists

Component information is recorded for all executed processor steps that read or update monitored files. The recorded component information is used to create a new component list, or if a component list exists, a new Component Delta level is created.

When a processor step abends, the monitored component information is logged in a new component Delta level. If no component information has been logged at the time of the abend, then a Delta level without component information is created.

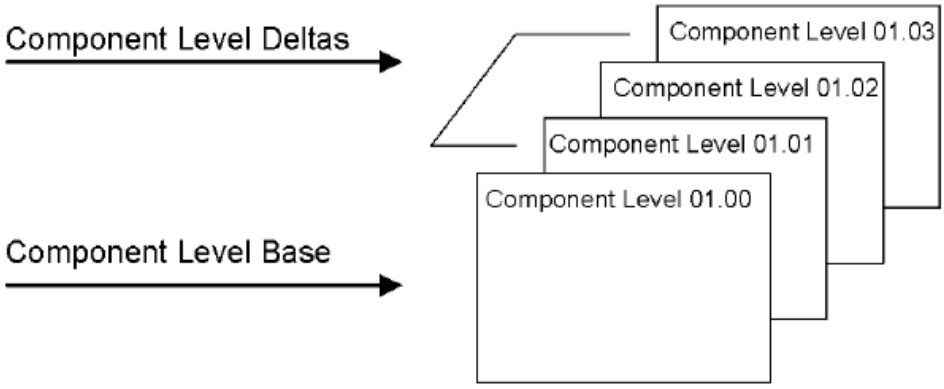
Both the component list Base and Delta members are stored in the Delta library as defined on the Type definition.

If the executed processor or processor group is changed to *NOPROC*, CA Endevor SCM deletes the existing component list and its history.

Base/Delta Technology

The first time an CA Endevor SCM ACM component list is produced for a processor, it becomes the base. Subsequent processor executions create new component lists, which are automatically compared against the base to reflect component changes. ACM gives each set of changes (delta) a new component level number. These base and deltas are stored in base and delta libraries as defined in the element type definition.

The following illustrates how Base/Delta Technology works in ACM.



As previously illustrated, ACM assigns each set of changes (delta) a new component level number. In this example, the numbers assigned are Component Level 01.00 through Component Level 01.03.

Component Levels

The component level is indicated in the COMPONENT LEVEL INFORMATION section of the component list. There may be a discrepancy between the component level and the element level because *there is absolutely no correlation between the component level and the element level*.

Component levels are created each time a component list is generated and has changed since the last generation. Component-level numbers are merely relative to the most recent generation; for example, if component level 01.53 is the most recent level, level 01.52 is the one immediately preceding it, and so forth.

The element level, on the other hand, is automatically increased each time the element is updated, regardless of whether or not the processor executed successfully.

The CONSCAN Processor Utility

The CONSCAN processor utility provides an additional mechanism for ACM to capture configuration information. A typical usage of this facility would be to capture the JCL— program relationships or JCL— data set name relationships. The captured relationship information can be added to ACM using the CONRELE processor utility. CONSCAN also creates control statements to be used by CONRELE to update component information.

Note: For more information about this utility, see the *Extended Processors Guide*.

How Component Levels are Renumbered

In order to eliminate massive storage considerations, there is a limit to the number of component levels which are stored in CA Endevor SCM. Starting at level 00, component levels are stored up until level 96 by default. At this point, when ACM is about to create another component list (and, hence, another component level), the first 50 component levels are consolidated into level 0101, and component levels 51 through 96 are renumbered as component levels 02 through 46. This process is repeated each time component level 96 is passed. What's important to remember is that, at any point in time, component level 00 is always considered to be the base, with subsequent component levels treated as deltas.

View Component List Information

You can view information about a Component List in ACM at any time.

To view component list information

1. Open the ISPF/PDF Primary Options panel.
2. Open the CA Endevor SCM Environment Selection panel.
3. Select an environment and press Enter.

   The CA Endevor SCM Primary Options panel appears.
4. Enter **1 (Display)** and press Enter.

   The Display Options menu appears.
5. Enter **1 (Element)** and press Enter.

   The Display Elements/Components List panel appears. From this panel, you can view different types of information about Components lists.

6. Identify the element and the information that you want to display on this panel and press Enter. The panel that displays next depends on the value in the DISPLAY SYS/SBS LIST field:

   - If you provided a wildcard in the SYSTEM and/or SUBSYSTEM fields and DISPLAY SYS/SBS LIST = **Y**, CA Endevor SCM displays a System and/or Subsystem Selection List. Make selections as necessary, pressing Enter after each selection.
   - If you provided a wildcard in the ELEMENT field, CA Endevor SCM displays an Element Selection List or Confirmation panel, as indicated in the following:

     Y

     - If Build using MAP=Y, CA Endevor SCM displays a selection list of all elements in all map environments that meet search criteria.
     - If Build using MAP=N (Default), CA Endevor SCM displays a selection list of all elements in the current environment that meet search criteria.

     N

     - If Build using MAP=Y, CA Endevor SCM displays a confirmation panel, indicating the number of elements selected (from all environments).
     - If Build using MAP=N, CA Endevor SCM displays a confirmation panel, indicating the number of elements selected (from the current environment).

   Important! Use DISPLAY LIST = **N** with caution, especially in conjunction with BUILD USING MAP = **Y**. When you press Enter at a Confirmation panel, you will have to view all the elements that have been selected.

7. Choose one of the following options.

   - If a Confirmation panel appears, press Enter to view the requested display for the number of elements indicated on the Confirmation panel.
   - If an Element Selection List appears, use it to display information for one or more elements. You can select any of the options discussed in the following section by typing the option to the left of the element name and pressing Enter.

Display Element/Component Lists Panel

The Display Element/Component Lists panel lets you display information about elements and components.

This window contains the following fields:

Option Field

Use an option code to specify the information you wish to display. Options **SX, BX, CX**, and **HX** are described in the following sections. Options include the following:

- **Blank**. A list appropriate to the information supplied on the panel, as described earlier.
- **S**. A Summary of Levels panel, showing a summary of change history for the element requested. From this panel you can select a specific level of the element for display, using option **B, C,** or **H**.
- **M**. An Element Master panel, showing Master Control File (MCF) information related to the element requested.
- **B**. An Element Browse panel, showing all statements in the current level of the element, and the level at which each statement was inserted.
- **C**. An Element Changes panel, showing all inserts and deletions made to the element as of the current level.
- **H**. An Element History panel, showing all statements in all levels of the element, from the base level through the current level. The display shows the level at which each insertion/deletion occurred.
- **SX**. A Summary of Levels panel, showing a summary of change history for the component list requested. From this panel you can select a specific level for display, using option **BX, CX,** or **HX**.
- **BX**. A Component Browse panel, showing the component information for the current level of the element, and the level at which component was inserted.
- **CX**. A Component Changes panel, showing all inserts and deletions made to the component information for the element as of the current level.
- **HX**. A Component History panel, showing the component information for all levels of the element, from the base level through the current level. The display shows the level at which each insertion/deletion occurred.

From CA Endevor SCM Fields

These fields contain information to describe the CA Endevor SCM location of the element. Options include the following:

- **Environment**. Name of the environment under which the element is defined (initially, the current environment). If the element is in a different environment, enter the environment's name in this field.
- **System**. Name of the system under which the element is defined.

- **Subsystem**. Name of the subsystem under which the element is defined.
- **Element**. Name of the element for which you want to display information.
- **Type**. Name of the element's type.
- **Stage**. ID of the stage in which the element resides. This must be one of the values shown to the right of the field (unless you are changing environments).

List Options Fields

These options allow you to specify further the information you want to display. Options include the following:

- **Display List**. Indicates whether you want to use list panels when requesting this action: **Y** (yes) or **N** (no). The default is **Y**.
- **Where CCID Eq**. Specifies a CCID that CA Endevor SCM uses to limit the selection list to only those elements whose last CCID matches the specified CCID. If omitted, CA Endevor SCM does not limit the selection list by CCID.
- **Where Proc Grp Eq**. Specifies a processor group that CA Endevor SCM uses to limit the selection list to only those elements to which the processor group has been assigned. If omitted, CA Endevor SCM does not limit the selection list by process or group.
- **Display SYS/SBS List**. Indicates whether you want to go directly to the Element Selection List from the Display Elements/Component Lists panel. Acceptable values are:
  - **Y—**Provide individual selection lists as required by your entries on this panel.
  - **N—**Default. Bypass system and subsystem selection lists.

Note: DISPLAY LIST = Y must be specified in order to see any of these lists.

- Build using map. Indicates whether you want CA Endevor SCM to search the map, starting at the FROM location, when building the Element Selection List. Acceptable values are:
  - **Y—**Search the map.
  - **N—**Default. Do not search the map.

Note: Avoid using BUILD USING MAP = Y in combination with DISPLAY LIST = N. You cannot cancel the build process once it has begun.

Display Summary Information

You can display a summary of component list levels by entering **SX** in the OPTION field of the Display Elements/Component Lists panel, or by entering **SX** to the left of an element name on the Element Selection List and pressing Enter. CA Endevor SCM displays the Summary of Levels panel.

Summary of Levels Panel

The Summary of Levels panel lets you display a summary of component list levels.

This window contains the following fields:

no title

Selection field. Enter BX, HX or CX.

Environment

The element's originating environment.

System

Name of the system under which the element is defined.

Subsystem

Name of the subsystem under which the element is defined.

Element

Name of the element for which the component list is being displayed.

Type

The element type for the element.

Stage

The stage for the element.

VVLL

Version/Level of the component.

User

The user whose job created the component list.

Date

The date the component list was created.

Time

The time the component list was created.

Stmts

Total number of statements in the component list.

Inserts

Number of lines inserted at this level of the component list.

Deletes

Number of lines deleted at this level of the component list.

Synch

Indicates whether this level was created through synchronization (S) or level consolidation (C).

Display the Current Version and Level of an Element (BX)

To view the current version/level of the element, FINAPP01, and its related components, enter **BX** in the OPTION field of the Display Elements/Component List panel, or to the left of an element name on the Element Selection List and press Enter. CA Endevor SCM displays the current version of the element and its related components.

Display Component Changes (CX)

To view component changes only, enter **CX** in the OPTION field of the Display Elements/Components Lists panel, or to the left of an element name on the Element Selection List and press Enter. CA Endevor SCM displays the current version of the element and its related component changes.

