CA Endevor® Software Change Manager

Extended Processors Guide

Version 17.0.00

Second Edition

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 Extended Processors (CA Endevor Extended Processors)
- CA Top Secret® for z/OS (CA Top Secret)
- CA Librarian® Base for z/OS (CA Librarian)
- CA Panvalet® for z/OS (CA Panvalet)
- CA Easytrieve® Online Query (CA Easytrieve Online Query)
- CA Ideal™ for CA Datacom® (CA Ideal for CA Datacom)
- CA JCLCheck™ Workload Automation (CA JCLCheck)

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. If the topic is long, the actual change may appear on a later page.

Version 17.0, Second Edition

- In-Stream Data (see page 45)— Updated regarding wrapping of a long-name symbolics.
- The BC1PMVCL Utility (see page 69)— Added to describe the utility.
- BSTCOPY and Aliases (see page 71)— Added to describe how BSTCOPY handles aliases.
- The CONDELE Utility (see page 80)— Updated to remove an obsolete statement about footprints.

Version 16.0, Third Edition

- CONPARMX - Dynamically Build a Parameter String (see page 90)— Updated to add a comment about using IGYCRCTL as the program name for which the options parameter string is to be built.
- Other Processors (see page 183)— Updated to remove an obsolete reference.

Version 16.0, Second Edition

- The CONLIST Utility (see page 81)— Updated to add a note to the STORE option about the record format of the output library.

Version 16.0

- CA Endevor SCM Symbols (see page 36)— Updated to add source and target symbols, which begin with &C1S and &C1T. These symbols can be used for the source and target locations of Move or Transfer actions. These symbols are shown in italics.
- The LOADONLY Generate Processor (see page 173)— Updated the sample processor.

Release 15.1

- CA Endevor SCM Symbols (see page 36)—Updated to add the symbols &C1ELMNOSRC, &C1SBASLIB, and &C1TBASLIB.
- Including Processor Utilities in Processor Logic (see page 61)—Updated to include a description of the ENUSSUTL utility.

- CONPARMX Utility (see page 90)—Restyled for clarity, along with its subtopics.
- The ENUSSUTL Utility (see page 126)—Added with its subtopics to describe the utility that creates package backout files for HFS and zFS files.

Version 15.0

- Using Site Symbols in Processors (see page 34)—Updated to change a reference in the example from &C1TYPE to &C1TY.
- Return Codes (see page 66)—Updated to add the return code 8 for the BC1PTMP0 utility.

Contents

Chapter 3: Using Processor Utilities 61

Chapter 1: Introduction

This section contains the following topics:

Processors

CA Endevor SCM allows you to create and maintain processors. Coded using standard JCL, processors instruct CA Endevor SCM to modify, move, verify, delete, or create executable forms of elements. There are three types of processors:

- Generate processors create listings, object modules, and/or load modules.
- Delete processors delete generate processor outputs.
- Move processors move elements from one map location to another.

The term *extended processor* refers to a user-written processor that is defined to CA Endevor SCM. It supplies functionality beyond writing or deleting a member in an output library.

Processor Groups

Processors may be specified in processor groups. A processor group consists of the following:

- One generate, one delete, and one move processor, or any combination thereof. (For example, a non-executable element may require only a move processor in its processor group. You cannot, however, have multiple processors of the same type in a processor group.)
- The default symbolic overrides for the processors' JCL.

Processor groups allow you to handle common variations among the members of a particular type quickly and easily. If you create a single set of processors using symbolic parameters, you can create different processor groups using the same set of processors but containing different default symbolic overrides. For example, your site has applications coded in COBOL and COBOL/370. You could define a single type, COBOL, and have two processor groups, one for COBOL II and another processor group for COBOL/370.

Note: For more information, see Symbolic Parameters (see page 30).

Processors and Element Types

Each element type can have multiple processor groups associated with it, but each element within that type must be associated with only one group. When you define the element type, you define one processor group as the default. When you add an element of that type, CA Endevor SCM automatically assigns the default processor group to the element. You can override this assignment on the Add/Update Elements panel.

Note: For more information about defining types, see the *Administration Guide*. For more information about adding elements, see the *User Guide*.

You add processors to CA Endevor SCM in the same manner as other elements. They must be added to type Process, or they are not executable.

Because CA Endevor SCM processors get translated into load modules, they should not be named the same as any programs you may be executing within a processor. For example, do not name a processor CONWRITE or IEWL.

Processors Invoked by Actions

CA Endevor SCM actions invoke processors as follows:

Add

Invokes the Generate processor and provides the following options:

- Processor group
- Bypass generate processor

Archive

Invokes the Delete processor and provides the Bypass element delete option.

Delete

Invokes the Delete processor and provides the Bypass element delete option.

Generate

Invokes the Generate processor (and Delete processor if processor group changes) and provides the Processor group option.

Move

Invokes the Move (default) or Generate, and Delete processors and provides the Bypass element delete option.

Restore

Invokes the Generate processor and provides the following options:

- Processor group
- Bypass generate processor

Transfer

Invokes the Generate (default) or Move processor, and Delete processor and provides the following options:

- Processor group
- Bypass generate processor
- Bypass delete processor
- Bypass element delete

Update

Invokes the Generate processor and provides the following options:

- Processor group
- Bypass generate processor

The COPY, DISPLAY, LIST, PRINT, RETRIEVE, and SIGNIN actions do not invoke processors.

Note: For more information about processor groups, see the chapter "Using Processor Groups (see page 139)." For more information about how CA Endevor SCM determines which processor group to use for an action, see the *User Guide*.

How CA Endevor SCM Represents Syntax

CA Endevor SCM uses the IBM standard for representing syntax.

Note: For information about syntax, how you code syntax,and sample syntax diagrams, see the *Administration Guide*.

Chapter 2: Writing Processors

This section contains the following topics:

Introduction

Processors are coded using standard JCL syntax and are converted to a CA Endevor SCM executable form. Processor statements specify which programs/utilities are run, the order in which they are run, and any special conditions required. (They are similar to JCL statements.)

CA Endevor SCM provides several keywords, symbolic parameters, and utilities for use in coding processors. While CA Endevor SCM supplies many of its own utilities, user-coded or third-party utilities or programs can be used within a processor as well.

All DD statements allocated for a CA Endevor SCM processor step are de-allocated at processor step termination. This means if a processor step uses a previously allocated DD statement such as SYSPROC, it is allocated for the processor step and subsequently de-allocated at processor step termination.

Note: For an explanation and illustration of the various types and uses of processors, see the appendix "Sample Processors (see page 155)."

Processor Names

Processor names can have up to eight characters. The following table provides abbreviations that do not represent a complete list, and are offered as guidelines only, shown by character position and then description:

Character Position	Description
1	Processor type; for example: ■ G = generate processor ■ D = delete processor ■ M = move processor
2 - 4	Language type or utility; for example: ■ ASM = ASSEMBLER ■ CLI = CLIST ■ CII = COBOL ■ DAT = DATA (documentation) ■ EAS = EASYTRIEVE ■ FOR = FORTRAN ■ JCL = JCL ■ LEC = Link edit control cards ■ LOD = Load modules ■ OBJ = Object modules ■ PLI = PLI ■ RPG = RPG ■ TEL = TELON ■ TRA = TRANSFORM
5	Data base environment; for example: ■ D = DB2/DL1 ■ S = IDMS ■ I = IMS ■ N = None
6	Operating environment; for example: ■ B = Batch ■ C = CICS ■ S = IDMS-DC ■ I = IMS-DC ■ N = None

Character Position	Description
7	Output type: ■ A = Impact Analysis SCL ■ L = Load Module ■ K = NCAL Load Module ■ O = Object ■ P = PDS ■ R = Report(s) ■ N = None
8	Stage ID

Processor Features

For the most part, CA Endevor SCM processors are written using standard OS JCL syntax. CA Endevor SCM supports most JCL parameters.

Note: For a complete list of unsupported JCL parameters, see the appendix Unsupported Parameters (see page 219).

To improve processing throughput, CA Endevor SCM allocates DD statements defined with DISP=OLD as DISP=SHR. During execution, it issues an enqueue (Qname=CTLIPROC) to single thread access to the data set. DISP=OLD requires longer exclusive access than the CA Endevor SCM queue.

If a DD statement or a library needs to run with a true DISP=OLD, you must enable the ENHOPTDD feature in the CA Endevor SCM Options table (ENCOPTBL).

CA Endevor SCM provides the following additional features and capabilities within processors:

- Keywords to tailor your processors
- CA Endevor SCM, User- and Site-defined symbols
- Component monitoring (for CA Endevor SCM Automated Configuration Option clients)
- CA Endevor SCM utilities
- Support for in-stream data

Each of these features is discussed in detail in this section.

Reserved Words and Labels for Processors

When writing a processor, avoid using the following reserved names:

- CC
- DDA
- DDB
- DDB
- JCL
- PGM
- SYM
- SIN
- PRITE
- FLD

Otherwise, your processor will terminate with this message:

```
ASMA043E *** ERROR *** Previously defined symbol xxxx
```

Authorized Program Table

You must add the names of non-CA Endevor SCM programs to the Authorized Program Table (C1GTAPGM), if these programs are invoked within a processor and need to run APF-authorized.

Note: For more information, see Authorized Program Table in the chapter "Logical and Physical Structure" in the *Administration Guide*.

Processor Keywords

The following keywords are specific to CA Endevor SCM processor statements:

FOOTPRNT

Causes CA Endevor SCM footprints to be created or verified.

MAXRC

Specifies the maximum acceptable return code for a job step.

EXECIF

Permits execution of a specific step only if the specified conditions are met.

BACKOUT

Allows you to maintain backout information on a library by library basis, if you are using package processing.

MONITOR

Allows monitoring of input and output components as a part of Configuration Management.

ALTID

ALTID=N enables you to disable the CA Endevor SCM alternate user ID during execution of the processor step.

ALLOC

Indicates whether the DD statement is to be expanded into a concatenation of data sets based on the map.

The FOOTPRNT Keyword

FOOTPRNT can be used in DD statements, to create or verify a CA Endevor SCM footprint, or to bypass footprint creation:

Note: The processor keyword FOOTPRNT cannot be specified on DD statements that refer to USS path and file names.

- **FOOTPRNT=CREATE** footprints a member in an output data set, to associate that member with the element being processed.
- If the output is written by a CA Endevor SCM action, such as RETRIEVE, to a specific PDS (or ELIB) member, or if you are using the CONWRITE or CONLIST utilities, the member is footprinted automatically, regardless of whether you use the FOOTPRNT keyword.
- If you are using BSTCOPY, the output is directed to a PDS by a non-CA Endevor SCM utility (such as a user program), or the output is a sequential object module, you must specify FOOTPRNT=CREATE for the member to be footprinted.

- To footprint a load module, you must specify FOOTPRNT=CREATE on the SYSLMOD DD statement for the load module, shown as follows:

```
//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).,FOOTPRNT=CREATE,
//             MONITOR=&MONITOR.,DISP=SHR
//SYSLIB   DD  DSN=&LSYSLIB1.,
//             MONITOR=&MONITOR.,
//             DISP=SHR
//         DD  DSN=&LSYSLIB2.,
//             MONITOR=&MONITOR.,
//             DISP=SHR
//         DD  DSN=&COBLIB.,
//             DISP=SHR
//SYSUT1   DD  UNIT=&WRKUNIT.,SPACE=(CYL,(1,1))
//SYSPRINT DD  DSN=&&LNKLIST,DISP=(OLD,PASS)
```

  When footprinting, keep in mind that:

  - In order to footprint an object deck, CA Endevor SCM locates the first, named CSECT within the object deck, and uses that name to build a linkage-editor identify control card. If there is no named CSECT in the object deck, the footprinting action fails.
  - All CSECTs within a load module should be footprinted.
  - You cannot footprint object modules that do not contain CSECT names.
  - The library member name must match the name of the element to which it corresponds.

- **FOOTPRNT=VERIFY** verifies the footprint in an existing library member corresponds to the current level of the element being processed. This statement is generally used in the move processor to verify the element against the CA Endevor SCM Master Control File before the element is moved or transferred.

  An error during the footprint verification step causes the job to fail because it is a CA Endevor SCM error. To override this default, the VFY_FP_FAILURES feature must be activated in the CA Endevor SCM Options table (ENCOPTBL). With this option, the footprint verification return code becomes a step return code instead of a CA Endevor SCM return code.

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

```
//STEPNAME  EXEC PGM=IEBCOPY
//SYSPRINT  DD  SYSOUT=*
//IN        DD  DSN=STAGE1.COPYLIB,DISP=SHR,FOOTPRNT=VERIFY
//OUT       DD  DSN=STAGE2.COPYLIB,DISP=SHR
```

- **FOOTPRNT=NONE** tells CA Endevor SCM to bypass footprint creation.

The MAXRC Keyword

MAXRC defines the highest acceptable step return code for a processor. It is coded on the EXEC statement for the corresponding job step and affects only the current element action. The syntax for the MAXRC parameter follows:

```
MAXRC=nnn
```

nnn

The highest acceptable OS return code for the step

If the return code exceeds the value specified:

- The CA Endevor SCM return code is set to 12.
- A processor-failed flag is set for the element within CA Endevor SCM.
- The processor's remaining steps are executed.
- All remaining actions are executed.

Use the Element Master Info display to confirm the processor-failed flag is set for an element. The literal “FAILED” appears next to the processor return code when the flag is set. If the processor-failed flag is set for an element in either stage, CA Endevor SCM does not allow a MOVE or TRANSFER action against the element.

You can bypass executing one or more steps, by coding the COND or EXECIF parameters on the steps or using IF/THEN/ELSE processor logic.

Note: To terminate all action processing, specify a value for STOPRC. For more information, see the *SCL Reference Guide*.

MAXRC Scenario

There is a processor containing three steps:

1. Compile
2. Link-edit
3. CONLIST

You would code the following parameters:

- MAXRC parameter on the compile and link-edit steps
- COND parameter on the link-edit step

Regardless of the return codes in the first two steps, you always want the CONLIST step to execute and store the listings.

MAXRC Example

A COBOL compile with warnings has a return code of 04. To allow warnings but prohibit serious problems, specify MAXRC=04 on the compile EXEC statement. Return codes greater than 04 set the CA Endevor SCM processor-failed flag for the element.

```
//STEPNAME  EXEC PGM=IKFCBL00,MAXRC=04,COND=(0,NE)
```

All remaining steps are executed unless the COND, EXECIF or IF/THEN/ELSE logic specifies otherwise.

The EXECIF Keyword

EXECIF allows you to define conditions under which a processor step is executed. Using the EXECIF keyword, you can create a single processor containing the steps that are conditionally required. You can, therefore, write one processor instead of executing multiple processors.

EXECIF is a truth statement, and in its simplest format its syntax is:

```
EXECIF=(value1,operator,value2)
```

value1

The user specified value for the truth statement. It can be a literal, a CA Endevor SCM, user, or site symbol.

operator

The required condition between value1 and value2. Valid values are as follows:

- EQ-equal to
- GT-greater than
- NE-not equal to
- LE-less than or equal to
- LT-less than
- GE-greater than or equal to

value2

The user specified value for the truth statement. It can be a literal, a CA Endevor SCM symbol, or a user symbol.

The EXECIF statement can contain as many value clauses as required, if you have the appropriate number of parentheses (that is, nested parentheses). For example:

```
EXECIF=((value1,EQ,value1A),(value2,EQ,value2A),(value3,EQ,value3A))
```

Note: When you use multiple clauses in the EXECIF statement, all clauses must be true in order for the step to be executed.

The following example illustrates the use of the EXECIF keyword to allow a single generate processor to be used for both DLI COBOL and IDMS COBOL. The generate data coded is the same for each type of COBOL, but different TRANSLATE steps are required by each in order to execute.

```
//DLI      EXEC PGM=DFHECP1$,COND=(0,NE),
//              EXECIF=(&C1PRGRP.,EQ,DLI),
//              PARM='&PRM1'.
//SYSPRINT   DD DSN=&&TRNLIST,DISP=(OLD,PASS)
//SYSPUNCH   DD DSN=&&TRN,DISP=(NEW,PASS,DELETE),
//              UNIT=SYSDA,SPACE=(CYL,(2,1),RLSE),
//              DCB=(RECFM=FB,LRECL=80,BLKSIZE=4000),
//              MONITOR=COMPONENTS
//SYSIN      DD DSN=&&SRC,DISP=(OLD,PASS)
//*************************************************************
//IDMS     EXEC PGM=IDMSDMLC,COND=(0,NE),MAXRC=04,
//              EXECIF=(&C1PRGRP.,EQ,IDMS),
//              PARM='SDMLC-IDMS-STEP',DBNAME='MINI'
//DBRUNLOG   DD DSN=CA.DBRUNLOG,DISP=SHR
//STEPLIB    DD DSN=CA.LOAD.TEST,DISP=SHR
//           DD DSN=CA.LOAD.CV1,DISP=SHR
//SYSDDL     DD DSN=CA.NQ.DDA01.CV1,DISP=SHR
//SYSIPT     DD DSN=CA.READONLY.DMLC,DISP=SHR
//           DD DSN=&&SRC,DISP=(OLD,PASS)
//SYSJRNL    DD DUMMY
//SYSLST     DD DSN=&&TRNLIST,DISP=(OLD,PASS)
//SYSPCH     DD DSN=&&TRN,DISP=(NEW,PASS,DELETE),
//              UNIT=SYSDA,SPACE=(CYL,(2,1),RLSE),
//              DCB=(RECFM=FB,LRECL=80,BLKSIZE=4000),
```

You can also use the EXECIF statement in generate processors for use with load modules, to make sure that a GENERATE action against the load module does not execute the generate processor.