Display Change History (HX)

To view the change history, enter **HX** in the OPTION field of the Display Elements/Components Lists panel, or to the left of an element name on the Element Selection List and press Enter. CA Endevor SCM displays the change history for the specified element/component.

Component List Panel

The Component List panel lets you display information about components.

This panel contains the following fields:

Banner

The Banner at the top of the component list describes the element to which the component list belongs. The Banner contains the following fields:

no title

Blank. The following message appears if the component list was copied from Stage 1 to Stage 2 without monitoring output components in the move processor: COMPONENT AND LIST COPIED FROM STAGE 1.

Component <Display>

<Display> is a variable that indicates the option specified (Component Browse, Component Changes, or Component History).

Date

The date the component list was created.

Time

The time the component list was created.

Environment

The element's originating environment.

System

Name of the system under which the element is defined.

Subsystem

Name of the subsystem under which the element is defined.

Element

Name of the element for which the component list is being displayed.

Type

The element type for the element.

Stage

The stage name for the element.

Component Level Information

Component information is recorded for all executed processor steps that read or update monitored files. For more information, see How ACM Stores Component Lists (see page 40). The Component Level Information section contains the following fields:

VVLL

Version/Level of the component.

SYNC

Indicates whether this level is a synchronize level; that is, built by CA Endevor SCM as a result of a MOVE or TRANSFER action. This field does not appear on a component list.

User

The user who ran a processor which created the component list.

Date

The date the component list was created.

Time

The time the component list was created.

Stmts

Total number of statements in the component list.

CCID

The CCID specified with the action which created this level of the component list.

Comment

The comment specified with the action which created this level of the component list.

Element Information

This section of the component list provides element information, and contains the following fields:

Level (no title) columns 1-7

Display-only. Indicates the level at which this particular line was inserted into or inserted into/deleted from the component list, as follows:

- **+LLLL** (in Col. 2-4) indicates that a line was inserted at a particular level; for example, +0102 signifies that the line was inserted at level 0102.
- **+LLLL-LLLL** (in Col. 2-7) indicates that a line was inserted at one level, then deleted at another level. For example, +0102-0103 signifies that a particular line was inserted at level 0102, then deleted at level 0103.
- **%** (in Col. 1) depends upon the type of display you request. If you request Component Browse, a percent sign (%) indicates that the line was inserted (+LLLL) or deleted (-LLLL) as of the level displayed.

If you request Component History, "%" appears next to every change that has occurred since the base level. For example, if the base level is 0100 and the current level is 0102, any changes occurring in levels 0101 and 0102 will be flagged with the percent sign.

VVLL

Version/Level of the element when the component list was created.

Date

Current level date of the element when the component list was created.

Time

Current level time of the element when the component list was created.

System

Name of the system under which the element is defined.

Subsystem

Name of the subsystem under which the element is defined.

Element

Name of the element for the component list.

Type

The element type for the element.

Group

The name of the processor group for the element.

Stage

The stage for the element.

Site

Location where CA Endevor SCM is installed.

Environment

Current environment.

Processor

The name of the processor used to process the element.

Processor Information

This section describes the CA Endevor SCM processor that created the component list. It contains the following fields from the processor's footprint:

Level (no title) columns 1-7

Display-only. Indicates the level at which this particular line was inserted into or inserted into/deleted from the component list, as follows;

- **+LLLL** (in Col. 2-4) indicates that a line was inserted at a particular level; for example, +0102 signifies that the line was inserted at level 0102.
- **+LLLL-LLLL** (in Col. 2-7) indicates that a line was inserted at one level, then deleted at another level. For example, +0102-0103 signifies that a particular line was inserted at level 0102, then deleted at level 0103.
- **%** (in Col. 1) depends upon the type of display you request. If you request Component Browse, a percent sign (%) indicates that the line was inserted (+LLLL) or deleted (-LLLL) as of the level displayed.

If you request Component History, "%" appears next to every change that has occurred since the base level. For example, if the base level is 0100 and the current level is 0102, any changes occurring in levels 0101 and 0102 will be flagged with the percent sign.

VVLL

Version/Level of the processor.

Date

The date the processor was generated.

Time

The time the processor was generated.

System

Name of the system under which the processor is defined.

Subsystem

Name of the subsystem under which the processor is defined.

Element

Name of the processor.

Type

The processor type.

Group

This field is not applicable to this particular section of the component list.

Stage

The stage for the processor.

Site

The location where CA Endevor SCM is installed.

Environment

Current environment.

Processor

This field is not applicable to this particular section of the component list.

Symbol Information

This section lists the user-defined symbols that appear in the PROC statements within the indicated processor.

Level (no title) columns 1-7

Display-only. Indicates the level at which this particular line was inserted into or inserted into/deleted from the component list, as follows:

- **+LLLL** (in Col. 2-4) indicates that a line was inserted at a particular level; for example, +0102 signifies that the line was inserted at level 0102.
- **+LLLL-LLLL** (in Col. 2-7) indicates that a line was inserted at one level, then deleted at another level. For example, +0102-0103 signifies that a particular line was inserted at level 0102, then deleted at level 0103.
- **%** (in Col. 1) depends upon the type of display you request. If you request Component Browse, a percent sign (%) indicates that the line was inserted (+LLLL) or deleted (-LLLL) as of the level displayed.

If you request Component History, "%" appears next to every change that has occurred since the base level. For example, if the base level is 0100 and the current level is 0102, any changes occurring in levels 0101 and 0102 will be flagged with the percent sign.

Defined

Indicates where the symbolic has been defined:

- **PROCESSOR—**The value of the symbolic is defined within the PROC statement in the processor.
- **PROC GRP—**The original value of the symbolic has been overridden using the Processor Group Symbolics panel.

Note: For more information about this option, see the *Extended Processors Guide*.

Symbol

The symbolic as it appears in the PROC statement.

Value

The value that is substituted for the symbolic when the processor is run.

Input Components

This section lists the input components that were used during processor execution, and it contains the following fields:

STEP:

STEP name of the processor.

DD=

DDname from the processor.

VOL=

Volume on which the data set resides.

DSN=

Library from which the input component was read.

Level (no title) columns 1-7

Display-only. Indicates the level at which this particular line was inserted into or inserted into/deleted from the component list, as follows:

- **+LLLL** (in Col. 2-4) indicates that a line was inserted at a particular level; for example, +0102 signifies that the line was inserted at level 0102.
- **+LLLL-LLLL** (in Col. 2-7) indicates that a line was inserted at one level, then deleted at another level. For example, +0102-0103 signifies that a particular line was inserted at level 0102, then deleted at level 0103.
- **%** (in Col. 1) depends upon the type of display you request. If you request Component Browse, a percent sign (%) indicates that the line was inserted (+LLLL) or deleted (-LLLL) as of the of the level displayed.

If you request Component History, "%" appears next to every change that has occurred since the base level. For example, if the base level is 0100 and the current level is 0102, any changes occurring in levels 0101 and 0102 will be flagged with the percent sign.

Member

Name of the input component read in by the CA Endevor SCM processor during execution.

The following fields are from the footprint (if applicable):

VVLL

Version/Level of the input component.

Date

The date the input component was generated.

Time

The time the input component was generated.

System

Name of the system under which the input component is defined.

Subsystem

Name of the subsystem under which the input component is defined.

Element

Name of the input component.

Type

Name of the input component type.

Stage

The stage for the input component.

Site

The location where CA Endevor SCM is installed.

Environment

The current environment.

LD

Indicates whether a footprint was created by the Load Utility.

Output Components

This section lists the members that were created during processor execution, and contains the following fields:

STEP:

STEP name of the processor.

DD=

DDname from the processor.

VOL=

Volume on which the data set resides.

DSN=

Library that contains the output component.

Level (no title) columns 1-7

Display-only. Indicates the level at which this particular line was inserted into or inserted into/deleted from the component list, as follows:

- **+LLLL** (in Col. 2-4) indicates that a line was inserted at a particular level; for example, +0102 signifies that the line was inserted at level 0102.
- **+LLLL-LLLL** (in Col. 2-7) indicates that a line was inserted at one level, then deleted at another level. For example, +0102-0103 signifies that a particular line was inserted at level 0102, then deleted at level 0103.
- **%** (in Col. 1) depends upon the type of display you request. If you request Component Browse, a percent sign (%) indicates that the line was inserted (+LLLL) or deleted (-LLLL) as of the level displayed.

If you request Component History, "%" appears next to every change that has occurred since the base level. For example, if the base level is 0100 and the current level is 0102, any changes occurring in levels 0101 and 0102 will be flagged with the percent sign.

Member

Name of the output component created during processor execution.

The following fields are from the footprint (if applicable):

VVLL

Version/Level of the output component.

Date

The date the output component was generated.

Time

The time the output component was generated.

System

Name of the system under which the output component is defined.

Subsystem

Name of the subsystem under which the output component is defined.

Element

Name of the output component.

Type

Name of the output component type.

Stage

The stage for the output component.

Site

The location where CA Endevor SCM is installed.

Environment

The current environment.

LD

Indicates whether a footprint was specified by the Load Utility.

Related Input Components

This section lists the elements created during processor execution and contains the following fields:

VVLL

Version/Level of the related input component.

Date

The date the related input component was generated.

Time

The time the related input component was generated.

System

Name of the system under which the related input component is defined.

Subsystem

Name of the subsystem under which the related input component is defined.

Element

Name of the related input component.

Type

Name of the related input component type.

Stage

The stage for the related input component.

Environment

The current environment.

The following fields and descriptions are for related members. You can generate footprint information for each of the following fields:

DSN=

Library from which the related input component was read.

Member

The name of the related input component read in by the CA Endevor SCM processor during execution.

VVLL

Version/Level of the related input component.

Date

The date the related input component was generated.

Time

The time the related input component was generated.

System

Name of the system under which the related input component is defined.

Subsystem

Name of the subsystem under which the related input component is defined.

Element

Name of the related input component.

Type

Name of the related input component type.

Stage

The stage for the related input component.

Environment

The current environment.

Related Output Components

This section lists the elements created during processor execution and contains the following fields:

VVLL

Version/Level of the related output component.

Date

The date the related output component was generated.

Time

The time the related output component was generated.

System

Name of the system under which the related output component is defined.

Subsystem

Name of the subsystem under which the related output component is defined.