Note: For more information about load modules, see the *Utilities Guide*.

The BACKOUT Keyword

If you are using package processing, **BACKOUT** allows you to maintain backout information on a library by library basis.

When you use packages, CA Endevor SCM, by default, maintains backout information for PDS members that are created, changed, or deleted during processor execution. If you do not want this information maintained for a library, code BACKOUT=N in the DD statement for the library.

```
//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).,FOOTPRNT=CREATE,BACKOUT=N,
//             MONITOR=&MONITOR.,DISP=SHR
//SYSLIB    DD DSN=&LSYSLIB1.,
//             MONITOR=&MONITOR.,
//             DISP=SHR
//          DD DSN=&LSYSLIB2.,
//             MONITOR=&MONITOR.,
//             DISP=SHR
//          DD DSN=&COBLIB.,
//             DISP=SHR
//SYSUT1    DD UNIT=&WRKUNIT.,SPACE=(CYL,(1,1))
//SYSPRINT  DD DSN=&&LNKLIST,DISP=(OLD,PASS)
```

The MONITOR Keyword

Component monitoring is a feature of the CA Endevor SCM Automated Configuration Manager (ACM). You can use the keyword MONITOR to monitor selected library data sets for component relationships.

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

Note: For more information about component monitoring, see the *Automated Configuration Option Guide*.

The ALTID Keyword

If a CA Endevor SCM alternate ID is defined in the C1DEFLTS table, then a processor step executes under the security context of the CA Endevor SCM alternate ID. However, you can override this behavior by specifying ALTID=N on the EXEC statement of the processor. If ALTID=N is specified, then the processor step runs under the security context of the user ID associated with the job.

If ALTID=N is specified, then CA Endevor SCM does not use the alternate user ID during execution of that processor step, the value of the INTRDR_ALTID option in the CA Endevor SCM Options Table is ignored, and any jobs submitted to the internal reader run under the security context of the user ID.

If you are using CA Top Secret and you want CA Endevor SCM to submit JOBs to the INTRDR using the credentials of the CA Endevor SCM user ID, you must set the CA Top Secret global option JES(VERIFY). If the JES(VERIFY) option is not set, CA Top Secret attempts to build the USER= JOBCARD parameter with the CA Endevor SCM Alternate ID as the USER parameter value. This causes the submitted JOB to fail, because users typically do not have the authority to specify the CA Endevor SCM Alternate ID on their JOBCARD. For more information about the internal reader, see The Internal Reader (see page 60).

The security context used for USS file access in processor steps is determined by the ALTID keyword. The IBM APAR OA39558 allows processors to run under either the credentials of the user, or under the Alternate ID, so that USS outputs (created, copied, compiled, and so on) will have the Alternate ID as the owner or group owner. For more information about USS supported files, see USS Supported Files and the Alternate ID in the *Administration Guide*.

Note: For more information about the Alternate ID, see Alternate ID Support in the *Security Guide*.

The ALLOC Keyword

Use the ALLOC keyword to implement the processor map allocation feature. The ALLOC clause indicates whether a DD statement is to be expanded into a concatenation of data sets based on the map. The allocation of the DD statement for map processing includes all data sets in a concatenation starting from the current location until the end of the map. The following fields are used to determine the map, starting from the CA Endevor SCM location at which the processor is being executed: environment, stage, system, subsystem, and type. If system, subsystem, or type are not defined at any point in the map, map allocation will stop at the last stage where they are all defined.This option simplifies processor maintenance, especially for work area (sandbox) implementations.

For more information, see Processor Map Allocation (see page 26).

The keyword has the following format:

```
ALLOC={LMAP|PMAP|STD}
```

LMAP

Bases the concatenation on the logical map.

PMAP

Bases the concatenation on the physical map.

STD

Standard allocation. No special map processing is done. This is the default when ALLOC is not specified.

Note: Certain symbols must be used to specify location information for a data set name that varies based on the map. These symbols, when specified with ALLOC={LMAP|PMAP}, will vary for each location in the map. For more information, see Implement Processor Map Allocation (see page 27).

Processor Map Allocation

The processor map allocation feature simplifies the administrator's task of building and maintaining processors—especially for private work area (sandbox) implementations.

This feature provides the ability to code a varying concatenation of data sets to a single DD statement based on the map using location (environment, stage, system, subsystem) information provided by symbol within the data set name, eliminating the need to use if-then-else logic. It facilitates in the implementation of private work spaces by eliminating the need for alias data set names when system or subsystem names change across environments.

The use of segregated private work areas is most easily implemented by using a private work area identifier (usually a user ID) as the subsystem name in the lowest environment in the life cycle. However, because the file names for source, copy, macro and header libraries contain the subsystem name, the processors definitions must be defined with many if-then-else statements to accommodate different file name formats. This precludes the use of symbolic variables to build the file names. The processor map allocation enhancement eliminates the need for conditional statements like if-then-else when concatenation allocations vary by life cycle location.

Processor map allocation works for data sets whose names follow the mapping structure of the location from which the processor is executing. Processors will not have to be generated after a change to the mapping structure.

You cannot use the processor map allocation feature for data sets that do not follow the current mapping structure, as in data sets that are from a different CA Endevor SCM implementation. This feature does not allow a concatenation of interspersed data set names based on the map.

Implement Processor Map Allocation

To simplify processors, when system or subsystem names change across environments, you can use the processor map allocation feature, which lets you code a varying concatenation of data sets to a single DD statement.

To implement processor map allocation

1. Write a processor DD statement that includes the ALLOC= clause to indicate whether the DD statement is to be expanded into a concatenation of data sets based on the logical or physical map.

   ALLOC=LMAP

   Bases the concatenation on the logical map.

   ALLOC=PMAP

   Bases the concatenation on the physical map.

   The allocation of the DD statement for map processing includes all data sets in a concatenation starting from the current location until the end of the map.

2. Use the following symbols for the location specifications in the DD statement.

   &C1ENVMNT, &C1EN – Environment

   &C1STAGE, &C1ST – Stage Name

   &C1STGID, &C1SI – Stage ID

   &C1STGNUM, &C1S# – Stage Number

   &C1SYSTEM, &C1SY – System

   &C1SUBSYS, &C1SU – Subsystem

   These symbols, when specified with ALLOC={LMAP|PMAP}, will vary for each location in the map. Other symbols are allowed, but will remain the same for all data sets in the concatenation.

Example--Implement Processor Map Allocation

Assume the logical map to be D1=>D2=>Q2=>P2, with the physical map including Q1 and P1 prior to Q2 and P2, respectively.

Start with the following DD statement in your processor, where &LIB=MACLIB is specified in the processor:

```
SYSLIB DD DSN=BST.MORMI08.&LIB,DISP=SHR
       DD DSN=BST.CSCRE.&C1SY..&C1SU..&C1EN(1,1)&C1S#..&LIB,
          DISP=SHR,ALLOC=LMAP
```

If executing the processor in DEV stage 1, system NDVR, subsystem MORMI08, the logical map allocation would be generated as follows:

```
SYSLIB DD DSN=BST.MORMI08.MACLIB,DISP=SHR
       DD DSN=BST.CSCRE.NDVR.MORMI08.D1.MACLIB,DISP=SHR
       DD DSN=BST.CSCRE.NDVR.MORMI08.D2.MACLIB,DISP=SHR
       DD DSN=BST.CSCRE.NDVR.BASE.Q2.MACLIB,DISP=SHR
       DD DSN=BST.CSCRE.NDVR.BASE.P2.MACLIB,DISP=SHR
```

If executing from QA stage 2:

```
SYSLIB DD DSN=BST.MORMI08.MACLIB,DISP=SHR
       DD DSN=BST.CSCRE.NDVR.BASE.Q2.MACLIB,DISP=SHR
       DD DSN=BST.CSCRE.NDVR.BASE.P2.MACLIB,DISP=SHR
```

If the prior DD's were coded with ALLOC=PMAP, the following physical map allocations would result:

```
SYSLIB DD DSN=BST.MORMI08.MACLIB,DISP=SHR
       DD DSN=BST.CSCRE.NDVR.MORMI08.D1.MACLIB,DISP=SHR
       DD DSN=BST.CSCRE.NDVR.MORMI08.D2.MACLIB,DISP=SHR
       DD DSN=BST.CSCRE.NDVR.BASE.Q1.MACLIB,DISP=SHR
       DD DSN=BST.CSCRE.NDVR.BASE.Q2.MACLIB,DISP=SHR
       DD DSN=BST.CSCRE.NDVR.BASE.P1.MACLIB,DISP=SHR
       DD DSN=BST.CSCRE.NDVR.BASE.P2.MACLIB,DISP=SHR
```

If executing from QA stage 2:

```
SYSLIB DD DSN=BST.MORMI08.MACLIB,DISP=SHR
       DD DSN=BST.CSCRE.NDVR.BASE.Q2.MACLIB,DISP=SHR
       DD DSN=BST.CSCRE.NDVR.BASE.P1.MACLIB,DISP=SHR
       DD DSN=BST.CSCRE.NDVR.BASE.P2.MACLIB,DISP=SHR
```

In addition, if the DD statements were coded as follows, with the subsystem symbol &SUBSYS, and where &SUBSYS=&C1SU.. is specified in the processor, the same respective allocation results as above would apply :

```
SYSLIB DD DSN=BST.MORMI08.&LIB,DISP=SHR
       DD DSN=BST.CSCRE.&C1SY..&SUBSYS&C1EN(1,1)&C1S#..&LIB,
          DISP=SHR,ALLOC={LMAP|PMAP}
```

Example--Implement Processor Map Allocation with Interspersed Data Sets

Assume the logical map to be D1=>D2=>Q2=>P2, with the physical map including Q1 and P1 prior to Q2 and P2, respectively.

What this enhancement will not do is provide the capability of interspersing datasets in a concatenation based on the map.

If executing a processor in DEV stage 1, system NDVR, subsystem MORMI08, with a DD statement, after symbolic substitution, that would be allocated as follows (with alternating NDVR and EA data sets), you would have to make a choice as to which set of data sets needs to be first in the concatenation, if you want to take advantage of map allocation.

```
SYSLIB DD DSN=BST.CSCRE.NDVR.MORMI08.D1.MACLIB,DISP=SHR
       DD DSN=BST.CSCRE.EA.BASE.D1.MACLIB,DISP=SHR
       DD DSN=BST.CSCRE.NDVR.MORMI08.D2.MACLIB,DISP=SHR
       DD DSN=BST.CSCRE.EA.BASE.D2.MACLIB,DISP=SHR
       DD DSN=BST.CSCRE.NDVR.BASE.Q2.MACLIB,DISP=SHR
       DD DSN=BST.CSCRE.EA.BASE.Q2.MACLIB,DISP=SHR
       DD DSN=BST.CSCRE.NDVR.BASE.P2.MACLIB,DISP=SHR
       DD DSN=BST.CSCRE.EA.BASE.P2.MACLIB,DISP=SHR
```

You could code it as follows, using logical map allocation:

```
SYSLIB DD DSN=BST.CSCRE.&C1SY..&C1SU..&C1EN(1,1)&C1S#..MACLIB,
       DISP=SHR,ALLOC=LMAP
       DD DSN=BST.CSCRE.EA.BASE.&C1EN(1,1)&C1S#..MACLIB,
       DISP=SHR,ALLOC=LMAP
```

This would result in the following allocation:

```
SYSLIB DD DSN=BST.CSCRE.NDVR.MORMI08.D1.MACLIB,DISP=SHR
       DD DSN=BST.CSCRE.NDVR.MORMI08.D2.MACLIB,DISP=SHR
       DD DSN=BST.CSCRE.NDVR.BASE.Q2.MACLIB,DISP=SHR
       DD DSN=BST.CSCRE.NDVR.BASE.P2.MACLIB,DISP=SHR
       DD DSN=BST.CSCRE.EA.BASE.D1.MACLIB,DISP=SHR
       DD DSN=BST.CSCRE.EA.BASE.D2.MACLIB,DISP=SHR
       DD DSN=BST.CSCRE.EA.BASE.Q2.MACLIB,DISP=SHR
       DD DSN=BST.CSCRE.EA.BASE.P2.MACLIB,DISP=SHR
```

If this would not work for your situation, then you would not be able to take advantage of processor map allocation.

Symbolic Parameters

Symbolic parameters (symbols) provide a powerful means of writing processors for your environment. The processors treat the symbols in the same manner as OS JCL treats parameters. A specific value is substituted for a variable embedded in the JCL. This allows you to write one processor for multiple environments, systems, subsystem, or stage. You can write one processor to perform many functions.

There are three types of symbols within CA Endevor SCM:

- User symbols
- Site symbols
- CA Endevor SCM symbols

In addition, CA Endevor SCM provides capabilities not available with standard JCL. For example:

- You can use a symbol as the numeric operand (portion) in the COND parameter of an execution statement.
- CA Endevor SCM supports symbolic substitution, using CA Endevor SCM, site, or user symbols, within in-stream data.
- Either operand in an EXECIF statement can be a symbol.

The Ampersand (&) Character

Symbols can be used to tailor processors. Symbols are recognized by an ampersand (**&**) as the first character of the value. Any value that appears on the right side of a keyword parameter can be represented by a symbol. Examples follow:

- To define an execution parameter, for example: **PARM='&execparm'**
- To define a data set name, for example: **DSN=&dsname..LOADLIB**

  If you want a period to follow a CA Endevor SCM symbol, you must specify two consecutive periods, if your symbol uses a single ampersand. For example, assume you code the following and CA is the value substituted for &dsname.

  ```
  DSN = &dsname..LOADLIB
  ```

  Then the result is as follows:

  ```
  DSN = CA.LOADLIB
  ```

  Note: If your symbol uses double ampersands, then you would specify only one period.

The following considerations apply:

- When a statement containing a symbol requires a period after the symbol, the symbol itself must end with one of the following:

  ```
  . , / ' ) ( * & + - = " or space
  ```

- When using a symbol for a data set, do not include the member name as part of the symbol or you will get an allocation error. Use two symbols, for example:

  ```
  //SYSLIB DD DSN=&LIB1(&C1ELEMENT)
  ```

- When using concatenated symbols to create a symbol, use an intermediate symbol instead or you will get a translation error. For example, if you combine the symbols into a single symbol as shown in the following sample, the result is an undefined symbol error for the &H&C1SY:

  ```
  DSN=&H&C1SY..&L1TSTL
  ```

  However, the following sample does not result in an undefined symbol error, provided that DISTLQ='&H&C1SY..' is how the intermediate symbol is defined:

  ```
  DSN=&DISTLQ..&L1TSTL
  ```

Guidelines for Symbols

When using symbols in CA Endevor SCM processors, remember that:

- The first character of a site symbol name must be a #.
- When referencing any symbol, an ampersand (&) must be appended to the beginning of the symbol name.
- A user symbol name can consist of any character, except for any of the following terminating characters: . , / ' ( ) * + - = " or space
- User-defined symbolic parameters must be defined in PROC statements at the beginning of the processor.
- You can specify symbols only on the right side of keyword parameters.
- For data set allocation, one symbol must replace only one subparameter. For example, to use symbolic substitution in the statement SPACE=(TRK,(1,1),RLSE), you might code SPACE=(&UNITS,(&PRIM,&SECD),RLSE). It would be incorrect to code SPACE=(&TRAK,RLSE), because this statement merges three subparameters (TRK, 1 and 1) into the single symbol &TRAK.

  Note: In this case, (1,1) is considered to be two subparameters. Thus, defining a single symbol to be '1,1' and substituting that symbol in the SPACE statement results in an allocation error.

- Processor symbols are not validated until execution time. Therefore, ensure that user symbols have valid default values in the processor or valid override values in the processor group definition.

- When a character string contains symbols, the symbol names must be terminated by a terminating character. The terminating character is not considered part of the symbol name. It is just the character at which CA Endevor SCM stops when searching for symbol names in the string.

  Valid terminating characters are:

  - space ' '
  - period '.'
  - comma ','
  - forward slash '/'
  - quote '''
  - double quote '"'
  - close parenthesis ')'
  - open parenthesis '('
  - asterisk '*'
  - ampersand '&'
  - plus '+'
  - minus '-'
  - equal '='
  - less than '<'
  - greater than '>'
  - semi colon ';'

Example: Symbolic Name Terminating Character in Character String

In this example the PARM specifies a character string containing symbolics.