Element

Name of the related output component.

Type

Name of the related output component type.

Stage

The stage for the related output component.

Environment

The current environment.

The following fields and descriptions are for related members. You can generate footprint information for each of the following fields:

DSN=

Library from which the related output component was read.

Member

The name of the related output component read in by the CA Endevor SCM processor during execution.

VVLL

Version/Level of the related output component.

Date

The date the related output component was generated.

Time

The time the related output component was generated.

System

Name of the system under which the related output component is defined.

Subsystem

Name of the subsystem under which the related output component is defined.

Element

Name of the related output component.

Type

Name of the related output component type.

Stage

The stage for the related output component.

Environment

The current environment.

Related Objects

Related objects are associated with the following field:

Path ID

70-character object path identifier.

Related Comments

Related comments are associated with the following field:

Comment

70-character freeform description.

Input and Output Component Footprints

When footprints appear within an input or output component on a component list, it signifies that the source that created that component was controlled (and thus footprinted) by CA Endevor SCM.

CA Endevor SCM footprints contain the following information: site ID, environment, system, subsystem, element, type, stage, version/level, and generate date/time.

When elements are controlled by CA Endevor SCM, their footprints are carried along into the component list during program execution. If, for example, you reference a copy record in a PDS that is under the control of CA Endevor SCM, it will contain a footprint that will show up on the component list.

Load Modules controlled by CA Endevor SCM can contain multiple footprints. Consequently, when an element is a load module (with multiple footprints), ACM does not capture all of the footprints for display on the component list.

How Element Action Processing Works

CA Endevor SCM element action processing and Quick-Edit invoke the ACMQ query facility to report on existing dependencies on the element modified by the action or Quick-Edit. After CA Endevor SCM element action processing is completed, CA Endevor SCM invokes ACMQ to determine if the element is referenced by other elements. It then issues a message that appears in the execution message log, indicating the result of the query.

Similarly, Quick-Edit invokes ACMQ to determine element dependencies when it displays the initial text edit panel. It communicates the results of the query via ISPF ‘note’ lines. (The additional messages appear after the ‘fetched from location’ information note lines, if the element is being fetched to the edit inventory location.)

Chapter 3: Using the ACM Query Facility

This section contains the following topics:

Getting Started with the ACM Query Facility

In addition to the functionality discussed in the previous chapter, you can also view "where-used" information against the ACM component data by utilizing the ACM Query Facility. To activate this facility, see Using CA Endevor SCM ACM (see page 21).

The ACMQ command is used to perform online queries or SCL generation against the root and cross-reference data sets.

Start ACMQ

You can start the ACMQ facility at any time to manage, through advanced automated technology, the interrelationships between software components.

To start ACMQ

1. Execute a CLIST that allocates the library that contains the C1DEFLTS table and the CA Endevor SCM product libraries.
2. Do *one* of the following:
   - Enter **AC** on the command line of any CA Endevor SCM panel.
   - Enter **TSO ACMQ** on any ISPF panel.
   - Execute ACMQ from ISPF, Option 6.
   - Select Option 2 from the NDVRUSER panel.

Refresh ACMQ Data

To enhance query performance, when you enter the ACM Query Facility, the component data available at the time of invocation is used for all the queries performed inside of the facility. Therefore, the data that you search in the ACMQ facility may not contain CA Endevor SCM data that has been updated since you entered ACMQ.

To refresh the component data that the facility uses in its searches, exit the ACMQ facility and then re-enter it.

ACMQ Circular, Indirect, Related References

In addition to reporting on elements that have a direct reference to the object of your search, ACMQ also returns elements that have a circular, indirect, or related relationship to the object of your query. In ACMQ output, a "C", "*", or "R" next to the level number identifies circular, indirect or related references respectively. All query methods allow you to eliminate these references from the ACMQ output.

Circular References

A *circular reference* is an element that uses the same element as an input component. For example, element A that uses element A as an input component. Another example, is element B that uses element C, which uses element B. Circular references are, by default, included in ACMQ results and are identified in ACMQ results by a C character following the level number.

ACMQ is able to identify circular references regardless of the number of elements between the first and second occurrence of the same element.

A circular reference for a specific element is listed only once on every level. When a circular reference for the same element exists on multiple levels (for example, A-B-A as well as A-C-D-A) then it is listed on every level where it appears.

Indirect References

An *indirect reference* is an element whose name and type are the same as a previously found element, but whose location is different. Indirect relationships can result when an element is moved, but not generated, using a Move processor. By default, indirect references appear in ACMQ results and are marked by an asterisk (*) following the level number.

The location of the indirect reference must be mapped with its model (the previously found element). For *where-used* queries, indirect references are taken from locations lower in the map from their model. For *components used* queries, indirect references are taken from locations higher in the map from their model.

The purpose of indirect references is to find relations that may have been lost while using Move processors. An indirect reference that adds no new relations to the ACMQ report, other than itself, is not listed on the ACMQ report.

Example: Where-used Query and Indirect References

Suppose you perform a where-used query for an element named COPYA. Assume the following:

- COPYA is an input component of an element named PGMB, type COBOL.
- PGMB is an input component of an element named PGMC, type LNK.
- All three elements (COPYA, PGMB, and PGMC) exist in the same environment (ENV1) and stage (STG 1).

The output for a where-used query on COPYA would be as follows:

```
LVL ELEMENT  TYPE     ENVIRON SYSTEM   SUBSYS   STG
 1  COPYA    CPY      ENV1    ..       ..        1
 2  PGMB     COBOL    ENV1    ..       ..        1
 3  PGMC     LNK      ENV1    ..       ..        1
```

Now suppose that all three of the elements listed in the previous report are moved to the next stage (STG 2) using a Move processor, meaning that the component list data has been copied, but not rebuilt by a Generate processor. In this case, a where-used query on COPYA would result in the following output:

```
LVL ELEMENT  TYPE     ENVIRON SYSTEM   SUBSYS   STG
 1  COPYA    CPY      ENV1    ..       ..        1
 2  PGMB     COBOL    ENV1    ..       ..        2
 2* PGMB     COBOL    ENV1    ..       ..        1
 3* PGMC     LNK      ENV1    ..       ..        2
```

The first of the two elements that are marked with an asterisk (*) has the same name and type as an element that is known to contain direct references to COPYA. However, because this element has been moved to another CA Endevor SCM location without being rebuilt, ACMQ cannot be sure that it still contains references to COPYA. Therefore, ACMQ treats this element as having only an indirect reference to the object of the query. In the report output, these indirect references are marked with an asterisk and are displayed after the elements that definitely contain direct references to your search.

Note also that elements that contain a reference to an indirect reference are themselves considered indirect references, unless they also contain a direct reference to your search, or to an element that directly references your search.

The following notes apply to this example:

- COPYA has no component list. Thus, when it is moved to the next stage, no component list changes take place, nor do any ACMQ changes take place. Accordingly, ACMQ continues to reference it in ENV1 / STG 1.
- When PGMB COBOL is moved to the next stage, its component list is copied and becomes the component list of PGMB COBOL in STG 2. However, because no changes have been made to the list (other than copying it), the reference in PGMB COBOL in STG 2 to COPYA remains. ACMQ then determines that PGMB COBOL in ENV1 / STG 1 is an indirect reference because it has the same element name and type as PGMB COBOL in ENV1 / STG 2.
- PGMC LNK in ENV1 / STG 1 had a reference to PGMB COBOL in ENV1 / STG 1. When it was moved, it underwent the same changes as PGMB COBOL; thus, PGMC LNK in ENV1 / STG 2 continues to reference PGMB COBOL in ENV1 / STG 1. Because PGMC LNK refers to an indirect reference (PGMB COBOL in ENV1 / STG 1), ACMQ considers it to be an indirect reference.

Related Element References

ACMQ distinguishes between references created by the ACM Component Monitor and references created using the CONRELE utility

A *related reference* is an element whose relationship to another element is identified using the extended processor CONRELE utility, which includes entities related to an element in a component list when generating component list reports and when using the LIST action. Related references appear in ACMQ results, by default, and are marked by an R character following the level number. Indirect references of these CONRELE related references are marked by an *R (R preceded by an asterisk) following the level number.

ACMQ distinguishes between references created by the ACM Component Monitor and references created using the CONRELE utility. The references created with the CONRELE utility appear in ACMQ reports marked with an R character following the level number. ACMQ also searches for indirect references of these CONRELE utility related references. If found they are marked by an *R (R preceded by an asterisk) on the report.

If you have many thousands of related comments or related objects, you may want to enable the option ACMQ_CMNT_QUERY_PERF=ON in the Optional Features Table (ENCOPTBL). When this feature is enabled, the search string is matched with the *beginning* of each comment or object record. By default the option is disabled, which allows searching throughout the entire string.

The ACM Query Panel

The ACM Query panel is the primary panel within ACM that supports all online query functions. Use this panel to perform ACM queries for specified elements, members, comments, or objects. You can also build GENERATE SCL based on the results.

This panel contains the following fields:

Option

Specifies the type of query to be performed. Choose one of the following values:

- **BLANK** - Element Query: Displays WHEre-used or COMponents-used information associated with the element at the inventory location specified. The inventory name specifications can be name masked.
- **M** - Member Query: Displays WHEre-used information for non-element PDS members referenced in element component lists. Optionally, the DSNAME field can be used to further identify the member (for example, SYS1.MACLIB, specified without quotes.)
- **C** - Where-used Comment Query: Displays WHEre-used Comment relationships that have been manually defined using the CONRELE utility in a processor. Use the Comment/Object field to specify the data. This field is case-sensitive.
- **O** - Where-Used Object Query: Displays WHEre-used Object relationships that have been manually defined using the CONRELE utility in a processor. Use the Comment/Object field to specify the data. This field is case-sensitive.

Element/Member

Specifies the name of the element or member object of the query. Valid for options blank or M.

Element Query Information

Specifies the inventory location used to limit which elements the search pertains to.Valid for option blank. Name masking is allowed.

MEMBER Query Information (DSNAME)

Specifies an optional data set name used to limit which library the search pertains to. Valid for option M (member query). Name masking of characters is not allowed and beginning and ending quotes are not allowed.

Comment/Object Query Information

Specifies the name of the comment or object the search pertains to. Valid for options C or O. Name masking of characters is not allowed. Beginning and ending quotes are optional. This field is case-sensitive. For best results we recommend you leave the inventory fields blank.