If the symbolics are coded as follows, then during resolution CA Endevor SCM will recognize the symbolic name &ABC because the name is terminated by a "<". CA Endevor SCM will recognize the symbolic name &DEF because it is terminated by a quote character """. The resolved character string will be: UVW<ACTIONXYZ

```
//TEST PROC ABC=UVW,
// DEF=XYZ
//STEP EXEC PGM=IEFBR14,PARM='&ABC<ACTION&DEF'
```

If &ABC is not terminated by a terminating character, it would not be recognized as a symbolic. The following coding would resolve the character string to: &ABCAACTIONXYZ

```
//TEST PROC ABC=UVW,
// DEF=XYZ
//STEP EXEC PGM=IEFBR14,PARM='&ABCAACTION&DEF'
```

User Symbols

User symbols are specific to your site and requirements. As mentioned above, these symbols must be defined in PROC statements that may be specified in the processor.

Processors treat user symbols in the same manner as does standard JCL-as a specific value is substituted for a variable embedded in the JCL code. You can override the symbol default values initially established in the PROC statement. The user symbol override function is part of the processor group definition procedure.

Note: For more information, see The Processor Group Definition Panel (see page 147).

Example (PROC statement)

```
//**
//GCIIDBL PROC PRM1='(XOPTS(DLI,COBOL2,NOSOURCE))',
//            PRM2='(DYN,DATA(24))',
//            PRM3='XREF,AMODE=31,RMODE=ANY,RENT,SIZE=(256K,128K)',
//            PRM4=STORE,
//            STG1='CA.STG1.DEMO',
//            STG2='CA.STG2.DEMO'
//**
```

By changing the parameters established for a particular processor, you can use that processor to perform different functions. The above values are used whenever the processor GCIIDBL is used.

Site Symbols

Site symbols can be used wherever CA Endevor SCM symbols are used. At execution time, site symbols referenced by a processor are stored with the processor symbols in the component data. If a site symbol is also specified as a processor symbol, the processor symbol (and the processor symbol override) take precedence.

When CA Endevor SCM is initialized, the site symbols are placed into memory and when CA Endevor SCM is terminated, the site symbol storage is released.

To implement site symbols, the symbol and its data value are defined in a table that is assembled and linked into an authorized load library. Once this is complete, the table is associated with the appropriate environment by updating SYMBOLTBL parameter in the C1DEFLTS table with the name of the site symbols table. These actions are described in the following sections.

Note: Site symbols are required if you are using USS path name specifications for element type base or source output file definitions.

Using Site Symbols in Processors

Site symbols, like other symbols, are preceded by an ampersand (&) when they are referenced in a processor. The site symbol **#VENDORLIB** looks like this in a processor: &#VENDORLIB

Example: Using Site Symbols in Processors

Assume that your site uses the following site symbols and values:

- #ENDPRE— CA
- #MIDDLE— 'QA&C1SY..S1'
- #SITESYM— 'QA&&#ENDPRE..S1'

This is how your site symbols are coded in your processor:

```
//IN      DD DISP=SHR,DSN=&#ENDPRE..R40.SRCLIB
//OUT     DD DISP=SHR,DSN=CA.&#MIDDLE..OBJLIB
//DUMMY   DD DISP=SHR,DSN=&#ENDPRE..&#MIDDLE..&C1ELTYPE
```

Then at runtime, the following values would be used:

```
//IN      DD DISP=SHR,DSN=CA.R40.SRCLIB
//OUT     DD DISP=SHR,DSN=CA.QAc1system.S1.OBJLIB
//DUMMY   DD DISP=SHR,DSN=CA.QAc1system.S1.c1eltype
```

If the current system value is ADMIN and the current type value is ASMMAC at runtime, then the following values would be used:

```
//IN      DD DISP=SHR,DSN=CA.R40.SRCLIB
//OUT     DD DISP=SHR,DSN=CA.QAADMIN.S1.OBJLIB
//DUMMY   DD DISP=SHR,DSN=CA.QAADMIN.S1.ASMMAC
```

Define Site Symbolics

Before defining the site symbols, remember:

- Site symbol names must begin with “#".
- Site symbol names can contain up to twelve characters, including the "#".
- The symbol value may be up to seventy characters long.
- Site symbols are referenced with an ampersand preceding the symbol name. For example, site symbol #VENDORLB is referenced as:

  ```
  &#VENDORLB.
  ```

To define a site symbols table

1. Define the symbol and its data value. This is the syntax for the site symbols table:

   ```
   $ESYMBOL SYMNAME=#symbolname,SYMDATA=symbolvalue
   ```

   symbolname

   The symbol name is 2 to 12 characters long and its first character must be a #. The # indicates that the symbol is defined in the site-defined symbols table.

   symbolvalue

   The data value associated with the site symbol is 1 to 70 characters in length, with no restrictions on the content of the data. If you do not specify a data value for a symbol, CA Endevor SCM treats it as a null variable.

2. Assemble and link-edit the symbols table. Use JCL member BC1JTABL in *iprfx.iqual.CSIQJCL* to assemble and link-edit the site defined symbols table. An alternative is to use an SMP/E USERMOD to accomplish this.

3. Update C1DEFLTS to associate the site symbols to the CA Endevor SCM environment.

   By using the Site Information panel, you can determine if site symbols are defined in your CA Endevor SCM Environment.

   Note: A site symbol may contain references to CA Endevor SCM symbols, or other site symbols. In such cases, the value of SYMDATA contains the name of the CA Endevor SCM or site symbols preceded by a double ampersand (&&). For example,

   ```
   $ESYMBOL SYMNAME=#H,SYMDATA='BST.CSCRE'
   $ESYMBOL SYMNAME=#BASEPRFX,SYMDATA='&&#H..&&C1SY..&&C1EN(1,1)&&C1S#.'
   ```

   The Site Information from C1DEFLTS panel indicates if site symbols are installed. The name of the site symbols file is displayed in the SYMBOLICS Table field when symbols are used.

   Note: To display your site symbols and the associated values, from the CA Endevor SCM Primary Options Menu select the option for Display. Select **S** to display the site symbols and values defined in your CA Endevor SCM environment.

CA Endevor SCM Symbols

CA Endevor SCM symbol names are reserved, and represent values specific to CA Endevor SCM, such as environment and element name. CA Endevor SCM determines the value to be substituted at execution time. For example, if you code the CA Endevor SCM symbol **&C1STGID1** in a processor and add an element, CA Endevor SCM substitutes the appropriate Stage 1 ID when the processor executes. For compatibility with previous releases, CA Endevor SCM symbols can begin with one or two ampersands: & or &&.

When CA Endevor SCM encounters a CA Endevor SCM symbol immediately followed by a period, it takes one of the following actions:

- If the symbol begins with a single ampersand (&), CA Endevor SCM does not include the period in the resolved statement.
- If the symbol begins with a double ampersand (&&), CA Endevor SCM retains the period in the resolved statement.

For example, assume the symbol &C1SYSTEM is set to TEST, and the following DD statements are encountered in the processor:

```
//DD1 DD DSN=&C1SYSTEM..TEST.DATASET,DISP=OLD
//DD2 DD DSN=&&C1SYSTEM..PROD.DATASET,DISP=OLD
```

CA Endevor SCM resolves these statements as:

```
//DD1 DD DSN=TEST.TEST.DATASET,DISP=OLD
//DD2 DD DSN=TEST..PROD.DATASET,DISP=OLD
```

As this example shows, if you want a period to follow a CA Endevor SCM symbol, you must specify two consecutive periods if your symbol uses a single ampersand, and only one period if your symbol uses double ampersands.

The symbolic parameters are listed next along with a description of what the symbol is replaced by in the processor. The numbers in parenthesis following each symbol name indicate the maximum length of the value to be substituted at execution time.

Note: The symbols in italics beginning with &C1S in the following list should be used only for the *source* location of Move or Transfer actions made using the move/generate processor. The symbols in italics beginning with &C1T in the following list should be used only for the *target* location of Move or Transfer actions made using the move/generate processor. For all other actions that execute a generate processor and actions that execute a delete processor, CA Endevor SCM assigns these symbols the same values as those assigned to the corresponding symbol beginning with &C1.

&C1ACTION (8)

Action currently being executed.

Note: If action is ADD with update option turned on and element exists on store, the C1ACTION resolves to UPDATE.

&C1ADDMMMYY (7)

Date in the format DDMMMYY. For example: 15JUN01

&C1ADDMMYY (6)

Date in the format DDMMYY. For example: 150601

&C1ADDMMYYYY (8)

Date in the format DDMMYYYY. For example: 15062001

&C1ADD (2)

Date in the format DD. For example: 15

&C1AMM (2)

Date in the format MM. For example: 06

&C1AMMM (3)

Date in the format MMM. For example: JUN

&C1AYY (2)

Date in the format YY. For example: 01

&C1AYYYY (4)

Date in the format YYYY. For example: 2001

&C1AHHMMSS (6)

Time in the format HHMMSS. For example: 125930

&C1AHHMM (4)

Time in the format HHMM. For example: 1259

&C1AHH (2)

Time in the format HH. For example: 12

&C1ATMM (2)

Time in the format MM. For example: 59

&C1ASS (2)

Time in the format SS. For example: 30

&C1BASELIB (44)

Contains the resolved base library name definition for the current element.

***&C1SBASLIB* (44)**

Base data set name on the Type definition for the action source location. Any embedded symbolic variables are resolved before the action is performed on the element by source management.

***&C1TBASLIB* (44)**

Base data set name on the Type definition for the action target location. Any embedded symbolic variables are resolved before any action is performed on the element by source management.

&C1CCID (12)

Last-specified CCID for the element.

&C1COMMENT (&C1COM) (40)

Comment associated with the action being executed.

&C1ELEMENT (*&C1TELEMENT*) (8)

Up to eight character name of the element being processed. Names longer than eight characters are shortened.

&C1ELMCHG (1)

One character used to indicate whether the source manager detected changes to this element; Y=Yes, N=No.

&C1ELMNOSRC (1)

Indicates whether this is a NoSource element.Y = Yes, the element is sourceless. N=No, the element has source.

***&C1SELMNOSRC* (1)**

One character indicating whether the element is Nosource at the source location prior to being executed by a Move or Transfer action. Y = Yes, the element is sourceless. N=No, the element has source. The symbol is resolved before the action is performed on the element by source management.

***&C1TELMNOSRC* (1)**

One character indicating whether the element is Nosource at the target location prior to being executed by a Move or Transfer action. Y = Yes, the element is sourceless. N=No, the element has source. The symbol is resolved before the action is performed on the element by source management.

&C1ELMNT10 (*&C1TELMNT10*) (10)

1- to 10-character name of the element being processed. Generally used to assign a name to a CA Panvalet member.

&C1ELMNT255 (*&C1TELMNT255*) (255)

1- to 255-character name for element names greater than 10 characters and mixed/lower case names less than 10 characters.

Note: When using this symbol within a processor, consider using the processor symbolic substringing feature. For example, &C1ELMNT255(7,50) or &C1TELMNT255(5,200)

&C1ELTYPE (*&C1TELTYPE*) (&C1TY) (8)

Type associated with the element being processed.

&C1ENVMNT (*&C1TENVMNT*) (&C1EN) (8)

Name of the current environment.

&C1EUDATA (80)

Up to 80 bytes of user data.

&C1FOOTPRT (64)

Footprint of the element being processed.

&C1LEV (2)

Level number for the element being processed.

***&C1SLEV* (2)**

Element level at the source location prior to being executed by a Move or Transfer action.

***&C1TLEV* (2)**

Element level at the target location prior to being executed by a Move or Transfer action.

&C1MACLIB (44)

Value of MACDSN defined in the C1DEFLTS table.

&C1PKGID (16)

Name of the package being executed. Blank if no package is being executed.

&C1PRGRP (*&C1TPRGRP*) (8)

Name of the current processor group. This symbol can be used as part of the source output library or include library data set name specifications.

Note: &C1PRGRP cannot be used as part of the base or delta library data set specification.

&C1PRTYPE (8)

Function of the current processor (generate, move, or delete).

&C1SITE (1)

Current site ID.

&C1STAGE (*&C1TSTAGE*) (&C1ST) (8)

Name of the current stage.

&C1STAGE1 (*&C1TSTAGE1*) (&C1ST1) (8)

Stage 1 name. This symbol must be used for the target stage name for a MOVE action.

&C1STAGE2 (*&C1TSTAGE2*) (&C1ST2) (8)

Stage 2 name. This symbol must be used for the target stage name for a MOVE action.

&C1STGID (*&C1TSTGID*) (&C1SI) (1)

ID of the current stage.

&C1STGID1 (*&C1TSTGID1*) (&C1SI1) (1)

Stage 1 ID. This symbol must be used for the target stage ID for a MOVE action.

&C1STGID2 (*&C1TSTGID2*) (&C1SI2) (1)

Stage 2 ID. This symbol must be used for the target stage ID for a MOVE action.

&C1STGNUM (*&C1TSTGNUM*) (&C1S#) (1)

Number of the current stage.

&C1SUBSYS (*&C1TSUBSYS*) (&C1SU) (8)

Name of the current subsystem.

&C1SYSID (8)

System ID of the current system the processor is running on.

&C1SYSTEM (*&C1TSYSTEM*) (&C1SY) (8)

Name of the current system.

***&C1SARCFIL* (1)**

Indicates if the source is an archive file. The value can be Y or N.

***&C1SELEMENT* (8)**

Element name at the source of MOVE or TRANSFER actions that execute a move/generate processor.

***&C1SELMNT10* (10)**

CA Panvalet element name at the source of MOVE or TRANSFER actions that execute a move/generate processor.

***&C1SELMNT255* (255)**

1-255 character name for elements at the source location of actions such as move and transfer.

Note: When using this symbol within a processor, consider using the processor symbolic substringing feature. For example, &C1SELMNT255 (10,100).

***&C1SELTYPE* (8)**

Element type at the source of MOVE or TRANSFER actions that execute a move/generate processor.

***&C1SENVMNT* (8)**

Environment name at the source of MOVE or TRANSFER actions that execute a move/generate processor.

&C1SPRGRP (8)

Processor group name at the source of MOVE or TRANSFER actions that execute a move/generate processor.

***&C1SSTAGE* (8)**

Stage name at the source of MOVE or TRANSFER actions that execute a move/generate processor.

***&C1SSTAGE1* (8)**

Stage 1 name at the source of MOVE or TRANSFER actions that execute a move/generate processor.

***&C1SSTAGE2* (8)**

Stage 2 name at the source of MOVE or TRANSFER actions that execute a move/generate processor.

***&C1SSTGID* (1)**

Stage ID at the source of MOVE or TRANSFER actions that execute a move/generate processor.

***&C1SSTGID1* (1)**

Stage 1 ID at the source of MOVE or TRANSFER actions that execute a move/generate processor.

***&C1SSTGID2* (1)**

Stage 2 ID at the source of MOVE or TRANSFER actions that execute a move/generate processor.

***&C1SSTGNUM* (1)**

Stage number at the source of MOVE or TRANSFER actions that execute a move/generate processor.

***&C1SSUBSYS* (8)**

Subsystem name at the source of MOVE or TRANSFER actions that execute a move/generate processor.

***&C1SSYSTEM* (8)**

System name at the source of MOVE or TRANSFER actions that execute a move/generate processor.

&C1USERID (8)

User ID associated with the current action.

&C1USRDSN (44)

Data set name of the source for ADD or UPDATE requests.

&C1USRFILE (255)

Source USS file names for the ADD or UPDATE requests.

Note: When using this symbol within a processor, consider using the processor symbolic substringing feature. For example, &C1USRFILE (5,150).

&C1USRMBR (10)

Source member name for the ADD or UPDATE requests.

&C1USRPATH (768)

USS path name of the source for the ADD or UPDATE requests.

Note: When using this symbol within a processor, consider using the processor symbolic substringing feature. For example, &C1USRPATH(720,48).

&C1VER (2)

Version number of the element being processed.

***&C1SVER* (2)**

Element version at the source location prior to being executed by a Move or Transfer action.

***&C1TVER* (2)**

Element version at the target location prior to being executed by a Move or Transfer action.

&C1XLANG (8)

External language for the element type.

Note: The &&C1VIOUNIT variable is taken from the VIOUNIT value in the C1DEFLTS table and used throughout the CA Endevor SCM process. This value is only obtained by the initialization program BC1PINIT and set for use throughout the product.

Substringing

CA Endevor SCM allows you to substring symbolic variables. Substringing allows you to use portions of the symbol value in the processor. A substring expression is identified as follows:

- &SYMBOLIC(start,length,pad)

In this expression:

start

The position within the symbol where substringing is to begin.

Default: 1.

length

Indicates the length of the substituted value.

Default: Length of the symbol.

pad

A single character used for padding.

Default: Blank.

If the start length is greater than the maximum symbol length, CA Endevor SCM does not perform the substring substitution. If the length of the substring exceeds the length of the symbol, CA Endevor SCM substitutes the specified characters and the remainder of the substring is filled with the pad character.

For example, your data set names are prefixed with a special two-character code denoting your department. When you use the data set name in a processor, however, you want to exclude the department identification. Therefore, you might code a statement similar to the following:

- DSN=&dsname(3,3).LOADLIB

In this example, **(3,3)** indicates that you want to begin with the third character of the substitution value and you want the substituted entry to consist of the third, fourth, and fifth characters of the substituted value.

In this example, the data set name to be used for *&dsname* is **APNDVR01**. Using the above substring statement, the characters **AP** are ignored and the substituted value begins with the third character (**N**). Because you have indicated a length of 3, CA Endevor SCM substitutes only three characters, **N**, **D,** and **V**. The result of the substitution is:

- DSN=**NDV**.LOADLIB
- If you indicate a substring specification of (3,5) in this example, **NDVR0** is substituted.

  If you indicate a length of more characters than exist in the substitution value, the resulting field is blank-filled to meet the indicated number of spaces. For example, if you specify (3,7), the substituted value is **NDVR01_** (where "_" indicates a blank).
- If you want to designate a specific character, rather than blanks, to pad the substituted value, simply include that character in the substring. In the above example, assume you want to use dollar signs ($) to pad the substituted value. This particular substring is coded as (3,7,$), and the substituted value is **NDVR01$**.

You can indicate the substring start position, length, and pad character as symbolic parameters. For example, you may assign:

- **&S** to indicate the starting position of the substituted value.
- **&L** to indicate the length of the substituted value.
- **&P** to indicate the character to be used for padding.

In this situation, the statement in our example would be coded as:

- DSN = &dsname(&S,&L,&P).LOADLIB

When you use several symbolic parameters (nested symbols), CA Endevor SCM begins substitution with the innermost parentheses. The following example illustrates the principles discussed in this section:

- DSN = &LIB(&C1ELEMENT(&S,&L,&P))

where:

- &LIB = CA.LIB
- &C1ELEMENT = ABC
- &S = 1
- &L = 8
- &P = $

CA Endevor SCM resolves this statement by substituting the values for the following:

- the starting position (&S), length (&L), and padding character (&P):

  DSN = &LIB(&C1ELEMENT(1,8,$))
- &C1ELEMENT:

  DSN = &LIB(ABC$$$$)
- &LIB:

  DSN = CA.LIB

When CA Endevor SCM finishes resolving the statement, the result is:

DSN = CA.LIB(ABC$$$$$)

Note: When using a substring within an IF statement in a processor, you must put a set of single quotes around the substringing part of the statement. For example:

```
IF(('&C1ELEMENT(1,3)'='ACC'))
```

In-Stream Data

CA Endevor SCM supports in-stream data (DD*/DD DATA), which are coded according to JCL syntax.

Columns 73 through 80 are unpredictable within processor data. If any symbolic parameters are present, substitution occurs only between columns 1 and 72 of the input statement. Each line is truncated or padded to 72 characters.

To turn on wrapping for a symbolic that is longer than 72 characters (a long-name symbolic), the processor heading must include the symbolic C1WRAP set to Y. If C1WRAP is set to anything else, or is absent, no wrapping will occur.

When wrapping is turned on by C1WRAP, long-name symbolics are split on several lines in the following ways:

- If there is a continuation character in column 72 and numeric characters in columns 73 through 80, substitution occurs without any split to preserve the behavior when continuation is coded.
- If there are blank characters in columns 72 through 80 and a long-name symbolic is used, substitution occurs and the line is split on several lines with a length of 72. If there are spaces in the substituted line, CA Endevor SCM will try to split it on the space character if that is possible.

If SCL syntax is used in the in-stream data, CA Endevor SCM will join lines without space characters if the split is in the middle of a name in apostrophes. Therefore if the name contains spaces, split it manually with a substring to keep the spaces in the correct positions.

BPXBATCH uses a different approach to joining more lines in the in-stream data. It will join them with an additional space character between them. Because of this, you need to edit this string to remove those space characters to get the original string (if it did not contain spaces).

Example: BPXBATCH and wrapping of in-stream data

Assume the following BPXBATCH example:

```
SH echo '/tmp/&C1ELMNT255' | sed 's/\ //g'| xargs ls -l ;
```

The whole name is sent to SED program, which substitutes all space characters with empty characters. After that, the substituted name is sent to XARGS which constructs an argument list and run a command (for example, in this case it is ls - l).

Controlling Processor Flow Using the IF-THEN-ELSE Statement

The CA Endevor SCM IF-THEN-ELSE JCL statement provides control of the order SCL statements are processed. The IF-THEN-ELSE statement is derived from the IBM OS JCL statement and provides the following functionality:

- Control over the execution of job steps.
- Use of symbolic variables and IF-THEN-ELSE statements to control the inclusion (or exclusion) of complete DD statements and input in-stream data.

At run time, the IF-THEN-ELSE statement is evaluated and then the appropriate statements are selected for inclusion. Consequently, the processor JCL can be customized at each run without causing intervening modifications to the processor.

The CA Endevor SCM IF-THEN-ELSE statement is similar to the COND and EXECIF keywords function. However, the IF-THEN-ELSE statement provides the following advantages over the COND and EXECIF keywords:

- Control for multiple processor steps.
- Control data set inclusion
- One-time coding.
- A larger selection of conditional choices

COND and EXECIF keywords must be coded on each applicable EXEC statement.

CA Endevor SCM allows selection of steps and DD statements using not only condition codes from prior steps but also values of CA Endevor SCM symbols and processor symbols. This allows the customer more control of the processor execution.

The basic syntax of IF-THEN-ELSE follows the syntax of most JCL statements, but also has some idiosyncrasies. Columns 1 and 2 must contain slashes "//". An optional name may be placed in Column 3, can be up to eight characters long, and must be followed by a blank.

Note: It is highly recommended you specify a name. This helps in the debugging process if unexpected results occur.

The IF statement must be present followed by a conditional statement and is concluded with the THEN statement. The conditional statement may be enclosed in parentheses to indicate nesting. An ELSE statement is coded similarly to the IF statement excluding the conditional statement. The IF block is completed by the ENDIF.

Syntax example

```
//TEST1      IF keyword operator value THEN
//
.
.
.
statements
//ELSE1    ELSE
//
.
.
.
statements
//ENDIF1 ENDIF
```

keyword

Valid keywords are:

- CA Endevor SCM symbolics
- IBM defined expressions
  - RC
  - ABENDCC
  - ABEND
  - ¬ABEND
  - RUN
  - ¬RUN

operator

Valid operators are:

- EQ or =
- GT or >
- NE or ¬=
- LE
- LT or <
- GE

Note: For more information about these operators, see The EXECIF Keyword (see page 22).

value

Values are:

- 1- to 16-characters in length
- Alphanumeric
- Numerics. Numeric values must always be enclosed in quotes. There is one exception to this rule, if the numeric value is compared to the RC keyword, quotes are not required.

statements

Represents any valid JCL statements inserted between the IF-THEN-ELSE statements.

These statements can be EXEC or DD statements.

The entire statement must be coded within the IF-THEN-ELSE structure. If a JCL statement is continued on multiple lines, it must be completed before an intervening IF-THEN-ELSE is encountered.

Example: IF-THEN-ELSE Statement (example 1)

This example is valid.

```
//* Valid example
//IF    IF   (&STAGE=PRD) THEN
//DD1   DD   DISP=(,CATLG),UNIT=SYSDA,
//      VOL=SER=VOLUME,DSN=ABC.DEF
//      ENDIF
```

The following example is not valid.

```
//* Invalid example
//IF   IF  (&STAGE=PRD) THEN
//DD1  DD  DISP=(,CATLG),UNIT=SYSDA,
//     ENDIF
//     VOL=SER=VOLUME,DSN=ABC.DEF
```

Example: IF-THEN-ELSE Statement (example 2)

This example is valid.

```
//* Valid example
//SYSABEND   DD DUMMY
//           IF (&CDEBUGD='D') THEN
//SYSLIB     DD DSN=ABC.DEF1,DISP=SHR,MONITOR=COMPONENTS
//           DD DSN=BCD.DEF2,DISP=SHR,MONITOR=COMPONENTS
//           DD DSN=CDE.DEF3,DISP=SHR,MONITOR=COMPONENTS
//           DD DSN=IJK.DEFADD,DISP=SHR,MONITOR=COMPONENTS
//           ELSE
//SYSLIB     DD DSN=EFG.DEF4,DISP=SHR,MONITOR=COMPONENTS
//           DD DSN=FGH.DEF5,DISP=SHR,MONITOR=COMPONENTS
//           DD DSN=HGI.DEF6,DISP=SHR,MONITOR=COMPONENTS
//           DD DSN=IJK.DEFADD,DISP=SHR,MONITOR=COMPONENTS
//           ENDIF
```

The following example is not valid.

```
//*Invalid example
//SYSABEND   DD DUMMY
//           IF (&CDEBUGD='D') THEN
//SYSLIB     DD DSN=ABC.DEF1,DISP=SHR,MONITOR=COMPONENTS
//           DD DSN=BCD.DEF2,DISP=SHR,MONITOR=COMPONENTS
//           DD DSN=CDE.DEF3,DISP=SHR,MONITOR=COMPONENTS
//           ELSE
//SYSLIB     DD DSN=EFG.DEF4,DISP=SHR,MONITOR=COMPONENTS
//           DD DSN=FGH.DEF5,DISP=SHR,MONITOR=COMPONENTS
//           DD DSN=HGI.DEF6,DISP=SHR,MONITOR=COMPONENTS
//           ENDIF
//           DD DSN=IJK.DEFADD,DISP=SHR,MONITOR=COMPONENTS
```

An IF clause may be coded at three points in the processor JCL:

1. Prior to EXEC statements and all of their associated DD statements.

   The IF clause is coded here when a test is to be made to determine if one or more complete steps within an IF-THEN-ELSE block are to be either included or excluded from the processor execution.

```
//STEP1 EXEC PGM=COBOL1
//  IF (STEP1.RC=0)
//    THEN
//STEP2 EXEC PGM=LINK
//DD1   DD DISP=SHR,DSN=INPUT
//DD2   DD DISP=SHR,DSN=OUTPUT
//STEP3 EXEC PGM=PRINT
//DD1   DD DISP=SHR,DSN=INPUT
//DD2   DD DISP=SHR,DSN=OUTPUT
//    ELSE
//STEP4 EXEC PGM=PRINT
//DD1   DD DISP=SHR,DSN=INPUT
//DD2   DD DISP=SHR,DSN=OUTPUT
//    ENDIF
//STEP5 EXEC PGM=COPY
```

   If the return code of STEP1 is equal to 0, STEP2 and STEP3 are executed. STEP4 is not executed.

2. Prior to DD statements and any DD statements that are concatenated to them.

   The IF clause is coded here when a test is to be made to determine if one or more complete DD statements within an IF-THEN-ELSE block are to be either included or excluded from the processor step definition.

```
//STEP1 EXEC PGM=COBOL1
//STEP2 EXEC PGM=LINK
//DD1   DD   DISP=SHR,DSN=INPUT1
//  IF (&STAGE=PRD).
//    THEN
//DD2   DD   DISP=SHR,DSN=INPUT2
//DD3   DD   DISP=SHR,DSN=INPUT3
//    ELSE
//DD4   DD   DISP=SHR,DSN=INPUT4
//    ENDIF
//STEP3 EXEC PGM=PRINT
```

   If &STAGE is equal to PRD when STEP2 is executed, then DD1, DD2, and DD3 are included in the step definition. DD4 is not included in the step definition.

3. Prior to a DD statement that is part of a DD concatenation.

   The IF clause is coded here when a test is to be made to determine if one or more statements that are part of a DD concatenation and within an ITE (IF-THEN-ELSE) block are to be either included or excluded from the processor step definition.

```
//STEP1 EXEC PGM=COBOL1
//STEP2 EXEC PGM=LINK
//DD1   DD  DISP=SHR,DSN=INPUT1
// IF (&STAGE=PRD).
//   THEN
//      DD  DISP=SHR,DSN=INPUT2
//      DD  DISP=SHR,DSN=INPUT3
//   ELSE
//      DD  DISP=SHR,DSN=INPUT4
//   ENDIF
//DD2   DD  DISP=SHR,DSN=INPUT5
//DD3   DD  DISP=SHR,DSN=INPUT6
//STEP3 EXEC PGM=PRINT
```

   If &STAGE is equal to PRD when STEP2 is executed, datasets INPUT1, INPUT2, INPUT3 are included in the dataset concatenation defined by DD1. The INPUT4 dataset is not included.

   An IF-THEN-ELSE IF specification may extend over multiple statements. The specification must be within columns 1 - 71 of each statement. Column 72 may contain an X to indicate a continuation of the line, but it is not mandatory.

Example (An IF statement extending over multiple statements, without an 'X' in column 72)

```
CC                                                  CC
1                                                   72
//TESTIF    IF  ((&STG. EQ PRD1) OR (&STG. EQ PRD2)
//          OR   (&STG. EQ PRD3) OR (&STG. EQ PRD4)
//          OR   (&STG. EQ PRD5)) THEN
```

Example (An IF statement extending over multiple statements, with an 'X' in column 72)

```
CC                                                  CC
1                                                   72
//TESTIF   IF   ((&STG. EQ PRD1)                    X
//         OR   (&STG. EQ PRD2)                     X
//         OR   (&STG. EQ PRD3)                     X
//         OR   (&STG. EQ PRD4)                     X
//         OR (&STG. EQ PRD5)) THEN
```

The THEN clause may be on the same statement as the preceding IF definition or it can be specified on a subsequent statement. All text to the right of the THEN clause is treated as comments.

Example (THEN clause on the same line as the IF statement)

```
//TESTIF  IF (&STG. EQ PRD) THEN   COMMENTS
//STEP1   EXEC PGM=COBOL1
//TESTIF           ELSE
//STEP2   EXEC PGM=COBOL2
//TESTIF  ENDIF
```

Example (THEN clause on a separate line from the IF statement)

```
//TESTIF  IF (&STG. EQ PRD)
//             THEN  COMMENTS
//STEP1   EXEC PGM=COBOL1
//TESTIF       ELSE
//STEP2   EXEC PGM=COBOL2
//TESTIF  ENDIF
```

Stacking IF-THEN-ELSE specifications on the same statement is not acceptable.

```
//T1    IF(&SYS. EQ ACC)THEN IF(&SUB. EQ REC)THEN
//STEP  EXEC PGM=COBOL1
//      ELSE
//STEP  EXEC PGM=COBOL2
//      ENDIF
//      ELSE
//STEP  EXEC PGM=COBOL3
//      ENDIF
```

The syntax in this IF-THEN-ELSE definition is rejected at translation time because anything on the first statement following the THEN clause is treated as a comment.

IF-THEN-ELSE specifications can be nested, however. The above could have been coded in the following manner.

```
//T1    IF(&SYS. EQ ACC)
//      THEN
//T2    IF(&SUB. EQ REC)
//      THEN
//STEP  EXEC PGM=COBOL1
//T2    ELSE
//STEP  EXEC PGM=COBOL2
//T2    ENDIF
//T1    ELSE
//STEP  EXEC PGM=COBOL3
//T1    ENDIF
```

Example (nested IF-THEN-ELSE statement)

DD statements related to an EXEC must be included within the IF-THEN-ELSE block for that step. CA Endevor SCM has no way of knowing that DD statements following an EXEC statement belong to a prior statement. In short, the step must be complete as if IF-THEN-ELSE were not coded. As an example, this processor is invalid:

```
//IF     IF     (&COBOL='1'). THEN
//STEP1  EXEC  PGM=COBOL1
//       ELSE
//STEP1  EXEC  PGM=COBOL2
//       ENDIF
//SYSIN  DD   DISP=SHR,DSN=INPUT.SOURCE
//SYSLIN DD  DISP=SHR,DSN=OBJECT.OUTPUT
//...
```

It is desirable to associate the DD statements with program COBOL1 or program COBOL2. However, in this case the DD statements are only associated with program COBOL2. If the IF-THEN-ELSE statements are removed, the DDs are allocated to the COBOL2 program. Internally, DD statements are associated with the prior exec statement. To associate the DD statements with the first exec statement, they must be duplicated in that step. This example is valid:

```
//       IF  (&COBOL='1')  THEN
//STEP1  EXEC PGM=COBOL1
//SYSIN  DD  DISP=SHR,DSN=INPUT.SOURCE
//SYSLIN DD  DISP=SHR,DSN=OBJECT.OUTPUT
//...
//       ELSE
//STEP1  EXEC PGM=COBOL2
//SYSIN  DD  DISP=SHR,DSN=INPUT.SOURCE
//SYSLIN DD  DISP=SHR,DSN=OBJECT.OUTPUT
//...
//       ENDIF
```

The relational expression of the IF statement tests six IBM-defined possible expressions and the CA Endevor SCM symbolic variables and literals. The six IBM expressions are:

- RC
- ABENDCC
- ABEND
- ¬ABEND
- RUN
- ¬RUN
- ¬RUN

The RC Expression

Indicates the relational expression tests a return code. Evaluate a return code by coding RC, a comparison operator, and a numeric value.

IF(RC=8)

Tests for a return code equal to 8 from any previous step.

IF(stepname.RC>=10)

Tests for a return code greater than or equal to 10 from a specified step.

The ABENDCC Expression

Indicates the relational expression tests for a system abend completion code or a user-defined completion code.

IF(ABENDCC=Sxx)

Tests true when most recent system abend is equal to Sxx.

IF(ABENDCC=Uxxxx)

Tests true when most recent user abend is equal to Uxxxx.