Query Options

Specifies the type of query to be performed. Choose one of the following values:

- **Where Used/Components Used:** Directs ACMQ to provide where-used or components-used information. Specify “WHE” for where-used information or “COM” for components-used information.

  Note: "COM" is valid for Element queries only, as Members, Comments, and Objects do not have component lists.

- **Foreground/Batch Mode:** Directs ACMQ either to submit a JOB for batch processing or to perform the query processing in foreground mode. (If batch mode is specified, a secondary panel is displayed to allow entry of a JOBCARD.)

- **Create Generate SCL:** Directs ACMQ to create standard CA Endevor SCM GENERATE SCL syntax. A secondary panel is displayed to allow entry of GENERATE action-related options.

- **Exclude circular references:** Specifies whether to exclude circular references. Valid values are Y or N:

  Y - Exclude circular relationships records from the output data returned

  N - Include circular relationships records from the output data returned.

- **Exclude indirect:** Specifies whether to exclude indirect references. Valid values are Y or N:

  Y - Exclude indirect relationships records from the output data returned

  N - Include indirect relationships records from the output data returned.

- **Exclude related:** Specifies whether to exclude related references. Valid values are Y or N:

  Y - Exclude related relationships records from the output data returned

  N - Include related relationships records from the output data returned.

The ACMQ Create GENERATE SCL Panel

The ACMQ Create GENERATE SCL panel enables you to specify GENERATE-related options.

This panel contains the following field areas:

ACTION OPTIONS

The CCID field allows you to specify any 12-character CCID. This CCID is not validated until the built SCL is actually executed. The Comment field is available for you to define a comment of up to 40 characters. No validation is done on the Comment field.

Either Copyback or NoSource can be set to Y, but not both. The Copyback and NoSource options use the inventory specified on this panel. If neither Copyback nor NoSource is specfied, then the default option is Generate In Place. For the Generate in Place option, Inventory is invalid and cannot be specified.

To GENERATE W/COPYBACK or NOSOURCE specify ENV/STG and optionally SYS/SBS:

Use these fields to specify the target CA Endevor SCM location for the Generate action with the Copyback option or with the NoSource option. Environment and Stage Number are required entries. When you specify values for these fields, ACMQ builds Generate SCL with the option you specified. You can also use the System and Subsystem optional fields to further specify the CA Endevor SCM location where the element will be copied back. For Generate action without the Copyback or NoSource options, leave the Environment, Stage Number, System and Subsystem fields blank.

REQUEST DATA SET

Use these fields to define the data set to which you want to write the action requests. The data set must be a partitioned data set or a sequential file, and must be allocated prior to referencing it on this panel.

STOPRC

Use this field to insert a SET STOPRC value. Valid values range from 4 to 16. CA Endevor SCM will stop processing actions when a previous action has a return code equal to or higher than the defined STOPRC.

Note: If you append SCL to existing SCL and the existing SCL already contains a SET STOPRC card, then this option will not modify the existing SET STOPRC card.

OTHER PARTITIONED OR SEQUENTIAL DATA SET

As an alternative to the REQUEST DATA SET fields, you can use the OTHER PARTITIONED OR SEQUENTIAL DATA SET field.

The ACM Submit JOBCARD Statements Panel

The ACM Submit JOBCARD Statements panel accepts batch-related user information (JOBCARD).

Batch SCM Query Utility

The Batch ACM Query utility, BC1JACMQ, allows you to extract ACM data to produce reports or to create SCL statements for generate actions.

The utility accepts SCL as input. Depending on the list syntax you include in BC1JACMQ, the utility produces one of the following types of reports.

- The List Using Components For Element action produces a components-used report that lists elements that use the component you specify.
- The List Used Components For Element action produces a where-used report that lists components that are used by the element you specify.

If the Set Build Action Generate statement is coded in the SCL prior to one of the list actions, then generate action SCL statements are created, instead of a report.

The utility offers equivalent functionality to the online ACM query facility and has some extra filtering options that are not available in foreground.

Create SCL Statements for Generate Actions

You can use the Batch ACM Query utility, BC1JACMQ, to create SCL statements for generate actions.

To create SCL statements for generate actions

1. Add a valid job card at the front of the BC1JACMQ job stream.
2. Change the data set names and other variables to the appropriate values for your installation.
3. Edit the SCL in BC1JACMQ as appropriate for your query.
    a. Edit the Set Build Action Generate statement. For more information, see Set Build Action Generate.

b. After the Set Build Action Generate statement, include one of following List statements.

- To build SCL statements limited to elements that use a specified component, use the SCL for List Using Components for Elements. (see page 72)
- To build SCL statements limited to components that are used by a specified element, use the SCL List Used Components for Element. (see page 75)

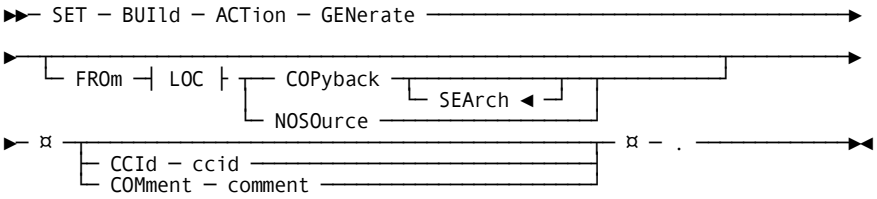
The utility will build SCL statements for generate actions for the elements identified by the List statement.

4. Edit BC1JACMQ to change the ACMOUT statement to allocate, or to point to, an existing 80-character file.
5. Execute the ACM query utility.

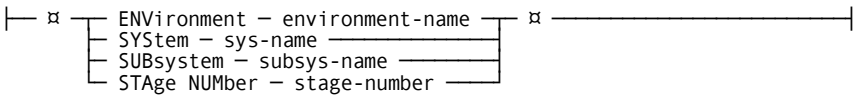
   The SCL statements are written to the ACMOUT file.

Set Build Action Generate

The Set Build Action Generate statement syntax for the Batch ACM Query utility, BC1JACMQ, is shown next:



Expansion of LOC



Parameters

This action uses the following parameters.

SET BUIld ACTion GENerate

Builds SCL statements for generate actions for the elements identified by the List statement coded after the Set Build Action Generate statement in the Batch ACM Query utility. Each generate action SCL statement that is created can be used to execute the generate processor for the current level of the element identified by the List statement.

FROM *LOC 1* COPyback SEArch | NOSOurce

(Optional) Adds the FROM *LOC 1* to each generate action SCL statement.Either COPYBACK or NOSOURCE must also be specified.

COPYBACK SEARCH

(Optional) Causes the current level of the element to be copied back to the FROM stage before the element is generated. CA Endevor SCM first copies the current level of the element back to the FROM stage, then generates the element. CA Endevor SCM searches for the element first in the current environment, then in other stages along the map.

If the element currently exists in the FROM stage, CA Endevor SCM ignores the COPYBACK option and simply generates the element.

SEARCH is only valid with the COPYBACK option. NOSEARCH is no longer valid on the COPYBACK clause. COPYBACK implies SEARCH. It is not necessary to specify SEARCH.

COPYBACK cannot be used with NOSOURCE.

NOSOurce

(Optional) When the target location has a sourced element, the element is generated in place.

When the target location has a sourceless element, the element is generated at the target location using the source of the first occurance of the element found up the map.

When the element does not exist at the target location, the element is generated at the target location using the source of the first occurance of the element found up the map. The source is not fetched to the target. The MCF element created at the target location will contain data similiar to a fetched back element except that the element base and delta name fields will be blank and the record will be marked as a sourceless element.

NOSOURCE cannot be used with COPYBACK. It is not necessary to specify the SEARCH option with NOSOURCE, because NOSOURCE implies SEARCH.

CCID *ccid*

(Optional) Specifies a 1- to 12-character CCID to be included in each of the generate action SCL statements. If your site's system definitions requires a CCID, you must specify this option. If you do not provide a required CCID, then when the generate action SCL statements are executed, the generate action will fail. The generate with copyback action uses the CCID to:

- Set the generate and component list delta CCID field
- Set the last action CCID field
- Set the source and source delta CCID field.

COMMENT *comment-text*

(Optional) Specifies the 1- to 70-character comment to be included in the SCL generate actions. If your site's system definitions requires a comment, you must specify this option. If you do not provide a required comment, then when the generate action SCL statements are executed, the generate action will fail. The generate with copyback action uses the comment to:

- Set the generate and component list delta comment field
- Set the last action comment field
- Set the source and source delta comment field.

LOC

Identifies the location to which the elements are copied before the elements are generated. This is the location from which the element is generated. If you use the FROM keyword, you must at least specify the environment and stage. In addition, you can also specify system, subsystem, or both, but environment and stage are always required.

ENVironment *environment-name*

Specifies the one- to eight-character name of the environment in which you want to perform your query.

STAge NUMber *stage-number*

Specifies the number of the stage in which you want to perform your query.

SYStem *system-name*

Specifies the one- to eight-character name of the system in which you want to perform your query.

SUBsystem *subsystem-name*

Specifies the one- to eight-character name of the subsystem in which you want to perform your query.

Create a Query Report Using the Batch ACM Query Utility

You can use the Batch ACM Query utility, BC1JACMQ, to produce an ACM query report.