IF(stepname.ABENDCC=Sxxx)

Tests true when a specific step abends with Sxxx.

The ABEND Expression

Indicates the relational expression tests for an abend condition that occurred during processing of a prior job step.

IF(ABEND)

Tests true if an abend occurs on any previous step.

IF(stepname.ABEND)

Tests true if an abend occurred on a specific step.

The ¬ABEND Expression

Indicates the relational expression verifies an abend condition did not occur during the execution of a prior job step.

IF(¬ABEND)

Tests true when no abend occurred on any previous step.

IF(¬stepname.ABEND)

Tests true when no abend occurred on a specific step.

The RUN Expression

Indicates the relational expression tests for execution of a specific job step.

```
IF(stepname.RUN)
```

```
IF(stepname.RUN=TRUE)
```

The ¬RUN Expression

Indicates the relational expression verifies a specific step did not execute.

```
IF(¬stepname.RUN)
```

```
IF(stepname.RUN=FALSE)
```

The IF-THEN-ELSE Trace Facility

The IF-THEN-ELSE Trace facility helps you to determine if the IF THEN/ELSE logic is functioning correctly. When the Trace Facility is activated, trace records are written that indicate the result of IF-THEN-ELSE condition testing.

Include the following ddname in your CA Endevor SCM JCL to activate the trace.

```
//EN$TRITE DD DUMMY
```

The following is extracted from an output listing of an action with tracing activated:

```
C1G0249I //T01 IF ((&C1SY. EQ 'SYSTEM')AND(&C1SU. NE 'SUB1'))
C1G0249I //    OR((&C1SY. EQ 'SYSTEM2')AND(&C1SU. EQ 'SUB1'))
C1G0249I //    OR((&C1SY. NE 'SYSTEM2')AND(&C1SU. EQ 'SUB3')))
C1G0249I //T01      THEN
C1G0249I //T02 IF(('&C1ELEMENT(1.,3)'='ACC'))
C1G0249I //T02      THEN
C1G0249I //STEP1   EXEC PGM=COBOL1
C1G0249I //T02      ELSE
C1G0249I //STEP2   EXEC PGM=COBOL2
C1G0249I //T02      ENDIF
C1G0249I //T01      ELSE
C1G0249I //STEP3   EXEC PGM=COBOL3
C1G0249I //T03  IF (&C1TY='JCL').
C1G0249I //T03     THEN
C1G0249I //DD1    DD DSN=SYS2.PROCLIB,DISP=SHR
C1G0249I //DD2    DD DSN=SYS2.LINKLIB,DISP=SHR
C1G0249I //T03     ELSE
C1G0249I //T03     ENDIF
C1G0249I //T01     ENDIF
C1G0011I PROCESSOR SYMBOLIC SUBSTITUTION OCCURRED-
C1G0009I  ORIGINAL  :&C1SY.
C1G0009I  SUBSTITUTED:GIKSYS
C1G0009I  ORIGINAL :&C1SU.
C1G0009I  SUBSTITUTED:GIKSUB
```

```
C1G0009I  ORIGINAL :&C1SY.
C1G0009I  SUBSTITUTED:GIKSYS
C1G0009I  ORIGINAL :&C1SU.
C1G0009I  SUBSTITUTED:GIKSUB
C1G0009I  ORIGINAL :&C1SY.
C1G0009I  SUBSTITUTED:GIKSYS
C1G0009I  ORIGINAL :&C1SU.
C1G0009I  SUBSTITUTED:GIKSUB
C1G0009I  ORIGINAL :&C1ELEMENT(1.,3)
C1G0009I  SUBSTITUTED:T01
C1G0009I  ORIGINAL :&C1TY.
C1G0009I  SUBSTITUED:JCL
C1X0000I ITE(T01 )Expression: GIKSYS EQ SYSTEM1 Result: FALSE
C1X0000I ITE(T01 )Expression: GIKSUB NE SUB1 Result: TRUE
C1X0000I ITE(T01 )Connecting: FALSE AND TRUE, Result: FALSE
C1X0000I ITE(T01 )Expression: GIKSYS EQ SYSTEM2 Result: FALSE
C1X0000I ITE(T01 )Expression: GIKSUB EQ SUB1 Result: FALSE
C1X0000I ITE(T01 )Connecting: FALSE AND FALSE, Result: FALSE
C1X0000I ITE(T01 )Expression: GIKSYS NE SYSTEM2 Result: TRUE
C1X0000I ITE(T01 )Expression: GIKSUB EQ SUB3 Result: FALSE
C1X0000I ITE(T01 )Connecting: TRUE AND FALSE, Result: FALSE
C1X0000I ITE(T01 )Connecting: FALSE OR FALSE, Result: FALSE
C1X0000I ITE(T01 )Connecting: FALSE OR FALSE, Result: FALSE
C1X0000I ITE(T01 )ITE IF encountered, truth set to FALSE
C1X0000I ITE(T02 )Expression: T01 EQ ACC Result: FALSE
C1X0000I ITE(T02 )ITE IF encountered, truth set to FALSE
C1X0000I ITE(T02 )ITE ELSE encountered, truth set to TRUE
C1X0000I C1GP2000: Step STEP2 selected for execution by ITE
C1X0012I STEP STEP2 INVOKING PROGRAM COBOL2
C1X0010I STEP STEP2 PROGRAM COBOL2 COMPLETED, RC=0000
C1X0000I C1GP2000: Step STEP2 has completed, RC(0000)
C1X0000I ITE(T02 ) ITE ENDIF encountered, truth set to FALSE
C1X0000I ITE(T01 ) ITE ELSE encountered, truth set to TRUE
C1X0000I C1GP2000: step STEP3 selected for execution by ITE
C1X0000I ITE(T03 ) Expression: JCL EQ JCL Result: TRUE
C1X0000I ITE(T03 ) ITE IF encountered, truth set to TRUE
C1X0000I DD DD1  included by ITE
C1X0000I DD DD2  included by ITE
C1X0012I STEP STEP3 INVOKING PROGRAM COBOL3
C1X0010I STEP STEP3 PROGRAM COBOL3 COMPLETED, RC=0000
C1X0000I C1GP2000: Step STEP3 has completed, RC(0000)
```

Symbolic Resolution Trace Facility

The Symbolic Resolution trace facility can help support technicians diagnose errors in processors that use symbolic variables. To turn on the trace facility, include a DD statement with DDname EN$TRSYM in the execution JCL.

Note: For more information, see the appendix "Using the Trace Facilities" in the *Administration Guide*.

Dynamically Allocated Libraries

You can create footprints, monitor components, or control creation of backout members on dynamically allocated libraries by adding EN$DYNxx DD statements to a processor.

Some user programs in a processor dynamically allocate data sets. Typical examples are IKJEFT01 and EDCPRLK. When these programs read or write members to the data sets, the user has no control over the following parameters:

- MONITOR=COMPONENTS
- BACKOUT=N
- FOOTPRNT=CREATE

As a result, the default values for these parameters are used. Therefore, by default:

- No components are monitored
- Backouts are written
- No footprints are written

You can change these defaults, even though the file allocations are beyond your control, by specifying special DDNAMES for the data sets that receive special treatment. Any DDNAME starting with EN$DYN can be used to specify one or more data set names with their monitor, backout, and footprint specification.

Note: CA Endevor SCM does not support FOOTPRNT=CREATE on a concatenated data set. Therefore, if FOOTPRNT=CREATE is required, more than one statement is required.

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

```
//STEP1    EXEC PGM=IKJEFT01
//EN$DYN00 DD DISP=SHR,DSN=&SYSLIB1.,BACKOUT=NO
//EN$DYNM  DD DISP=SHR,DSN=&SYSLIB2.,FOOTPRNT=CREATE
```

In the following example, components are monitored from all files in the SYSLIB concatenation, even if EDCPRLK reallocates each file separately under another DDNAME.

```
//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.
```

Note: In the previous example, you can remove the MONITOR=COMPONENTS statements from the SYSLIB concatenation. This is because the MONITOR=COMPONENTS statements on the EN$DYNDS DD statement cause all components to be monitored, regardless of whether each file is reallocated separately under another DDNAME.

The Internal Reader

By default, any job submitted to the internal reader by a processor step runs under the security credentials of the user ID. You can override this by specifying INTRDR_ALTID=ON in the CA Endevor SCM options table. When this value is specified, jobs submitted to the internal reader run under the security context of the alternate ID.

However, if you code ALTID=N on the EXEC statement of a processor, then any jobs submitted to the internal reader run under the security credentials of the user ID, regardless of the setting of INTRDR_ALTID.

Note: For more information, see Alternate ID Support in the *Security Guide*.

If you are using CA Top Secret for z/OS, the CA Top Secret global option of JES(VERIFY) should be set if you want CA Endevor SCM to submit JOBs to the INTRDR, using the credentials of the CA Endevor SCM User ID. If JES(VERIFY) is not specified, CA Endevor SCM's default behavior for CA Top Secret is to submit jobs to the internal reader using the alternate ID.

Chapter 3: Using Processor Utilities

This section contains the following topics:

Including Processor Utilities in Processor Logic

The following utilities are distributed with CA Endevor SCM. You can include these utilities anywhere in your processor logic, to perform the functions described. A Yes or No indicates USS File Support.

BC1PDSIN

(No) Initializes any number of allocated data sets that begin with ddname C1INIT. This utility is generally used to allocate list data sets.

BC1PTMP0

(Yes) Executes TSO commands, once a TSO environment has been created. This program accepts as input a parameter defining the data set containing the TSO commands to be called. BC1PTMP0 acts as a terminal monitor program and command processor, calling one program (EXEC) to place the commands in a TSO stack, then issuing GETLINE requests to extract the commands from the TSO stack.

BC1PXFPI

(No) Installs “transportable” footprints in object modules generated under z/OS, VSE, or z/VM.

Note: For more information about "transportable" footprints, see the *Administration Guide*.

BC1PMVCL

(No) Provides the ability to update the component list to reflect the location of the output components as the result of the move processor. (The Delta library, where the component list is stored, cannot be a USS file.)

BSTCOPY

(No) Copies members from one partitioned data set to another. BSTCOPY allows for the use of MONITOR=COMPONENTS and backout.

C1BM3000

(Yes) Executes CA Endevor SCM from within a processor.

C1PRMGEN

(No) Creates 80-column (card-image) statements from a parameter passed to the utility. These statements, once expanded, are passed as input control statements to subsequent job steps. C1PRMGEN expands any CA Endevor SCM symbolic parameters contained in the statements, allowing you to vary the input control specifications based on the values of CA Endevor SCM symbolic parameters.

CONAPI

(Yes) Invokes a program which issues API calls through a processor.

CONDELE

(Yes) Removes a member from a user library or USS directory, after verifying the footprint for the member.

CONLIST

(No) Manages output listings generated by the processors: store (and footprint) new members in listing libraries, print listings stored as sequential files or as members in listing libraries, or copy a member from one listing library to another (optionally after appending one or more listings at the end of the member).

CONPARMX

(Yes) Reduces processor complexity and the number of CA Endevor SCM Processor Groups. The primary use of CONPARMX is to support varying program (for example: compiler or linkage editor) options without having to create and update large numbers of processors or processor groups by allowing different levels of Options members (Default, Processor Group, Element) to be included at execution time.

CONRELE

(Yes) Includes entities related to an element in a component list when generating component list reports and when using the LIST action.

CONSCAN

(No) Creates ACM relationships between a CA Endevor SCM ELEMENT and scanned values from the content of the ELEMENT. It then applies parsing rules to the ELEMENT content, and passes these as standard CONRELE syntax to the CONRELE step as exemplified in the CONSCAN prototype.

CONWRITE

(Yes) Combines all levels of an element, optionally expands INCLUDE statements and writes the merged source to a user-specified data set or USS directory. Can be used as input to a specified program.

Writes a component list to an external data set for further processing.

ENBX1000

(Yes) For more information, see the “Expand Includes” section of the *Utilities Guide*.

ENUSSUTL

(Yes) Collects package backout information for USS processor output files.

LEXTRCTR

To parse and capture all the pieces of a CSP application when the user is managing at the application rather than the component level.

BC1PCCSP

Attaches the compiler for each piece of the application identified by LEXTRCTR.

Note: The following non-processor utilities do not support USS files:

- BC1PNLIB-ELIB Copy Utility
- ENBS1000-Text Search-and-Replace
- ENBX1000-Include/Copy Member Expansion Utility
- C1BML000-Initial Inventory Load Utility

Important! ELIB utilities, BC1PNLIB, NCPY, and NLST cannot be used within a processor.

The BC1PDSIN Utility

BC1PDSIN can be used to initialize sequential data sets. For example, in a processor with COMPILE, LINK, and CONLIST steps (where CONLIST uses temporary listing data sets created by the COMPILE and LINK steps), an allocation error would occur in the CONLIST step if the COMPILE or LINK did not run thereby not creating the needed temporary data sets. If you use BC1PDSIN, however, it allocates temporary data sets, so CONLIST has the files it needs to execute.

BC1PDSIN initializes all data sets that are allocated by the C1INIT or C1LIST DD statements. If you specify more than one C1INIT or C1LIST data sets, assign the ddnames sequentially. For example, C1LIST01, C1LIST02 and so forth, or C1INIT01, C1INIT02, and so forth.

To initialize a PDS within BC1PDSIN, specify:

- The member name in the data set name
- DSORG=PO
- The number of directory blocks to allocate in the SPACE parameter.

To allocate but not initialize an entire temporary PDS, use a DDNAME that does not begin with //C1INIT. For example, //C2INIT, //TEMPPDS, or //C1INTPO. This will allocate the library in the same manner as IEFBR14.

Sample JCL

A sample JCL for the BC1PDSIN utility is shown:

```
//STEPNAME   EXEC PGM=BC1PDSIN
//C1INITxx   DD DSN=&&COBLST,DISP=(,PASS,DELETE),
//              UNIT=SYSDA,SPACE=(TRK,(1,2),RLSE),
//              DCB=(RECFM=FBA,LRECL=121,BLKSIZE=3630,DSORG=PO)

//C1LISTxx   DD DSN=&&LNKLST,DISP=(,PASS,DELETE),
//              UNIT=SYSDA,SPACE=(TRK,(1,2),RLSE),
//              DCB=(RECFM=FBA,LRECL=121,BLKSIZE=3630,DSORG=PO)
```

The BC1PTMP0 Utility

BC1PTMP0 allows TSO commands to be executed once a TSO environment has been created. The program accepts as input a parameter that defines the data set containing the TSO commands to be called. This program is usually used for processors that are executed under the CA Endevor SCM ISPF dialog.

Note: BC1PTMP0 issues TSO command processor STACK services using the BARRIER option. Only releases of TSO that support the BARRIER option can use this utility.

Note: If using a REXX EXEC instead of a TSO CLIST, a PUSH END statement must be coded at the end of the EXEC.

BC1PTMP0 first checks whether a TSO environment is present and performs the following actions:

- If no TSO environment is present, BC1PTMP0 ends with a return code of **5**.

  Note: For more information about return codes, see Return Codes (see page 66).

- If a TSO environment is present, BC1PTMP0 acts as a terminal monitor program and command processor. BC1PTMP0 calls the program **EXEC**, which then places the commands (specified in the data set coded on the PARM= parameter) in the TSO stack. BC1PTMP0 will issue GETLINE requests to extract those commands from the TSO stack.

TSO does not allow the program **IKJEFT01**, a TSO program, to be executed once TSO processing has started. (That is, TSO does not allow TSO to run under it.) Executing BC1PTMP0 in a processor rather than executing IKJEFT01 allows TSO commands to be executed within a CA Endevor SCM processor.

Both the CA Endevor SCM Interface for InfoMan and the CA Endevor SCM for DB2 product require the use of BC1PTMP0.

You can code both a BC1PTMP0 step and a IKJEFT01 step in a processor, so that BC1PTMP0 executes, if the processor is running under TSO, and IKJEFT01 executes, if the processor is running in batch mode. Use the following steps to implement this technique.

```
//TSOMODE      EXEC PGM=BC1PTMP0
//XX
//BATMODE      EXEC PGM=IKJEFT01,COND=(5,NE,TSOMODE)
//XX
```

Important! ISPF services cannot be invoked in a TMP. CA Endevor SCM is using ISPF services to invoke the processor and an attached TMP (BC1PTMP0) can not start another ISPF service in the same TSO address space. ISPF services (ISPEXEC, ISPSTART, etc.) cannot be invoked via BC1PTMP0.

Sample JCL and Parameters

The following JCL is specified in the processor:

```
//STEPNAME EXEC  PGM=BC1PTMP0,
//    PARM='uprfx.uqual.CLISTLIB(clist)'
//STEPLIB DD DSN=iprfx.iqual.CSIQLOAD,DISP=SHR
```

The parameters in the previous syntax are described as follows:

PARM=

The name of the data set containing the commands to be executed by BC1PTMP0.

uprfx.uqual.CLISTLIB

The name of the CLIST library containing the commands that you want to execute.

clist

The name of the CLIST that you want to execute.

iprfx.iqual.CSIQLOAD

The CA Endevor SCM installation load library containing the program BC1PTMP0. (For DB2 processors, you may need to add your DB2 load library.)

Return Codes

BC1PTMP0 utility can return any of the following return codes:

0

All commands were successfully processed.

5

TSO (that is, program IKJEFT01) currently is not active. This step within the processor was executed, however.

6

No parameter was specified in the PARM= statement within the JCL, or a data set name greater than 56 characters was specified.

7

A command specified in the CLIST was not found. BC1PTMP0 attempts to load all programs prior to issuing an ATTACH. If the load fails, this return code is passed back to CA Endevor SCM.

8

The PARM= specification on the EXEC card was coded incorrectly, for example:

- No parameter was passed
- Parameter is too large (greater than 54)
- Parameter does not end with a pair of parentheses surrounding a member name
- Member name portion of the parameter is invalid

9

A GETMAIN failed for the command buffer required by attached programs. BC1PTMP0 attempted to acquire a command buffer in subpool 78 and the GETMAIN failed.

Any program or command invoked by BC1PTMP0 can issue any return code for their own reasons. This includes return codes 5,6,7,8,9 and any other return code. BC1PTMP0 simply returns the return code passed by those programs. For example, a command invoked by BC1PTMP0 may return a RC=8 when it gets an allocation error. A different command invoked by BC1PTMP0 may return a RC=8 when a program is not found.

REXX Exec or CLIST Executing a DB2 Bind

To enable BC1PTMP0 to invoke a REXX exec or CLIST to execute a DB2 Bind, a DD statement with the name EN$SELCT must be coded in the BC1PTMP0 definitions.

Assume that a processor running in foreground is executing BC1PTMP0 which invokes a CLIST or a REXX exec to execute a DB2 Bind. A STEPLIB in the BC1PTMP0 step defines the library containing the DB2 DSN*xxxxx* programs. In this case, the STEPLIB is ignored and an 806 occurs. The error occurs if the following processor step and CLIST are coded:

Example processor step:

```
//TMP0 EXEC PGM=BC1PTMP0
// PARM='PROD.CLIST(TSTCLIST)'
//STEPLIB DD DSN=PUBLIC.PXXXX.STEPLIB,DISP=SHR
//SYSTERM DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSOUT DD SYSOUT=*
```

Example CLIST TSTCLIST:

```
DSN SYSTEM(D91A)
BIND PLAN(P2229PRG) -
MEMBER (P2229PRG) -
OWNER (KEIGR01) -
ACTION (REPLACE) RETAIN -
VALIDATE (BIND)
```

The problem occurs due to the way the program DSN is invoked internally by BC1PTMP0. To correct this, the REXX exec or CLIST must be invoked by a different method. To use this method, code a DD statement with the name EN$SELCT in the BC1PTMP0 definitions. Also, the library containing the REXX exec must be a member of the SYSPROC dd definition. In the following corrected processor step, the clist library 'PROD.CLIST' was defined to the SYSPROC DD concatenation.

Example of a corrected processor step:

```
//TMP0 EXEC PGM=BC1PTMP0
// PARM='PROD.CLIST(TSTCLIST)'
//STEPLIB DD DSN=PUBLIC.PXXXX.STEPLIB,DISP=SHR
//SYSTERM DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//EN$SELCT DD DUMMY
```

If this method is used, then the STEPLIB will be passed through to the invocation of the DB2 DSN programs.

The BC1PXFPI Utility

The BC1PXFPI utility installs “transportable” footprints in object modules generated under z/OS, VSE, or z/VM. Transportable footprints provide footprint audit trails for software that executes in non-z/OS environments (VSE and z/VM).

Note: For more information about transportable footprints, see the *Footprints Guide* (This facility is for non-z/OS environments only).

The BC1PMVCL Utility

The BC1PMVCL utility provides the ability to update the component list to reflect the location of the output components as the result of the move processor.

A move processor typically copies the outputs of the generate processor from their libraries to the next stage's output libraries. Once the outputs have been copied, the move processor typically also copies the existing component list to the new inventory area. The utility to move component lists is a pseudo processor program called BC1PMVCL.

When using BC1PMVCL, the existing component data does not get updated so all the input and output components in the component list reflect the data sets used at generate time. If, in a move processor, MONITOR=COMPONENTS is coded on the output files, and BC1PMVCL is part of the processor (as the last step), CA Endevor SCM will:

- Remove existing output components from component list
- Add new output components to component list
- Will not change input components

Technically, the BC1PMVCL does not have to be the last step; however, it must execute after all monitored components. If components have been monitored in a move processor (prior to the execution of BC1PMVCL), then all of the gathered output components will be used to replace all of the existing output components in the existing component list. These gathered output components are subsequently removed from the storage lists in which they were collected in order to avoid subsequent update at the end of the processor execution.

This will cause output component lists to be generated by a move if a move processor has collected new output components prior to the execution of BC1PMVCL. The output components will replace all of the output in the existing components list.

BC1PMVCL provides the ability to update the component list to reflect the location of the output components as the result of the move processor.

The move processor utility BC1PMVCL now can dynamically replace old Endevor output components with the new output components at the end of the move processor execution. By using BC1PMVCL in your move processor, Quick Edit commands LL and LO will function after a move.

The BSTCOPY Utility

The BSTCOPY utility offers a limited subset of the functionality provided by IEBCOPY.

You must use BSTCOPY instead of IEBCOPY if you use the CA Endevor SCM Automated Configuration Manager (ACM), or if package backout has been enabled for the output library. This is because CA Endevor SCM cannot determine which members are changed by IEBCOPY.

Note: BSTXCOPY is a CA Endevor SCM utility that performs the same functions as the CA Endevor SCM processor utility BSTCOPY. However, you can use BSTXCOPY to perform these functions outside of a CA Endevor SCM processor. For more information about BSTXCOPY, see The BSTXCOPY Utility, in the *Utilities Guide*.

BSTCOPY Copy Functions

BSTCOPY can be used to copy between PDS load libraries and PDSE load libraries, from PDSE to PDSE, and from PDSE back to PDS. It cannot copy between a PDSE load library and any other CA Endevor SCM access method (CA Librarian, CA Panvalet, ELIB).

BSTCOPY can copy aliases, but the alias must be explicitly requested. To copy an alias, the original loadlib member must be copied first. For more information about aliases, see BSTCOPY and Aliases (see page 71).

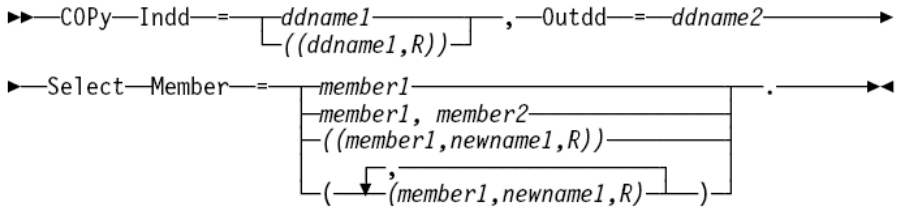
Copying from a PDSE load library to a PDS may result in errors, depending on whether the PDSE contains any program object members which cannot be converted back to load modules. This includes program objects which are greater than 16 megabytes in size, or which included mixed-case, extended names, or contain more than 32K of external symbols.

Note: For more information about the conversion restrictions, refer to the program management manuals for DFSMS/MVS 1.1 or higher.

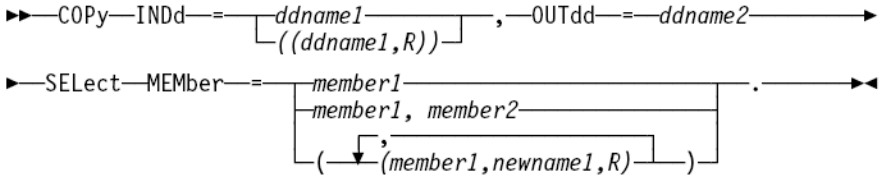
BSTCOPY is not intended to replace IEBCOPY. Rather, it is provided to support simple member copy operations from one library to another.

When a PDS load module is copied to a PDSE, CA Endevor SCMinframe load module footprint information is retained only if the CA Endevor SCM processor utility BSTCOPY is used. Other utilities, such as IEBCOPY, do not have the capability to copy the *LOADMOD footprint. For more information, see TEC316937, How to Convert AllFusion Endevor Change Manager Load Libraries between PDS and PDSe Formats, on ca.com/support.

BSTCOPY Syntax - Literal Interpretation



BSTCOPY Syntax - Alternate Interpretation



Important! Spaces before or after the comma between COPY INDD and OUTDD results in a syntax error.

BSTCOPY and Aliases

Load modules and program objects with an alias can be handled by BSTCOPY even though BSTCOPY does not support COPYGRP syntax. The following methods are suggested.

- To bring loadonly software under CA Endevor SCM control, use the following method, which requires two steps in the Generate processor:
  - A step to COPYGRP with IEBCOPY into a temporary PDSE, and then
  - A BSTCOPY step to copy all the results into the final target PDSE library. This latter step ensures that the *CA Endevor SCM Automated Configuration* option output components are monitored, while preserving the aliases.

The following Generate Processor example uses this method:

```
//*************************************************************
//* FIRST COPY LOADMODULE TO TEMPORARY PDSE
//*************************************************************
//CPY1    EXEC PGM=IEBCOPY,MAXRC=04
//SYSPRINT  DD SYSOUT=*
//SYSUT3    DD UNIT=&WRKUNIT,SPACE=(TRK,(1,1))
//SYSUT4    DD UNIT=&WRKUNIT,SPACE=(TRK,(1,1))
//INDD      DD DSN=&C1USRDS,DISP=SHR
//OUTDD     DD DSN=&&TEMPDSN,DISP=(,PASS),UNIT=SYSDA,
//             SPACE=(CYL,(5,5,44)),DSNTYPE=LIBRARY,
//             DCB=(LRECL=0,BLKSIZE=32760,RECFM=U,DSORG=PO)
//SYSIN     DD *
   COPYGRP OUTDD=OUTDD,INDD=INDD
   SELECT MEMBER=&C1ELEMENT
//*********************************************************************
//* NOW COPY All INTO TARGET OUTPUT LOADLIB MONITOR=COMPONENTS         *
//*********************************************************************
//CPY2    EXEC PGM=BSTCOPY,MAXRC=04
//SYSPRINT  DD SYSOUT=*
//SYSUT3    DD UNIT=&WRKUNIT,SPACE=(TRK,(1,1))
//SYSUT4    DD UNIT=&WRKUNIT,SPACE=(TRK,(1,1))
//INDD      DD DSN=&&TEMPDSN,DISP=(OLD,PASS)
//OUTDD     DD DSN=&LOADLIB1,DISP=SHR,MONITOR=COMPONENTS,
//            FOOTPRNT=CREATE
//SYSIN     DD *
   COPY    OUTDD=OUTDD,INDD=((INDD,R))
```

- To copy aliases of load modules or program objects inside a Move processor, use the following method. The temporary library and the source library for IEBCOPY must be the same library type (either PDS or PDSE). Otherwise IEBCOPY might not copy the *LOADMOD Footprint.

  The following Move Processor example uses this method:

```
//IF2  IF (&C1ACTION=TRANSFER) OR (&C1ACTION=MOVE) THEN
//********FIRST IEBCOPY COPYGRP TO PRESERVE ALIAS**********************
//TRCOPY1 EXEC PGM=IEBCOPY,MAXRC=04
//SYSPRINT  DD SYSOUT=*
//FCOPYOFF DD DUMMY
//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=&&TMPDSN,DISP=(,PASS),UNIT=SYSDA,
//             SPACE=(CYL,(5,5,44)),DSNTYPE=LIBRARY,
//             DCB=(LRECL=0,BLKSIZE=32760,RECFM=U,DSORG=PO)
//SYSIN     DD *
   COPYGRP OUTDD=OUTDD,INDD=INDD
   SELECT MEMBER=&C1ELEMENT
```

```
//********NOW COPY ALL IN TEMPORARY TO TARGET PDSE**********************
//TRCPY2 EXEC PGM=BSTCOPY,MAXRC=04
//SYSPRINT  DD  SYSOUT=*
//SYSUT3    DD  UNIT=&WRKUNIT,SPACE=(TRK,(1,1))
//SYSUT4    DD  UNIT=&WRKUNIT,SPACE=(TRK,(1,1))
//OUTDD     DD  DSN=&LOADLIB2,DISP=SHR,MONITOR=COMPONENTS
//INDD      DD  DSN=&&TMPDSN,DISP=SHR,FOOTPRNT=VERIFY
//SYSIN     DD  *
   COPY    OUTDD=OUTDD,INDD=((INDD,R))
//********NOW MOVE THE COMPONENT LIST ..must be the LAST step       ***
//BC1PMVCL EXEC PGM=BC1PMVCL,COND=(0,NE)
//END2  ENDIF
```

BSTCOPY Unsupported Functions

BSTCOPY does not:

- Copy members of a partitioned data set to a sequential data set.
- Compress partitioned data sets.
- Copy load modules that have the linkage editor OVERLAY attribute.
- Reblock load modules when the BLKSIZE of the input library is greater than that of the output library.
- Support multiple DD definitions on the INDD statement.
- Support splitting a SELECT statement on multiple lines. A SELECT statement must be specified on one physical input record.
- Support FREE-CLOSE on any input or output DD names.

```
//BSTCOPY  PGM=BSTCOPY,MAXRC=0
//SYSPRINT DD  SYSOUT=*
//IN1      DD  DISP=SHR,DSN=DSNAME1
//IN2      DD  DISP=SHR,DSN=DSNAME2
//OUT1     DD  DISP=OLD,DSN=DSNAME3
//SYSIN    DD  *
    COPY INDD=((IN1,R)),OUTDD=OUT1
    SELECT MEMBER=MEMBER1
    COPY INDD=IN2,OUTDD=OUT1
    SELECT MEMBER=((MEMBER2,,R),(MEMBER3,,R))
    SELECT MEMBER=((MEMBER4,NEWNAME4,R))
 /*
```

BSTCOPY and OVERLAY Modules

BSTCOPY does not support the copy of a load module that is linked as OVERLAY. When BSTCOPY cannot be used, backout processing is effectively disabled for packages.

One solution is to relink the load modules. If the load modules are vendor-supplied, however, and relinking may jeopardize ongoing vendor support, another solution is to write a two-step processor. Follow these steps to create the new processor:

1. Create a dummy module and execute a BSTCOPY step that copies it and renames it to the name of the OVERLAY module. This bypasses the system's security and renames the target module, creating a backout.
2. Code an IEBCOPY step that copies the real module in, overlaying the dummy module. Because IEBCOPY cannot be screened, backouts are not affected.

The following processor executes these two steps:

```
//STEP1    EXEC PGM=BSTCOPY
//SYSPRINT DD SYSOUT=*
//IN       DD DSN=source.data.set,DISP=SHR
//OUT      DD DSN=target.data.set,DISP=SHR
//SYSIN    DD *
 C I=IN,O=OUT
 S M=((dumymbr,&C1ELEMENT.,R)) <===DUMMY MEMBER COPIED TO CREATE BACKOUT
//STEP2    EXEC PGM=IEBCOPY
//SYSPRINT DD SYSOUT=*
//SYSUT3   DD UNIT=SYSDA,SPACE=(TRK, (5,5))
//SYSUT4   DD UNIT=SYSDA,SPACE=(TRK, (5,5))
//IN       DD DSN=source.data.set,DISP=SHR
//OUT      DD DSN=target.data.set,DISP=OLD
//SYSIN    DD *
 C I=IN,O=OUT
 S M=((&C1ELEMENT.,,R)) <===COPY REAL MEMBER
```

For PDSMAN users

1. Using the PDSMAN $IEBCOPY statement, BSTCOPY can be replaced with the PDSMAN utility FASTCOPY. Specify NAME=BSTCOPY on the $IEBCOPY statement to enable this support. Package shipment jobs and processors coded to run BSTCOPY in this situation will execute FASTCOPY instead.
2. To disable substitution of FASTCOPY for BSTCOPY in this environment, code a DD statement in the processor JCL for the ddname 'FCOPYOFF,' allocated to DD DUMMY.

3. For CA Endevor SCM Automated Configuration Option users, when executing FASTCOPY, input components will not be collected. Output components will be collected and package backout members and data are collected.
4. OVERLAY and SCTRLOAD modules are supported when using FASTCOPY.
5. Set the $IEBCOPY operand STORFAIL= to the value TERMINATE when using FASTCOPY substitution for BSTCOPY.

SYSPRINT DCB Information

When directing SYSPRINT output to a data set, the DCB information specified should be as follows:

- RECFM=VBA
- LRECL=121
- BLKSIZE=125

The C1BM3000 Utility

The C1BM3000 utility allows you to execute CA Endevor SCM actions from within a processor.

You must pass the SCLIN (SCL input) and MSGOUT1 (output messages) parameters to the C1BM3000 utility. Optionally, if you want:

- A package ID associated with this execution, you must pass the PACKAGE parameter.
- CA Endevor SCM to write the Action Summary report to a separate file, you must pass the MSGOUT2 parameter.

Important! When a package action executes a processor that invokes C1BM3000, the actions run by the C1BM3000 step are considered non-package actions. If the actions update an inventory area that has approver relations, the C1BM3000 utility issues a C1U0900E error (package processing required). Also, backout information is not created for actions run in a C1BM3000 step.