To create a query report using the Batch ACM Query utility

1. Add a valid job card at the front of the BC1JACMQ job stream.
2. Change the data set names and other variables in BC1JACMQ to the appropriate values for your installation.
3. Edit BC1JACMQ to specify the SCL required for your query.
   a. To produce a components-used report, use the SCL for List Using Components For Element. (see page 72)
   b. To produce a where-used report, use the SCL for List Used Components For Element. (see page 75)
4. (Optional) Edit BC1JACMQ to change the ACMOUT file format, depending on the type of report layout you prefer.
   - To return the data in report format, which includes headings and page numbers, code this statement:

   ```
   //ACMOUT  DD  SYSOUT=*,DCB=(RECFM=FBA,LRECL=133,BLKSIZE=0)
   ```

   - To return the data in panel display format, which includes the same data shown in report format, except it does not include headings or page numbers and is shown in 80-character layout, code this statement:

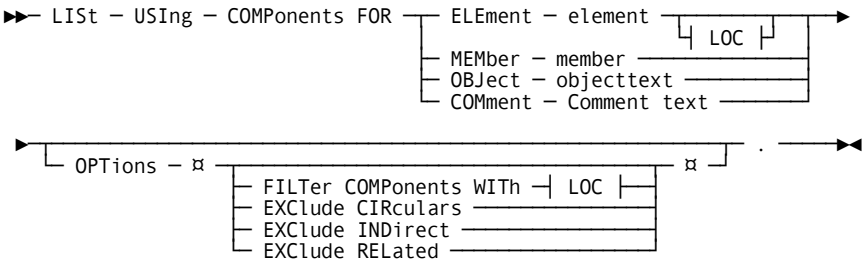
   ```
   //ACMOUT  DD  DSN=uprfx.uqual.file-name,RECFM=FB,LRECL=80
   ```

5. Execute the ACM query utility.

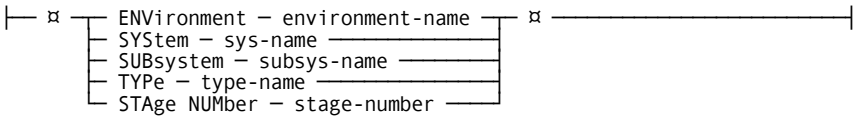
   The data is extracted depending on the SCL list statement you specified and is written to the ACMOUT file in the format you specified.

List Using Components For Element

The List Using Components for Element syntax for the batch ACM Query utility, BC1JACMQ, is shown next:



Expansion of LOC



Parameters

This action uses the following parameters:

LISt USIng COMPonents FOR

Extracts a list of elements that use the specified component. The component is identified by one of the options. You can only code one of the following options:

ELEment *element LOC*

Specifies the 1- to 10-character element name. Name masking is allowed. You can limit the search by specifying the location, but that is optional. You can specify any of the environment, stage, system, subsystem, or type parameters to limit the scope of the query. These location parameters are valid only for Element queries. You can specify no parameters or any combination of environment, stage, system, subsystem, or type. When any of these parameters are specified, the query searches for dependents that used the element when the component resided in this specific inventory location at the time the component list was built. If you want to limit the scope of the query to those elements that currently reside in a specific inventory location you should use the Options clause with the Filter Elements Where Location option.

MEMber *member*

Specifies a component that matches the specified 1- to 10-character member name. Name masking is allowed.

DSName '*dsname*'

(Optional) Specifies the 1- to 44-character data set name for the specified member. Enclose the data set name in single quotes. Name masking is allowed.

Object *object-text*

Specifies the object text string. Name masking is not allowed. Each object record in the ACM query data sets is matched with the specified text string. If the string is found anywhere in the text, the record is used.

Comment *comment-text*

Specifies the comment text string. Name masking is not allowed. Each comment record in the ACM query datasets is matched with the specified text string. If the string is found anywhere in the text, the record is used. For example, if the ACMQ content is “Will ACMQ find this comment string”, then a query using “this*” will not find the string. However, a search for “this comm” will result in a match. ACMQ respects uppercase and lowercase letters if the option MONOCASE_SEARCH is activated in the optional features table ENCOPTBL.

OPTions

(Optional) Specifies various options.

FILTer COMPonents WITh *LOC*

Filters the ACM data to return only those elements found at the specified location. This option eliminates all non-element output records (member, object, and comment) from the output. It eliminates all related elements also. This is true even if you wildcard all the location values.

EXClude CIRculars

Filters the ACM data to exclude elements that have a circular relationship to the object of your search.

EXClude INDirect

Filters the ACM data to exclude indirectly related components.

EXClude RELated

Filters the ACM data to exclude related components.

LOC

The location fields further qualify the element query object. You can specify the location by the environment, system, subsystem, or type.

ENVironment *environment-name*

Specifies the one- to eight-character name of the environment in which you want to perform your query.

SYStem *system-name*

Specifies the one- to eight-character name of the system in which you want to perform your query.

SUBsystem *subsystem-name*

Specifies the one- to eight-character name of the subsystem in which you want to perform your query.

TYPe *type-name*

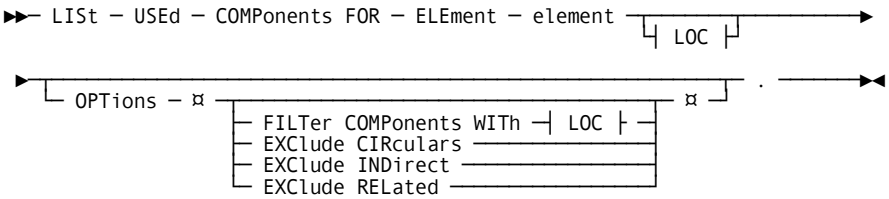
Specifies the type of the elements for which you are searching. The type name can be one to eight characters in length.

STAge NUMber *stage-number*

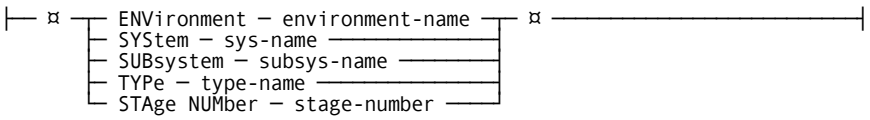
Specifies the number of the stage in which you want to perform your query.

List Used Components For Element

The List Used Components With syntax for the Batch ACM Query utility, BC1JACMQ, is shown next:



Expansion of LOC



Parameters

This action uses the following parameters:

LISt USEd COMPonents FOR ELEment element *LOC*

Extracts a list of components that use the specified element. Optionally, you can limit the search by specifying the element's location.

OPTions

FILTer COMPonents WITh *LOC*

Filters the ACM data to return only those elements found at the specified location. This option eliminates all non-element output records (member, object, and comment) from the output. It eliminates all related elements also. This is true even if you wildcard all the location values.

EXClude CIRculars

Filters the ACM data to exclude components that have a circular relationship to the object of your search.

EXClude INDirect

Filters the ACM data to exclude indirectly related components.

EXClude RELated

Filters the ACM data to exclude related components.

LOC

The location fields further qualify the element query object. You can specify the location by the environment, system, subsystem, or type.

ENVironment *environment-name*

Specifies the one- to eight-character name of the environment in which you want to perform your query.

SYStem *system-name*

Specifies the one- to eight-character name of the system in which you want to perform your query.

SUBsystem *subsystem-name*

Specifies the one- to eight-character name of the subsystem in which you want to perform your query.

TYPe *type-name*

Specifies the type of the elements for which you are searching. The type name can be one to eight characters in length.

STAge NUMber *stage-number*

Specifies the number of the stage in which you want to perform your query.

Chapter 4: Analyzing and Managing Software Configuration Information

This section contains the following topics:

Software Control Language (SCL)

CA Endevor SCM's Software Control Language (SCL) is a freeform language, with English-like statements, that allows you to manipulate elements and operate against multiple environments within CA Endevor SCM. You can either code SCL manually or generate it through selected batch panels.

The following section describes the various SCL statements that can be used in conjunction with ACM.

Note: For more information about SCL, see the *SCL Reference Guide*.

The LIST Action

The LIST action scans elements or members in the Master Control File or a library, and generates a list of elements/members that meet your specific selection criteria.

The CA Endevor SCM LIST action will invoke ACMQ to satisfy simple, component name, inventory location, "WHERE [INPUT | OUTPUT]" components criteria. Additional criteria such as CCID or historical dependencies ("ALL" parameter) require CA Endevor SCM to search its base / delta libraries to perform the search. Note that CA Endevor SCM will default to using ACMQ whenever possible to perform component-related list processing.

Within the LIST action are several clauses that pertain specifically to ACM. You can use these clauses to limit selection criteria as it applies to component lists. A brief explanation of each clause is provided below.

WHERE Clauses

WHERE clauses instruct CA Endevor SCM to generate a list of elements or members where specific criteria is met. Within the WHERE clause of the LIST action are two sections that pertain specifically to ACM:

- **WHERE component spec —** Allows you to indicate that only those component lists containing input, output, and/or processor components matching a designated component name can be selected for the LIST action. Conversely, you can indicate that you want to see only those component lists that do not contain the designated component.
- You can also indicate that a specific range of components be considered, using the **THROUGH component-name** clause in conjunction with the **WHERE component spec** clause.
- If the component is footprinted, you can specify the following additional selection criteria:
  - Version and/or level of the component. When you specify a version number, only those elements with components matching that number will be selected for the LIST action.
  - Similarly, if you specify a level number here, only those elements with components matching that level will be selected.
  - Component location information (environment, system, subsystem, type, and stage, or file or dsname).
- **WHERE ACM component spec —** Allows you to set compound criteria, when component lists must contain both one component and another or either one component or another. You can also specify that a component list that does not contain both one component and another or either one component or another be selected.
- You can have any number of compound criteria in a single clause.

Build Clauses

BUILD clauses indicate specific information to be applied to each action statement. Within the BUILD clause are two sections that apply to CA Endevor SCM ACM:

- **BUILD LEVEL —** Indicates whether you want the version and level of the specified element to appear on the action cards generated by the LIST request. Three options are available when coding this clause:
  - **CURRENT —** If the **WHERE component spec** clause has not been coded for the action, or no component list exists, CA Endevor SCM defaults to the current level of the element.
  - **NONE —** Indicates that the current version and level are not to be listed for the element.
  - **ACTUAL —** Indicates that the level of the component as recorded in the component list, rather than the current level of the element as recorded in the Master Control File, should be used when building the action statement.
- **BUILD WITH COMPONENTS —** Indicates that action cards should be generated for every input component that is associated with the specified element.

The Print Action

The PRINT action prints selected information about an element(s) or library member(s), depending on the criteria entered.

Note the COMPONENTS option. When you select this option, CA Endevor SCM prints all component information (element, processor, input, and output data) relating to the element specified. You can code this option alone or in conjunction with the following PRINT options: BROWSE, CHANGE, HISTORY, SUMMARY, or MASTER.

CA Endevor SCM prints as much information as is available for the component list. For example, if you request the COMPONENT CHANGES option but there are no changes to the output components section, that section would not appear on the associated listing.

The Set Build Statement

The SET BUILD statement is used when you do not code BUILD information in the LIST action request. As with the BUILD statement in the LIST action:

- SET BUILD LEVEL CURRENT defaults to the current level of the element if the WHERE component-spec clause has not been coded for the action or if no component list exists.
- SET BUILD NONE indicates that the current version and level are not to be listed for the element.
- SET BUILD LEVEL ACTUAL indicates that the actual level of the component should be used when building the request.
- SET BUILD WITH COMPONENTS indicates that action cards should be generated for every input component associated with the designated element.