Example (Sample JCL)

Sample JCL for the C1BM3000 utility is shown next. This JCL specifies a SCL input file, and the messages and Action Summary report are written to SYSOUT. A package ID is not specified, as indicated by the two commas between MSGOUT1 and MSGOUT2. The two commas are required.

```
//C1BM3000  EXEC PGM=C1BM3000,PARM='SCLIN,MSGOUT1,,MSGOUT2'
//STEPLIB   DD  DSN=iprfx.iqual.CSIQAUTH,
//          DISP=SHR
//MSGOUT1   DD  SYSOUT=*
//MSGOUT2   DD  SYSOUT=*
//SYSABEND  DD  SYSOUT=*
//SCLIN     DD  DSN=uprfx.uqual.SCL,DISP=SHR
//          DCB=(RECFM=FB,LRECL=80,BLKSIZE=6160,DSORG=PS)
//
```

Note: You need not specify the CONLIB DD statement, because the library has already been specified by the batch job or by the logon procedure in foreground.

Considerations When Using the C1BM3000 Utility

Do not use the following:

- BSTIPT01 for SCLIN, the name is reserved by CA Endevor SCM for input SCL.
- C1MSGS1 for MSGOUT1, the name is reserved by CA Endevor SCM for batch execution report messages.
- C1MSGS2 for MSGOUT2, the name is reserved by CA Endevor SCM for the Action Summary report.

Important! A C1BM3000 step in a processor step cannot perform actions against an element that is currently being acted upon. Only actions that do not require exclusive use of the element, for example LIST, PRINT or TRANSFER with BYPASS ELEMENT DELETE, can be executed against the same element. Failure to observe this rule will lead to an endless wait where the primary C1BM3000 task waits for the secondary task to terminate while the secondary C1BM3000 task waits for the primary task to release the element.

C1WORK01-06 DD Statements

When you use the C1BM3000 utility to TRANSFER an element, the datasets are dynamically allocated to process data. The dynamically allocated datasets may not be sufficient, if a large element is transferred, and the System abort B37 error may occur. In this case, you can use the C1WORK01-C1WORK06 DD statements to override the standard dynamic allocation of datasets, and run the TRANSFER action again. These DD statements override the following records during the TRANSFER action:

- C1WORK01-Merged configuration records
- C1WORK02-Configuration delta records
- C1WORK03-Configuration base records
- C1WORK04-Merged element records
- C1WORK05-Element delta records
- C1WORK06-Element base records

You can code any or all of the previous DD statements. If you code one C1WORK DD statement, then it overrides the equivalent dynamic allocation of datasets. For example, if a very large element is transferred the regular dynamic allocation of datasets may not be sufficient for storing element base, element delta, and merged element records. In this case, you can code the C1WORK04, C1WORK05 and C1WORK06 statements to override the dynamic allocation of datasets.

A sample JCL illustrating how to code these DD statements is shown as follows:

```
//STEP1     EXEC PGM=C1BM3000,PARM='SCLIN,MSGOUT1,,MSGOUT2'
//MSGOUT1   DD  SYSOUT=A
//MSGOUT2   DD  SYSOUT=A
//SCLIN     DD  *
//ARC       DD  DISP=OLD,DSN=ARCHIVE.DATASET
//C1WORK04  DD  UNIT=SYSDA,SPACE=(CYL,(50,50)),
//              DCB=(RECFM=VB,LRECL=5000,BLKSIZE=6233)
//C1WORK05  DD  UNIT=SYSDA,SPACE=(CYL,(50,50)),
//              DCB=(RECFM=VB,LRECL=5000,BLKSIZE=6233)
//C1WORK06  DD  UNIT=SYSDA,SPACE=(CYL,(50,50)),
//              DCB=(RECFM=VB,LRECL=5000,BLKSIZE=6233)
//BSTIPT01  DD  *
```

The C1PRMGEN Utility

The C1PRMGEN utility creates 80-column (card-image) statements from a parameter passed to it. These statements are passed as input control statements to subsequent job steps.

To create card-image data, you can include CA Endevor SCM symbols in the parameter passed to C1PRMGEN. If you do this, the utility expands the symbols as it creates the output statements, allowing you to vary the input control statements passed to subsequent job steps based on the values of CA Endevor SCM symbolic parameters. This is illustrated next, where C1PRMGEN creates two control statements for use by the IBM IEBCOPY utility.

Example (Sample JCL)

To use C1PRMGEN, pass the data to be expanded in the PARM= parameter of the EXEC statement. To create more than one card-image statement, separate the statements using a vertical bar (**|**) in the PARM= value, as illustrated in the following sample JCL:

```
//STEPNAME EXEC PGM=C1PRMGEN,
//              PARM=' C I=I,O=O| S M=((&&C1ELEMENT.,,R))'
//PARMOUT    DD  DSN=&&CPYPARM,DISP=(,PASS,DELETE)
//               DCB=(RECFM=FB,LRECL=80,BLKSIZE=80)
```

The parameters in the previous sample JCL are described as follows:

PARM=

The statement to be expanded, enclosed either in single quotes or parentheses. This statement can be 1-100 characters in length. Once expanded, it is output in card-image format starting in column 1. To output more than one card-image statement, use the separator character (**|**) in the PARM statement, as shown above.

If the statement contains more than 72 characters (up to 100 characters are allowed), you can add a second line to the parameter by coding the statement as follows:

- Enter the first 71 characters of the line.
- In position 72, type any non-blank character (except a single quote), and continue entering your data on the next line.
- Type a (single) quote only at the end of the entire statement. Note the example below:

```
Col                                                          Col
1                                                             72
PARM='abcd.............................................zx
//  z.............................................a'
```

- To include a quotation mark within the card-image data, specify two contiguous quotes: ''.
- You can include an unbalanced parentheses, with the restriction that you must use single quotes as the surrounding delimiters in this case. You cannot include unbalanced parentheses when the enclosing characters are parentheses, however.

PARMOUT

The data set to which the expanded statement is written.

In the previous example, the output written to PARMOUT, assuming that the current element is NDVR, is as follows:

```
Col
1
C I=I,O=O
S M=((NDVR,R))
```

The CONAPI Utility

The CONAPI utility allows you to execute a program that issues API calls through a processor.

A program that issues API calls CANNOT be executed from a processor directly. You must use this utility passing it the name of your program through the PARM= parameter on the EXEC statement. If your program requires parameter data, you may append it to the parameter string using a comma to separate the program name from your parameter data.

Note: For more information about the API interface, see the *API Guide*.

Example (Sample JCL)

A sample JCL is shown:

```
//STEPNAME  EXEC PGM=CONAPI,PARM='APIPROG1,1,22,333'
```

CONAPI invokes program APIPROG1. On entry to APIPROG1, register 1 contains an address which points to an address for the parameter data. The first two bytes of parameter data contains the length after which follows the data.

In the above example storage (in hex) would appear as follows:

```
0008F16BF2F26BF3F3F3
```

The CONDELE Utility

The CONDELE utility removes a member (load module, listing, etc.) from an output library. The output library can be any of the following:

- Source output library
- Processor listing library
- Processor load library
- User output library
- USS directory

This utility is generally used in delete processors.

Example (Sample JCL)

Sample JCL for this utility is shown next:

```
//STEPNAME  EXEC PGM=CONDELE,PARM='mbr-name'
//C1LIB       DD  DSN=CA.STAGE1.LOADLIB,DISP=SHR
```

The parameters in the preceding syntax are described as follows:

PARM=

Indicates an alternate member name (mbr-name) for the element. You have the option of overriding the default member name. To specify all output components (including object modules, load modules, and listings) instead of an individual member name, specify 'PARM=*COMPONENTS'. CONDELE accesses the component list at the current location and deletes all output components from that list.

Important! CONDELE does not verify the component list was generated at the current location. You need to verify this independently before using the PARM=*COMPONENTS parameter.

C1LIB

Specifies the library or USS directory containing the member targeted for deletion. The member name is the name of the element being processed (generally moved, deleted, or archived).

The CONLIST Utility

The CONLIST utility is a multi-purpose utility used to manage output listings. Specify one of the following options on the PARM= statement on the EXEC card. You can also specify the MBR option with any of these options, except for the PRINT option.

STORE

Consolidates and compresses one or more temporary list data sets into a member in the output library defined by the DD statement C1LLIBO. CONLIST converts the data sets to the record format of the output library, and uses the member name as the element name. The output library can be a PDS with record size and record format appropriate for listings (RECFM=VBA is recommended), or a ELIB data set which is stored as if RECFM=VBA. You can optionally generate a banner page and store it at the front of the member.

Note: RECFM=VBA forces the listing to be stored in compressed format. To store the listing in uncompressed (human readable) format, allocate the library with RECFM=FBA instead.

If more than one file is input, CONLIST concatenates the files before storing them.

The listing files must be sequential; they cannot be PDS members. If you use a PDS, you receive an error message stating the member cannot be found in the directory.

PRINT

Default. Prints a temporary list data set, optionally generating a banner page before the listing. If more than one file is input, CONLIST concatenates the files before printing them.

PRTMBR

Decompresses then prints a member from a listing library.

COPY

Copies a member from an input listing library to an output listing library (generally from Stage 1 to Stage 2), after optionally appending one or more temporary list data sets at the end of the member. You can optionally store a banner at the front of the member in the output library.

DELETE

Deletes a member from the output library.

MBR (mbr-name)

Overrides the default member name (that is, the element name) used by CONLIST. This option can be used with the STORE, PRTMBR, COPY, and DELETE options.

Banner Pages

You can request a banner when using the STORE, PRINT, and COPY options. Banner pages are defined with a C1BANNER DD statement. This statement must specify LRECL=121.

The banner is useful for scanning listings and browsing stored members. The banner includes the following:

- The element name across the top.
- A summary of the processor being run (user ID, date, return code, stage, and so on).
- An itemization of each processor step, up to but not including the CONLIST step, with the return code from each.

Error conditions are reflected in the banner; for example, an error may have occurred within a processor step, or steps were not executed because of condition-code testing.

The STORE Option

The STORE option consolidates and compresses one or more temporary list data sets into a member in the output library defined by the DD statement C1LLIBO.

```
//STEPNAME  EXEC PGM=CONLIST,PARM='STORE'
//C1BANNER    DD DISP=(,PASS,DELETE),
//               UNIT=SYSDA,SPACE=(TRK,(1,1)),
//               DCB=(RECFM=FBA,LRECL=121,BLKSIZE=6171,DSORG=PS)
//C1LLIBO     DD DSN=&C1STAGE.LISTINGS,DISP=SHR
//LIST01      DD DSN=&&COBLST,DISP=(OLD,DELETE)
//LIST02      DD DSN=&&LNKLST,DISP=(OLD,DELETE)
```

The parameters in the previous sample JCL are described next:

C1BANNER

Requests a banner (as illustrated above) be included at the front of the stored member. Omit this statement if you do not want the banner included.

C1LLIBO

Identifies the output listing library to which the new member is written. The output member name is the name of the element being processed. If a member by this name currently exists, it is replaced.

Note: FOOTPRNT=CREATE is not needed on this DD statement, becuase CONLIST footprints automatically.

LIST*nn*

Identifies a listing data set to be stored in C1LLIBO. If you specify more than one LISTnn library, assign the ddnames sequentially (LIST01, LIST02, and so forth). LISTnn data sets are concatenated before they are stored, in order by the nn suffix.

Note: You must specify at least one LISTnn DD statement.

Note: You can override the default member (element) name for the STORE option, using the MBR (mbr-name) option.

The PRINT Option

The PRINT option prints a temporary list data set, optionally generating a banner page before the listing.

```
//STEPNAME  EXEC PGM=CONLIST,PARM='PRINT'
//C1BANNER    DD DSN=&&BANNER,DISP=(,PASS,DELETE),
//               UNIT=SYSDA,SPACE=(TRK,(1,1)),
//               DCB=(RECFM=FBA,LRECL=121,BLKSIZE=6171,DSORG=PS)
//C1PRINT     DD SYSOUT=*,
//               DCB=(RECFM=FBA,LRECL=133,BLKSIZE=0,DSORG=PS)
//LIST01      DD DSN=&&COBLST,DISP=(OLD,DELETE)
//LIST02      DD DSN=&&LNKLST,DISP=(OLD,DELETE)
```

The parameters in the previous sample JCL are described:

C1BANNER

Requests a banner page (as illustrated above). Omit this statement if you do not want the banner to print.

C1PRINT

The output print file.

LIST*nn*

Identifies a listing data set to be printed. If you specify more than one LIST*nn* data set, assign the ddnames sequentially (LIST01, LIST02, and so forth). LIST*nn* data sets are concatenated before they are printed, in order by the *nn* suffix.

Note: You must specify at least one LISTnn DD statement.

The PRTMBR (Print Member) Option

The PRTMBR option decompresses then prints a member from a listing library.

```
//STEPNAME  EXEC PGM=CONLIST,PARM='PRTMBR'
//C1LLIBI     DD DSN=STAGE1.LISTINGS,DISP=SHR
//C1PRINT     DD SYSOUT=*,
//               DCB=(RECFM=FBA,LRECL=133,BLKSIZE=0,DSORG=PS)
```

The parameters in the previous syntax are described:

C1LLIBI

Identifies the input listing library from which a member is being printed. The name of the member to be printed is the name of the element being processed.

C1PRINT

The output print file.

Note: You can override the default member (element) name for the PRTMBR option, using the MBR (mbr-name) option. For example, to specify member mbrname in data set prtmbr.mbr, you would code:

```
//STEPNAME EXEC PGM=CONLIST,PARM='PRTMBR.MBR(mbr-name)'
```

The COPY Option

The COPY option copies a member from an input listing library to an output listing library (generally from Stage 1 to Stage 2), after optionally appending one or more temporary list data sets at the end of the member.

```
//STEPNAME  EXEC PGM=CONLIST,PARM='COPY'
//C1BANNER    DD DSN=&&BANNER,DISP=(,PASS,DELETE),
//               UNIT=SYSDA,SPACE=(TRK,(1,1)),
//               DCB=(RECFM=FBA,LRECL=121,BLKSIZE=6171,DSORG=PS)
//C1LLIBI     DD DSN=STAGE1.LISTINGS,DISP=SHR
//C1LLIBO     DD DSN=STAGE2.LISTINGS,DISP=SHR
//LIST01      DD DSN=&&COBLST,DISP=(OLD,DELETE)
//LIST02      DD DSN=&&LNKLST,DISP=(OLD,DELETE)
```

The parameters in the previous sample JCL are described:

C1BANNER

Requests a banner page. Omit this statement if you do not want the banner included in the output member.

C1LLIBI

Identifies the input listing library from which a member is being copied. The member being copied has the name of the element being processed.

C1LLIBO

Identifies the output listing library to which the copied member is being written (after appending any LISTnn files at the end of the member). The member name is the element name.

LIST*nn*

Identifies a listing data set to be concatenated at the end of the C1LLIBI member before it is written to C1LLIBO. If you specify more than one LISTnn data set, assign the ddnames sequentially (LIST01, LIST02, and so forth). LISTnn data sets are concatenated (in order by the nn suffix) before they are appended to the member.

Note: Specifying a LISTnn DD statement is optional for the COPY option.

Note: You can override the default member (element) name for the COPY option, using the MBR (mbr-name) option.

The DELETE Option

The DELETE option deletes a member from the output library.

```
//STEPNAME  EXEC PGM=CONLIST,PARM='DELETE'
//C1LLIBI     DD DSN=STAGE1.LISTINGS,DISP=SHR
```

The parameters in the previous sample JCL are described:

C1LLIBI

Identifies the input listing library from which a member is to be deleted. The name of the member to be deleted is the name of the element being processed.

Note: You can override the default member (element) name for the DELETE option, using the MBR (mbr-name) option. For example, to specify member mbrname in data set delete.mbr, you would code:

```
//STEPNAME EXEC PGM=CONLIST,PARM='DELETE.MBR(mbrname)'
```

Guidelines When Creating Listings

There are several ways you can approach the creation and handling of listings within CA Endevor SCM processors. The following is the recommended approach.

1. Writing the listings to a data set other than SYSOUT during the processor step
2. Running the CA Endevor SCM CONLIST utility at the end of the job to combine all the listings into a single member of a listing library.

To do this:

1. Initialize each listing data set before you write to it, to ensure that there will be a listing to open in the final (CONLIST) step. You need one such data set for each utility that outputs listings. For information about a utility that initializes data sets, see “BC1PDSIN Utility” in this chapter.
2. In the JCL for the processor utilities, write to the appropriate listing data set--not to SYSOUT.
3. Use the CA Endevor SCM CONLIST utility to condense and combine the listings from all the job steps, and either store them in a designated listing library or print them.

This approach is illustrated by the following compile and link-edit processor.