The Set Options Statement

The SET OPTIONS statement allows you to indicate that one or a series of options should be applied to all subsequent actions in a LIST request (until the next SET OPTIONS or a CLEAR OPTIONS statement is encountered, or processing ends). Note that those options that do not apply to the action are ignored. In addition, if you indicate a particular option in the action statement and have also coded that option in the SET OPTIONS statement, the entry in the action statement overrides the SET OPTIONS selection. The following options apply specifically to ACM:

- ONLY COMPONENTS (used in conjunction with the DELETE action) allows you to delete the component lists for an element, but not the element itself.
- COMPONENT (used in conjunction with the PRINT action) provides printed output pertaining to component list information for the element specified. You can use this option alone, or in combination with one of the following PRINT options: BROWSE, CHANGE, HISTORY, SUMMARY, or MASTER.

The Set Where Statement

The SET WHERE statement is used when you do not code WHERE information in the LIST request. The selection criteria you can enter here is the same as would be entered with the LIST action. Again, the following clauses pertain specifically to ACM:

- WHERE component-spec
- WHERE ACM component-spec

Clear Statements

CLEAR statements clear the information that has been designated by the related SET statements.

CLEAR BUILD clears like information you have entered in the SET BUILD statement.

- CLEAR OPTIONS clears like information you have entered in the SET OPTIONS statement. Note, however, that to clear the SET OPTIONS COMPONENT statement, you must enter the statement CLEAR OPTIONS PRINT.
- CLEAR WHERE clears the like information you have entered in a SET WHERE statement.

How to Perform Impact Analysis

You can perform change impact analysis functions using the configuration information that ACM collects in combination with CA Endevor SCM's Software Control Language (SCL). For example, you can:

- Analyze system behavior.
- Propagate a component change to all affected programs.

  Note: The Autogen action option provides a more efficient way to propagate component changes. When specified on an Add, Update, or Generate action for a component element, Autogen automatically generates the elements that use the component element. Autogen requires that ACM be enabled at your site. For a more detailed description of Autogen, see Autogen Action Option (see page 85). For more information about Autogen, see the scenario-based knowledge document *How to Automatically Generate "Using" Elements (*HTML/EndevorSCM_Autogen_Scenario_ENU/index.htm*), in the Scenario Guide (*HTML/EndevorSCM_Scenarios_ENU/index.htm*).*
- Validate a system for consistent use of components.

  Note: The Validate action is a more efficient way to validate components. The SCL statement Validate lets you verify that all components exist and are valid. Valdiate requires that ACM be enabled at your site. For more information, see Validate Components (see page 90).
- Recreate past program versions.
- Move related source components during promotion to production.

How to Analyze System Behavior

Programs sometimes abort after being moved into production. In order to fix the cause, problems must first be identified. Problem-solving can be extremely time-consuming when working in the absence of tools which specifically help detect problem areas.

This example uses ACM to find out why element FINAPP01 experienced a production outage. This trouble-shooting scenario involves four steps:

1. Create PRINT SCL.
2. Submit the job for batch execution and view the resulting report.
3. Analyze system behavior.

How to Create PRINT SCL

Although you could browse change history online, the following demonstrates how you can identify changed components using SCL's Print command as follows:

```
PRINT    ELEMENT            FINAPP01
FROM     ENVIRONMENT        DEMO
         SYSTEM             FINANCE
         SUBSYSTEM          ACCTPAY
         TYPE               COBOL
         STAGE              P
OPTIONS HISTORY            COMPONENTS.
```

By coding:

- PRINT ELEMENT FINAPP01 — Instruct ACM to print element FINAPP01.
- FROM ENVIRONMENT — Further specifies the production stage (STAGE P) of the ACCTPAY subsystem within the FINANCE system.
- OPTIONS HISTORY COMPONENTS — Instruct ACM to print the component list with history to determine what and where components changed.
- Since no TO statement has been coded, CA Endevor SCM uses the default TO C1PRINT to print the element.

How to Submit the Job for Batch Execution and View the Resulting Report

Once the SCL has been coded, submit the job for batch execution using the batch processing capabilities. The SCL commands are now automatically applied to the information collected and stored by ACM.

After submitting the job, you can view the batch execution results.

How to Analyze System Behavior

The Processor Information section in the above component list report shows that the processor used to compile and link the element has changed since the last compilation. The Input Components section shows that two input components—copybooks HEADER1 and PAGING—have changed. At this point, you can either view the changes made to these two components online, or print a report of the changes made.

ACM allows you to analyze system behavior and quickly determine the changes that might have caused our production failure.

How to Propagate a Component Change to All Affected Programs

Note: The Autogen action option provides a more efficient way to propagate component changes. When specified on an Add, Update, or Generate action for a component element, Autogen automatically generates the elements that use the component element. Autogen requires that ACM be enabled at your site. For a more detailed description of Autogen, see Autogen Action Option (see page 85). For more information about Autogen, see the scenario-based knowledge document *How to Automatically Generate "Using" Elements (HTML/EndevorSCM_Autogen_Scenario_ENU/index.htm), in the Scenario Guide (HTML/EndevorSCM_Scenarios_ENU/index.htm).*

When changes are made to a component, it is necessary to propagate those changes to all programs containing that component.

In the example that follows, copybook COPYREC needs to be changed in order to complete a change request for program C1PRTX00. Once the change is completed and tested in program C1PRTX00, the change to copybook COPYREC will be propagated to all programs in which it is used.

If you are not using AUTOGEN, then propagating component changes to all affected programs requires the following steps:

1. Change and test the retrieved copybook and program.
2. Add the copybook and program to the entry stage.
3. Create LIST SCL and execute it.
4. Tailor the generated SCL and execute it.

How to Change and Test the Retrieved Copybook and Program

Program C1PRTX00 was retrieved from CA Endevor SCM to complete Application Systems Request #9043. While making the change to program C1PRTX00, a change is also made to copybook COPYREC.

How to Add the Copybook and Program to the Entry Stage

Once the changes have been completed, program C1PRTX00 and copybook COPYREC are added to the entry stage. Before system testing can begin, the change to copybook COPYREC needs to be propagated to all programs in which it is used.

How to Create and Execute LIST SCL

The LIST command identifies all elements in the system Personnel that use the copybook COPYREC as an input component.

```
LIST      ELEMENTS       *
FROM      ENVIRONMENT    DEM0
          SYSTEM         PERSONEL
          SUBSYSTEM      *
          TYPE           *
          STAGE          P
TO        DSNAME 'BST.C1DEMO.SRCLIB'
WHERE     INPUT COMPONENT EQUAL COPYREC.
```

By coding:

- LIST ELEMENTS * — Instruct ACM to look at all elements, regardless of name.
- FROM ENVIRONMENT — Restrict the search of elements to the production stage (STAGE P) of the Personnel system (SYSTEM PERSONEL), regardless of subsystem and type.
- TO DSNAME — Instruct ACM to write out the list of elements which meet the search criteria for the data set BST.C1DEMO.SRCLIB. Since no member name was coded in the OPTIONS statement, a default member name (TEMPNAME) will be created. If member TEMPNAME already exists, it will not be replaced, and this will cause an error that stops execution of the LIST command.

  However, if member TEMPNAME does not exist, it will be created and SCL statements will be written identifying every element that meets the WHERE conditions.

  WHERE — Instruct ACM to look at every element with a component list, and to search the component list for an input component of COPYREC. Every element that contains INPUT COMPONENT = COPYREC will have an SCL action statement written out to the member TEMPNAME in the BST.C1DEMO.SRCLIB data set.

The SCL generated from the above request is as follows:

```
SET FROM ENVIRONMENT DEMO     SYSTEM PERSONEL SUBSYSTEM EMPMAINT
         TYPE COBOL    STAGE NUMBER 2.
    &&ACTION ELEMENT   C1PRTX00     VERSION 01 LEVEL 02
    &&ACTION ELEMENT   C1PRTX20     VERSION 01 LEVEL 02
    &&ACTION ELEMENT   C1PRTX30     VERSION 01 LEVEL 05
    &&ACTION ELEMENT   C1PRTX40     VERSION 01 LEVEL 05
```

How to Tailor and Execute the Generated SCL

Now add the following SET commands to the generated SCL:

```
SET ACTION GENERATE.
SET OPTION COPYBACK.
SET FROM ENVIRONMENT DEMO     SYSTEM PERSONEL SUBSYSTEM EMPMAINT
         TYPE COBOL    STAGE NUMBER 1.
    &&ACTION ELEMENT   C1PRTX00     VERSION 01 LEVEL 02.
    &&ACTION ELEMENT   C1PRTX20     VERSION 01 LEVEL 02.
    &&ACTION ELEMENT   C1PRTX30     VERSION 01 LEVEL 05.
    &&ACTION ELEMENT   C1PRTX40     VERSION 01 LEVEL 05.
```

By coding:

- SET ACTION — Instruct ACM to change all of the &&ACTION statements to GENERATE. ("&&ACTION" appears on the action cards generated for each element/component when you do not specify an action in the LIST request.) The GENERATE action executes the generate processor (compile) for all programs which use the copybook COPYREC.
- SET OPTION — By setting the COPYBACK option, ACM copies back the current version/level of each program to the entry stage. (if it does not already exist in the entry stage) before executing the generate processor.

Autogen Action Option

When specified on an Add, Update, or Generate action for a component element, the *Autogen action option* automatically generates using elements. A *using* element is an element that uses a component element. For example, if Autogen is specified for copybook COPYA, then the programs that use that copybook are known as using elements. Specifying Autogen for an element of Type Macro automatically generates the source elements that use the macro, which then generates the appropriate LNK elements.

Autogen is available in batch only for the Add, Update, and Generate actions and cannot be used in packages. CCIDs and comments from the original generated element are used. Autogen generates *only* those using elements that are found at the same inventory location as the target component or are found higher up the logical map. To generate using elements located across Systems or Subsystems, you can use the Autogen Span options. For more information about the Span options, see Autogen Spans Across Systems and Subsystems in the *Administration Guide*.

If you specify the Autogen option on any one of a group of statements, then all of those statements are resolved based on the current inventory contents before any statement is executed. Statements such as GENERATE ELEMENT * create actions based on the location and options of the Generate action. During processing, duplicate Generate actions are eliminated, and the NoSource option is enabled for all the Generate actions built by Autogen. An administrator can change the behavior of the Autogen feature, by activating AUTOGEN_SOURCE in the Optional Features Table (ENCOPTBL). When this option is activated, the Generate actions for the using elements are built with the Copyback, instead of the NoSource, option.

Autogen improves processing by eliminating duplicate processing of components and reduces the work that is required of users, who no longer must use the Automated Configuration Manager Query facility (ACMQ) to create additional Generate actions for element components and then run another batch job to process them.

You can run Autogen in simulation mode, to preview the effects of an Autogen request.

The following restrictions apply to Autogen:

- Autogen requires Global Type Sequencing so that the elements that use a component are generated after all of the components have been processed (for example, macros before source, source before load modules).
- Autogen only acts on components whose Types are listed in the Global Type Sequencing table. If the component's Type is not listed in the Global Type Sequencing table, the Autogen request is ignored.
- Your site must have purchased and activated the option CA Endevor Automated Configuration. You can use the Display Site panel to determine if this option is activated (ASCM=Y) at your site.
- Autogen cannot be specified on actions that are included in a package, because approvers must see the SCL statements that they are approving.
- Autogen and the bypass generate element (GENERATE ELEMENT=N) options are mutually exclusive.
- Autogen is a batch option. It cannot be specified on foreground requests.

Note: For more information about Autogen, see the scenario-based knowledge document *How to Automatically Generate "Using" Elements (*HTML/EndevorSCM_Autogen_Scenario_ENU/index.htm*), in the Scenario Guide (*HTML/EndevorSCM_Scenarios_ENU/index.htm*).*

How to Validate a System for Consistent Use of Components

Note: The Validate action is a more efficient way to validate components. The SCL statement Validate lets you verify that all components exist and are valid. Valdiate requires that ACM be enabled at your site. For more information, see Validate Components (see page 90).

In this example, a large number of changes have gone into production. Many common routines have changed which, in turn, have affected programs. ACM allows you to make sure that all programs are using components (copybooks, CALLed routines, etc.) that are at the right version/level.

Validating consistent use of components involves three steps:

1. Create LIST SCL and execute it.
2. View Execution Report.
3. Check generated SCL for inconsistent components.

How to Create and Execute LIST SCL

In this example, we use the LIST command to select all elements and their related components.

```
LIST     ELEMENTS       *
FROM     ENVIRONMENT    DEMO
         SYSTEM         PERSONEL
         SUBSYSTEM      *
         TYPE           *
         STAGE          P
TO       DSNAME 'BST.C1DEMO.SRCLIB'
         MEMBER 'PVALID'
WHERE    COMPONENTS = *
BUILD    WITH COMPONENTS
OPTIONS DETAIL REPORT.
```

By coding:

- LIST ELEMENTS * — Instruct ACM to search for all (*) elements as specified by the FROM statement
- FROM ENVIRONMENT — Restrict the search to the production stage (STAGE P) of the Personnel system (SYSTEM PERSONEL), regardless of subsystem and type.
- TO DSNAME — Instruct ACM to write out the list of elements which meet the search criteria to the data set BST.C1DEMO.SRCLIB.
- WHERE COMPONENTS = * — Instruct ACM to select each element with a component list in the Personnel system in Stage P.
- BUILD WITH COMPONENTS — Instruct ACM to build actions for each element and all input components for the element.
- OPTIONS DETAIL REPORT — Instruct ACM to list each element searched and its related components in the Execution Report. ACM sorts the information collected by environment, system, subsystem, type, stage, and element, and produces List-generated SCL in the member PVALID. Wherever there are inconsistent components, ACM highlights them.

There are two outputs: an Execution Report and List-generated SCL. You always receive an Execution Report, but this time ACM produces a larger Execution Report because of the DETAIL REPORT option. The information in the report is then sorted and sent to the indicated member. All inconsistencies are identified within the member PVALID.

How to View the Execution Report

The first portion of the sample Execution Report is a Syntax Request Report which numbers the SCL statements and highlights any syntax errors in the SCL. Each statement can result in more than one action being performed.

The second section of the Execution Report details each action generated for the original LIST SCL statement that meets the WHERE selection criteria. Each action is listed because of the OPTIONS DETAIL REPORT clause in the SCL. If this option had not been specified, only one action—LIST ELEMENT *—would be generated and, consequently, CA Endevor SCM would search for matches that meet the WHERE selection criteria.

How to Check Generated SCL for Inconsistencies

The components are then collected and sorted before being written to the file specified in the TO statement: 'BST.C1DEMO.SRCLIB(PVALID)'. The resulting List-generated SCL pinpoints inconsistent components, that is, components which share the same name but indicate more than one version/level. These inconsistencies are clearly highlighted in two ways:

1. An asterisk (*) appears in column 1, to the left of the inconsistent components.
2. A flag indicating duplication (**DUPLICATE**) appears to the right of the inconsistent components.

In the example, the List-generated SCL shown below indicates two inconsistent components:

```
SET FROM ENVIRONMENT DEMO     SYSTEM PERSONEL SUBSYSTEM EMPMAINT
         TYPE COBOL    STAGE NUMBER 2.
    &&ACTION ELEMENT   C1CALLER     VERSION 01 LEVEL 00.
    &&ACTION ELEMENT   C1SUB01      VERSION 01 LEVEL 00.
    &&ACTION ELEMENT   C1SUB02      VERSION 01 LEVEL 00.

SET FROM ENVIRONMENT DEMO     SYSTEM PERSONEL SUBSYSTEM EMPMAINT
         TYPE COPY     STAGE NUMBER 2.
    &&ACTION ELEMENT   C1CLINK      VERSION 01 LEVEL 00.
    &&ACTION ELEMENT   WSWITCH      VERSION 01 LEVEL 00.**DUPLICATE**
    &&ACTION ELEMENT   WSWITCH      VERSION 01 LEVEL 01.

SET FROM ENVIRONMENT DEMO     SYSTEM PERSONEL SUBSYSTEM EMPMAINT
         TYPE INCLUDES STAGE NUMBER 2.
    &&ACTION ELEMENT   FDPRINT      VERSION 01 LEVEL 03.
    &&ACTION ELEMENT   FDPRINTS     VERSION 01 LEVEL 02.
    &&ACTION ELEMENT   PDSTOP       VERSION 01 LEVEL 02.**DUPLICATE**
    &&ACTION ELEMENT   PDSTOP       VERSION 01 LEVEL 03.
SET FROM DSNAME BST.QATEST.LOADLIB2                    .
    &&ACTION ELEMENT   C1CALLER                        .
    &&ACTION ELEMENT   C1SUB01                         .
    &&ACTION ELEMENT   C1SUB02                         .
```

By viewing the generated SCL for asterisks (*), you can see that there are two components with duplicate version/levels. Refer back to the Execution Report in Step 2 and find the elements that have the inconsistent components (noted by arrows).

For example, to find the element using the input component PDSTOP Version 01 Level 02, scan the Execution Report for all occurrences of that component. Then check the list action data to determine which elements are using the older level of PDSTOP. In this example, the element in question is C1SUB02.

Validate Components

The SCL statement Validate lets you verify that all components exist and are valid. This feature is implemented as a batch element action. You can build the SCL statement in foreground and either submit it directly from foreground or schedule it for later execution. The Validate Elements panel is where you specify the location of the element and indicate what type of validation you want to perform.

Follow these steps:

1. Allocate a partitioned or sequential data set. This is the data set to which you plan to write the action request.
2. Select option 3, Batch, on the Primary Options Menu,

   The Batch Options Menu opens.
3. Specify the data set name in the Request Data Set field. This is the data set to which you want to write the action requests. This data set must be a partitioned data set or a sequential file, and must be allocated prior to referencing it on this panel. As an alternative, you can use the OTHER PARTITIONED OR SEQUENTIAL DATA SET field. Select option 1, BUILD SCL and press Enter.

   The SCL Generate panel opens. This panel allows you to select the type of action request you want to generate, or to request an element display. The request data set and append information defined on the Batch Options Menu appear at the bottom of the screen.
4. Complete the fields and set the COMPONENT VALIDATION field to Y. Then enter VE. The element must exist at the location you specify, although you can use full or partial name masking on all the FROM LOCATION fields.

Note: For more information about building SCL statements, see Build Element Action SCL in Foreground for Execution in Batch, in the *User Guide*.

Note: The Validate statement can only be performed in batch. For more information about the SCL statement, see The Validate Statement in the *SCL Reference Guide.*

How to Recreate Past Program Versions

Sometimes you may want to recreate a program's load module as it existed at a past point in time. This requires using not only an older version/level of the element, but also all the related components that were used at the date/time the load was created.

In the example that follows, program FINAPP01 needs to be recreated as of a production execution on May 1, 2001. By viewing the Component Level Information on the component list for the element, you can pinpoint the desired recreation date/time. Using the information in that version/level of the component list, you can then recreate the element and its components in the entry stage.

Recreating past program versions involves three steps:

1. Browse the component list at Stage 2.
2. Code LIST SCL and submit for execution.
3. Tailor the generated SCL and submit for execution.

How to Browse the Component List at Stage 2

By browsing an element's component list at Stage 2, you can determine the generate date/time of the module you want to recreate.

How to Code LIST SCL and Submit for Execution

After browsing the component list, you are ready to code SCL using the LIST command.

```
LIST      ELEMENTS        FINAPP01
FROM      ENVIRONMENT     DEMO
          SYSTEM          FINANCE
          SUBSYSTEM       ACCTPAY
          TYPE            COBOL
          STAGE           P
TO        DSNAME 'BST.C1DEMO.SRCLIB'
          MEMBER 'RC1SUB01'
WHERE     GENERATE DATE = 05/01/01 TIME = 12:50
          COMPONENTS = *
BUILD     WITH COMPONENTS LEVEL ACTUAL.
```

By coding:

- LIST ELEMENTS FINAPP01 — Instruct ACM to search for element FINAPP01.
- FROM ENVIRONMENT — Restrict the search to the production stage (STAGE P) of the ACCTPAY subsystem within the FINANCE system.
- TO DSNAME — Instruct ACM to write out the list of elements which meet the search criteria for the data set BST.C1DEMO.SRCLIB.
- WHERE GENERATE DATE = — Instruct ACM to view the component list for element FINAPP01 that was created specifically on 05/01/01 at 12:50.
- BUILD WITH COMPONENTS ACTUAL — Instruct ACM to use the specific version/level for each input component found on that component list. (Otherwise, ACM would use the current version/level of each input component.)

Submit the LIST request for batch processing.

How to Tailor and Submit the Generated SCL for Execution

The generated SCL in member TEMPNAME of data set 'BST.C1DEMO.SRCLIB' looks like this example:

```
SET FROM ENVIRONMENT DEMO     SYSTEM FINANCE SUBSYSTEM ACCTPAY
         TYPE COBOL    STAGE NUMBER 2.
    &&ACTION ELEMENT   FINAPP01     VERSION 01 LEVEL 00.

SET FROM ENVIRONMENT DEMO     SYSTEM FINANCE SUBSYSTEM ACCTPAY
         TYPE COPY     STAGE NUMBER 2.
    &&ACTION ELEMENT   HEADER1      VERSION 01 LEVEL 00.
    &&ACTION ELEMENT   PAGING       VERSION 01 LEVEL 01.
```

By viewing the component list from May 1, 2001 at 12:50 p.m., you can see that:

- element FINAPP01 was at a VVLL of 0100.
- element HEADER1 was at a VVLL of 0100.
- element PAGING was at a VVLL of 0101.

Thus, we see that elements FINAPP01, HEADER1, and PAGING have all been changed since May 1, 2001.

Now, we edit this SCL by coding RETRIEVE and ADD actions as follows:

```
SET ACTION RETRIEVE
SET TO FILE TMPPDS.
SET  OPTIONS  COMMENT 'RETRIEVE FINAPP01'  CCID  'EMG0195'.
SET FROM ENVIRONMENT  DEMO       SYSTEM FINANCE      SUBSYSTEM ACCTPAY
         TYPE COBOL  STAGE NUMBER 2 .
    &&ACTION ELEMENT FINAPP01          VERSION 01    LEVEL 00
SET FROM ENVIRONMENT  DEMO       SYSTEM FINANCE     SUBSYSTEM ACCTPAY
         TYPE COBOL  STAGE NUMBER 2 .
    &&ACTION ELEMENT HEADER1          VERSION  01    LEVEL 00
    &&ACTION ELEMENT PAGING           VERSION  01    LEVEL 01 .
CLEAR TO ALL
CLEAR FROM ALL.
SET FROM ENVIRONMENT  DEMO       SYSTEM FINANCE     SUBSYSTEM ACCTPAY
         TYPE COBOL  STAGE NUMBER 1
    &&ACTION ELEMENT FINAPP01        VERSION 01    LEVEL 00 .
SET FROM ENVIRONMENT  DEMO       SYSTEM FINANCE     SUBSYSTEM ACCTPAY
         TYPE COBOL  STAGE NUMBER 1 .
    &&ACTION ELEMENT       HEADER1         VERSION  01    LEVEL 00 .
    &&ACTION ELEMENT      PAGING             VERSION  01    LEVEL 01 .
```

Note: The ADD action would create new levels that will eliminate intervening levels (regression). An alternative would be not to add to the entry stage, but to compile and test in test libraries.

When you finish editing the SCL, resubmit the request for batch processing.

How to Move Related Source Components During Promotion to Production

Often, a request causes changes to several programs. Once tested and changed at the entry stage, a program and its related components will need to be moved to the next mapped stage. Typically, you would change the programs for a request under the same CCID. To insure that a CCID moves successfully from the entry stage to the next mapped stage with related components, specify the move not only by CCID, but WITH COMPONENTS.

In the following example, we are about to move a CCID from the entry stage to the next mapped stage. This CCID is an Application System Request (ASR#053010). When moving this CCID, we also want to move all related components.

The scenario for moving the CCID and its related components from the entry stage to the next mapped stage involves four steps:

1. Create LIST SCL.
2. Run a batch execution (to generate SCL using the LIST action).
3. Tailor the generated SCL.
4. Run a batch execution (to execute the tailored SCL).

How to Create LIST SCL

Begin the move scenario by creating SCL using the LIST command. The LIST command finds all elements modified by a specific CCID in the entry stage (and their related input components).

To begin moving CCID ASR#053010 from the entry stage to the next mapped stage, code the SCL as follows:

```
LIST      ELEMENTS        *
FROM      ENVIRONMENT     DEMO
          SYSTEM          PERSONEL
          SUBSYSTEM       *
          TYPE            *
          STAGE           D
TO        DSNAME 'BST.C1DEMO.REQDSN'
WHERE     CCID EQUALS 'ASR#053010'
          INPUT COMPONENTS EQUAL '*'
BUILD     WITH COMPONENTS LEVEL ACTION MOVE
OPTIONS   MEMBER 'ASR53010'.
```

By coding:

- LIST ELEMENTS * — Instruct ACM to list all (*) the elements from the PERSONEL system within the DEMO environment. Specify all (*) subsystems and types, and identify the stage as 'D' (the entry stage).
- TO DSNAME — Instruct ACM to write out the SCL request TO a data set named ‘BST.C1DEMO.REQDSN’.
- WHERE CCID EQUALS — Instruct ACM to find all (*) input components for the CCID ‘ARS#053010’.
- BUILD WITH COMPONENTS ACTION MOVE — Instruct ACM to build this SCL in this data set member with all the components, using a MOVE action rather than the default of &&ACTION
- OPTIONS — Specify that the member for the requested data set, BST.C1DEMO.REQDSN, be named ‘ASR53010’.

How to Run a Batch Execution

Once the SCL has been coded, run a batch execution to generate SCL using the LIST action. (This will be the first of two executions.) Your SCL commands are now automatically applied to the information which is collected and stored by ACM. The end result is a list of elements and related components.

In the example, the outcome is a list of all elements for the CCID (ASR#053010) and their related input components (programs, copybooks, INCLUDE modules).

```
SET FROM ENVIRONMENT DEMO     SYSTEM PERSONNEL  SUBSYSTEM EMPMAINT
         TYPE COBOL    STAGE NUMBER 1.
    &&MOVE. ELEMENT   C1CALLER     VERSION 01  LEVEL 00  .
    &&MOVE. ELEMENT   C1SUB01      VERSION 01  LEVEL 00  .
    &&MOVE. ELEMENT   C1SUB02      VERSION 01  LEVEL 00  .

SET FROM ENVIRONMENT DEMO     SYSTEM PERSONNEL  SUBSYSTEM EMPMAINT
         TYPE COPY     STAGE NUMBER 1.
    &&MOVE. ELEMENT   C1CLINK      VERSION 01  LEVEL 00  .
    &&MOVE. ELEMENT   WSWITCH      VERSION 01  LEVEL 00  .

SET FROM ENVIRONMENT DEMO     SYSTEM PERSONNEL  SUBSYSTEM EMPMAINT
         TYPE INCLUDES STAGE NUMBER 1.
    &&MOVE. ELEMENT   FDPRINT      VERSION 01  LEVEL 03  .
    &&MOVE. ELEMENT   FDPRINTS     VERSION 01  LEVEL 02  .
    &&MOVE. ELEMENT   PDSTOP       VERSION 01  LEVEL 02  .

SET FROM DSNAME BST.QATEST.LOADLIB1                    .
    &&MOVE. ELEMENT   C1SUB01                           .
    &&MOVE. ELEMENT   C1SUB02                           .
```

How to Tailor the Generated SCL

In the extracted sample below, the SET FROM DSNAME statement highlights that some "non-footprinted" input components should also be moved to the next mapped stage.

```
SET FROM DSNAME BST.QATEST.LOADLIB1                .
     MOVE ELEMENT C1SUB01                          .
     MOVE ELEMENT C1SUB02                          .
```

These load modules have source inside of CA Endevor SCM, and that source has already been selected by LIST as noted by elements 2 and 3 in the first SET command. You can now delete this SET statement and associated MOVE ELEMENT statements.

How to Run a Final Batch Execution

At this point, you can submit the SCL for final batch execution. Once executed, all related components will be automatically moved with a module as it is promoted from one stage of development to another. In the example, all components relating to the CCID ARS#053010 will be moved with that CCID as it is promoted from the entry stage to the next mapped stage.

How to Add Related Elements to a Component List

You can use the CONRELE utility to include entities related to an element in a component list. The entities can be data sets, CASE entities, JCL, parameter list members, documentation members, etc. The entities do not have to be CA Endevor SCM elements.

CONRELE accepts user syntax from the ENDVRIPT DD statement. After the parsing process is complete the data is formatted as special component record types and processed with the rest of the component list. The related data portion is appended to the end of the component list. You are not required to store the input in CA Endevor SCM.

You must include the CONRELE utility as a processor step and you must provide the input. Use the following sample processor to execute CONRELE:

```
//STEPxx   EXEC   PGM=CONRELE
//NDVRIPT  DD     DSN=&user.data.set,DISP=shr
```

Note: For more information about the CONRELE utility, see the *Extended Processors Guide.*

How to Write Elements to an External Location

The CONWRITE utility allows you to take component list data and store it in an external data set or use the component list data as input to other processes. You must first use the CONRELE utility to create related input components, related output components, related objects and related comments before using the CONWRITE utility.

You must use the extended form of the CONWRITE utility to extract component list records. CONWRITE reads WRITE ELEMENT control statements from the CONWIN DD statement to determine which component list records to extract.

You can write the component list record to an external data set or you can pass the component list record to a user specified exit program. The following is sample JCL for using the CONWIN DD statement to extract a component list with the extended form of CONWRITE:

```
//WRITE EXEC    PGM=CONWRITE
//COMPOUT DD    DSN=&user.data.set,DISP=PASS,UNIT=SYSDA,
//              SPACE=(TRK,(3,5),RLSE),
//              DCB=(RECFM=VB,LRECL=80,BLKSIZE=6160,DSORG=PS)
//CONWIN  DD     *
  WRITE ELEMENT &c1element
     FROM ENV &c1envmnt SYSTEM &c1system SUBSYSTEM &c1subsys
          TYPE &c1type STAGE &c1stgid
  TO DDN COMPOUT
  OPTION COMPONENT.
/*
```

Note: For more information about the CONWRITE utility, see the *Extended Processors* Guide.

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

T

U

V

W