Note: Userinfo, in the examples that follow, represents information you must supply.

```
//INITLST  EXEC PGM=BC1PDSIN,COND=EVEN
//C1INIT01   DD DSN=&&COBLST,DISP=(,PASS,DELETE),UNIT=SYSDA,
//              SPACE=(TRK,(1,2),RLSE)
//C1INIT02   DD DSN=&&LNKLST,DISP=(,PASS,DELETE),UNIT=SYSDA,
//              SPACE=(TRK,(1,2),RLSE)
//WRITE    EXEC PGM=CONWRITE,PARM='EXPINCL(N)'
//ELMOUT     DD DSN=&&SYSIN,DISP=(,PASS,DELETE),userinfo
//*
//*

//COMPILE  EXEC PGM=IGYCRCTL,COND=(0,NE),PARM=mmm,MAXRC=04
//SYSLIB     DD DSN=userinfo
//SYSIN      DD DSN=userinfo
//SYSLIN     DD DSN=&&SYSLIN,DISP=(,PASS,DELETE),userinfo
//SYSUT1     DD UNIT=SYSDA,userinfo
//*

//SYSUT6     DD UNIT=SYSDA,userinfo
//SYSPRINT   DD DSN=&&COBLST,DISP=(OLD,PASS,DELETE),
//              UNIT=SYSDA,SPACE=(TRK,(5,10),RLSE),
//              DCB=(RECFM=FBA,LRECL=133,BLKSIZE=0,DSORG=PS)
//*

//LINK    EXEC PGM=IEWL,PARM='userinfo,COND=userinfo,MAXRC=0
//SYSLIN    DD DSN=&&SYSLIN,DISP=(OLD,PASS)
//SYSLMOD   DD DSN=userinfo
//SYSUT1    DD UNIT=SYSDA,userinfo
//SYSPRINT  DD DSN=&&LNKLST,DISP=(OLD,PASS,DELETE),
//             UNIT=SYSDA,SPACE=(TRK,(5,3),RLSE),
//             DCB=(RECFM=FBA,LRECL=133,BLKSIZE=0)
//*
```

```
//CONLIST  EXEC PGM=CONLIST,PARM=STORE,COND=EVEN,MAXRC=0
//C1LLIBO    DD DSN=STAGE1.LISTINGS,DISP=SHR
//C1BANNER   DD DSN=&&BANNER,userinfo
//LIST01     DD DSN=&&COBLST,DISP=(OLD,DELETE)
//LIST02     DD DSN=&&LNKLST,DISP=(OLD,DELETE)
//
```

- BC1PDSIN initializes a listing library for use in the COMPILE and LINK steps.
- CA Endevor SCM CONWRITE utility creates a source file by merging all levels of the element.
- COBOL compile writes listings to the file initialized in the INITCOB step.
- Link-edit writes listings to the file initialized in the INITLNK step.
- CA Endevor SCM CONLIST utility creates a CA Endevor SCM banner page and combines the compile and link-edit listings and stores them as a single member in the Stage 1 listings library.

One Alternative

As an alternative, you can write the listings to SYSOUT. If you are running the processor in batch, SYSOUT=* data sets are attached to the message class (MSGCLASS) assigned in the jobcard for the corresponding batch job.

Note: If you are running in foreground, SYSOUT=* data sets are attached to the default SYSOUT class assigned for your user ID. TSO allocates the SYSOUT file when it is first opened, and does not free it until you log off from TSO or issue an explicit TSO FREE command for the file.

This approach is illustrated by the compile and link-edit processor shown next:

```
//*
//WRITE    EXEC PGM=CONWRITE,PARM='EXPINCL(N)'
//ELMOUT     DD DSN=&&SYSIN,DISP=(,PASS,DELETE),userinfo
//*
//COMPILE EXEC PGM=IGYCRCTL,COND=(0,NE),PARM=mmm
//SYSLIB     DD DSN=userinfo
//SYSIN      DD DSN=userinfo
//SYSLIN     DD DSN=&&SYSLIN,DISP=(,PASS,DELETE),userinfo
//SYSUT1     DD UNIT=SYSDA,userinfo
//*
//SYSUT6     DD UNIT=SYSDA,userinfo
//SYSPRINT   DD SYSOUT=*
//*
//*
//LINK     EXEC PGM=IEWL,PARM='userinfo,COND=userinfo,
//SYSLIN     DD DSN=&&SYSLIN,DISP=(OLD,PASS)
//SYSLMOD    DD DSN=userinfo
//SYSUT1     DD UNIT=SYSDA,userinfo
//SYSPRINT   DD SYSOUT=*
//*
```

- CA Endevor SCM CONWRITE utility creates a source file by merging all levels of the element.
- COBOL compile writes listings to SYSOUT.
- Link-edit writes listings to SYSOUT.

Another Alternative

This approach is similar to the recommended approach but uses the PRINT option of the CONLIST utility, so the listings are printed instead of stored. The JCL for this approach is the same as that for the recommended approach with the exception of the CONLIST step, which is shown below.

```
//*
//*
//CONLIST EXEC PGM=CONLIST,PARM=PRINT,COND=EVEN,MAXRC=0
//C1PRINT   DD SYSOUT=*,
//             DCB=(RECFM=FBA,LRECL=133,BLKSIZE=0,DSORG=PS)
//C1BANNER  DD DSN=&&BANNER,userinfo
//LIST01    DD DSN=&&COBLST,DISP=(OLD,DELETE)
//LIST02    DD DSN=&&LNKLST,DISP=(OLD,DELETE)
 *
```

The CONPARMX Utility

The *CONPARMX utility* lets administrators reduce processor complexity and the number of processor groups. CONPARMX dynamically builds execution parameters for processor programs such as compilers or the linkage editor. The build process uses the symbols defined in the processor and one or more of the following groups of options, which may or may not be concatenated:

- Default options
- Processor group options
- Element options

Using this utility avoids the need to create and update large numbers of processors or processor groups with different input parameters.

How to Create a CONPARMX Processor JCL Step

CONPARMX enables you to use fewer processors and processor groups. To use the utility, do the following:

- Write your CONPARMX processor JCL.
- Create the options members referenced by CONPARMX's PARM= parameters. Follow the syntax rules when you create the options members.

  Note: We recommend that you store the options members in CA Endevor SCM and monitor the members as part of an element's input component list.

CONPARMX - Dynamically Build a Parameter String

The CONPARMX utility can reduce the complexity of your processors by dynamically building execution parameters for a specified program. CONPARMX builds a parameter string and optionally invokes a program depending on how you code the utility.

The processor step JCL for the CONPARMX utility is shown next.

```
//CONP     EXEC PGM=CONPARMX,MAXRC=n,
//          PARM=(parm1,'(parm2)',parm3,parm4,parm5,
//          '(parm6)','parm7','parm8')
//STEPLIB  DD DSN=xxxxxx.loadlib.if.required,DISP=SHR
//PARMSDEF DD DISP=SHR,DSN=&PFX..PARMS,
//            MONITOR=COMPONENTS,ALLOC=PMAP
//PARMS    DD DSN=&&PPARMS,DISP=(,PASS),
//            SPACE=(CYL,(1,1)),
//            DCB=(RECFM=FB,LRECL=80,BLKSIZE=6160)
//PARMSMSG DD SYSOUT=*
//* Additional DD statements as required by the parm1 program
```

The JCL statements and variables used by the CONPARMX utility are as follows:

CONP EXEC PGM=CONPARMX,MAXRC=n,
PARM=(parm1,'(parm2)',parm3,parm4,parm5,*'(parm6)','parm7','parm8'*)

Specifies the JCL EXEC statement that executes the CONPARMX utility program. This step requires the following parameters.

MAXRC=*n*

Specifies the maximum return code value for the parm1 value. If the parm1 program is not called by CONPARMX, the MAXRC should be set to 0.

PARM=

Specifies the parameters the utility used. The total length of the output options string using all options values for CONPARMX parm2 through parm6 can be no more than 512 characters (the CONPARMX limit). This limit is the limit supported by the parm1 program.

parm1

Specifies a *program name* for which the options parameter string is to be built. This program name is used to find the options values within the options members. It can be any value up to eight characters. If CONPARMX is going to invoke the program (parm8 = N), this value must also be the name of the program to be run.

Note: When IGYCRCTL is passed as the program name, the parameters will have the prefix CBL. This is to allow those parameters to be passed through SYSIN DD. If this is not wanted, use another program name.

parm2

(Optional) Specifies the *processor symbolic* that identifies the options to be included first in the output options string. This value must be within parentheses in the CONPARMX parameter list. Parm2 is usually specified with a processor symbolic and can contain 0 to *n* options. Its value is placed first and is always included in the output options string. The parentheses are not included in the output.

parm3

Specifies the *default options member name*. The options values found in this member to match the parm1 value (the program name) are added in the output parameter string after the parm2 options values. This member is read from the PARMSDEF DD statement.

Parm7 determines whether the options members are concatenated. For more information, see parm7.

parm4

Specifies the *processor group options member name*. A processor group options member can be created for each processor group. In the member, only the options that are to be overridden for a particular processor group should be included, unless parm7 is set to N. This member is read from the PARMSDEF DD statement.

Parm7 determines whether the options members are concatenated. For more information, see parm7.

parm5

Specifies the *element options member name*. An element options member can be created for any element in the CA Endevor SCM inventory. In this member, only the options that are to be overridden specifically for this element should be included, unless parm7 is set to N. This member is read from the PARMSDEF DD statement.

Parm7 determines whether the options members are concatenated. For more information, see parm7.

parm6

(Optional) Specifies the *processor symbolic* that identifies the options to be included last in the output parameter string. This value must be specified within parentheses in the CONPARMX parameter list. Parm6 can contain 0 to *n* options and its value is placed last and always is included in the output options string. The parentheses are not included in the output.

parm7

(Optional) Specifies whether to concatenate the options values in the output parameter string. Valid values follow:

Y – (Default) Specifies that the contents of parm2 through parm6 are to be concatenated.

N – Specifies that the content of parm5, parm4, *or* parm3 are to be included between parm2 and parm6. The first options member found to have a record that matches the parm1 program name is used. The options members are searched until the match is found beginning with parm5, then parm4, and then parm3.

parm8

(Optional) Specifies whether to build a parameter string to a file. Valid values follow.

Y – Writes the parameter string to the PARMS DD statement and the parm1 program is *not* invoked. This data set can be used as input to a program step that follows.

N – (Default) Passes the parameter string to the parm1 program and invokes the program. The parameter string is *not* written to the PARMS DD statement.

STEPLIB DD DSN=*xxxxxx*.loadlib.if.required,DISP=SHR

Specifies the library that should be searched for the parm1 program executed by CONPARMX. Required only if parm8=N indicating that the parm1 program is to be invoked.

PARMSDEF DD DISP=SHR,DSN=&PFX..PARMS,

MONITOR=COMPONENTS,ALLOC=PMAP

Specifies a concatenation of one or more FB-80 PDS or PDSE data sets. The parm3, parm4, and parm5 values point to options members in this data set. Options members do not have to exist or can exist in one or more of the concatenated data sets. The first found options member is used. During processing, if an options member does not exist, it is skipped and processing continues.

MONITOR=COMPONENTS

(Optional) Instructs the CA Endevor SCM Automated Configuration option to monitor the &PRX..PARMS data set for input or output components and to build a component list with the information captured.

ALLOC=PMAP

(Optional) The ALLOC=PMAP processor DD statement parameter can be specified to indicate that this data set concatenation varies with the CA Endevor SCM location from which the processor is being executed.

PARMS DD DSN=&&PPARMS,DISP=(,PASS),

SPACE=(CYL,(1,1)),

DCB=(RECFM=FB,LRECL=80,BLKSIZE=6160)

Specifies the data set name to which the parameter string built by the utility is written. The PARMS DD statement is only required if the parm8 value in the CONPARMX parameter string is set to Y.

Note: For program ASMA90, this data set is read from the ASMAOPT DD statement. For COBOL and PL/I compiler programs, place this data set first within the SYSIN DD statement.

PARMSMSG DD SYSOUT=*

Specifies that the output options string value is written to this data set for diagnostic purposes. The PARMSMSG DD statement is not required. The data set DCB attributes are FB-80-32720, and the output options string is written to the data set 72 bytes per line until the string is exhausted. A maximum of 512 bytes of data (the maximum size of the output options string) or eight lines (7 x 72 = 504 bytes plus 8 bytes on the eighth line) are written to the data set.

How CONPARMX Creates a Parameter String

CONPARMX builds the parameter string and determines whether to execute the program based on how you code the CONPARMX JCL as follows:

- The following parameters can be specified for CONPARMX. These parameters are passed to CONPARMX on the EXEC statement using the PARM keyword as follows:

```
//STEP001 EXEC PGM=CONPARMX,PARM=(parm1,parm2,parm3,parm4,parm5,parm6,parm7,parm8)
```

- CONPARMX produces an output options string from the options values in parm2 through parm6, subject to the following parameters:
  - Parm1 specifies the processor program for which the parameters are built.
  - Parm7 is optional and specifies if the parameters are concatenated. The default forces concatenation.
  - Parm8 is optional and specifies whether the parm1 processor program will be invoked or if the parameters are stored for a future processor step. The default invokes the program.
- The PARMSDEF DD statement on the CONPARMX JCL reads the options members whose names are coded in parm3, parm4, and parm5. PARMSDEF finds the matching record in each options member by matching the record to the name of the program specified in parm1. The options members contains records that start with a program name followed by a parameter string. For example, the following record matches parm1 IGYCRCTL and specifies the parameters COBOL, LANGLVL(2), and NOSOURCE:

  ```
  IGYCRCTL    'COBOL3,LANGLVL(2),NOSOURCE'
  ```

- For parm3, parm4, and parm5, only the options in the matching records are used as input to building the output parameter string. Whether the options for parm3, parm4, or parm5 are included in the output parameter string depends on the parm7 concatenation specification.

Example: CONPARMX Builds a Parameter String and Invokes a Program

Sample processor step JCL for the CONPARMX utility is shown next. This JCL invokes program ASMA90 and passes a parameter string to the program.

The parameter string is built as follows:

- The string begins with the content of &PARMA.
- Then $$$$DFLT is read and the program name ASMA90 is searched for within the member. If there is a match for ASMA90, the options values are added to the output options string.
- Then &C1PRGRP and &C1ELEMENare processed the same way.
- The options values for &PARMZ are added last in the output options string.

```
//CONP     EXEC PGM=CONPARMX,MAXRC=4,
//          PARM=(ASMA90,'(&PARMA)',$$$$DFLT,&C1PRGRP,&C1ELEMENT,
//          '(&PARMZ)','Y','N')
//STEPLIB  DD DSN=ASMA90.loadlib.if.required,DISP=SHR
//PARMSDEF DD DISP=SHR,DSN=&PFX..PARMS,
//            MONITOR=COMPONENTS
//PARMS    DD DSN=&&PPARMS,DISP=(,PASS),
// SPACE=(CYL,(1,1)),
// DCB=(RECFM=FB,LRECL=80,BLKSIZE=6160)
//PARMSMSG DD SYSOUT=*
//* Additional DD statements as required by the parm1 program
```

Example: CONPARMX Builds a Parameter String and Writes It to PARM DD

This example JCL builds a parameter string and writes it to PARM DD for the program IGYCRCTL.

In the JCL, see the comments to the right that identify parm1, parm2, parm6, parm7, and parm8:

```
//COBPROC PROC PARMTAIL=XYZ
//STEP001  EXEC PGM=CONPARMX,
// PARM=(IGYCRCTL,     (invocation pgm name)
// ('ABC'),           (parm2 – constant parm data)
// $$$$DFLT,  &C1PRGRP.P, &C1ELEMENT.P,
// '(&PARMTAIL)',      (parm6 – last parm)
// 'Y','Y')            (parm7/8 directions)
//PARMSDEF DD DSN=input.parameter.pds,DISP=SHR
//PARMS    DD DSN=&&parmout,DISP=(NEW,PASS),…
```

The options members specified in the JCL are as follows:

- Parm3, $$$$DFLT, is the member that contains the default options.

  This example assumes that parm3, $$$$DLFT, contains the following records:

  ```
  ASMA90    'LIST…      '
  IGYCRCTL   'COBOL3,LANGLVL(2),NOSOURCE'
  IEWL 'AMODE(31)'
  IEWL 'RMODE(ANY)'
  ```

- Parm4, &C1PRGRP.P, is the member that contains the processor group override parameters.
- Parm5, &C1ELEMENT.P, is the member that contains the element override parameters.

This example JCL will result in a PARMS output DD that contains the following parameter string for IGYCRCTL:

```
ABC
COBOL3,LANGLVL(2),NOSOURCE
Parameters defined by the Processor member
Parameters defined by the Element member
XYZ
```

This parameter string includes the following:

- ABC- The constant parameter data specified by parm2.
- COBOL3,LANGLVL(2),NOSOURCE - The record in parm3 options member $$$$DLFT found to match IGYCRCTL.
- Parameters defined by the Processor member - The actual parameters defined by parm4, &C1PRGRP.P, would depend on the record found to match IGYCRCTL.
- Parameters defined by the Element member - The actual parameters defined by parm5, &C1ELEMENT.P, would depend on the record found to match IGYCRCTL.
- XYZ- The constant parameter data specified by parm6.

If parm8 in this example was N, instead of Y, the parameter string would not have been written to the PARMS output DD. Instead, IGYCRCTL would have been invoked directly with the following parameter string:

```
//CONPARMX EXEC PGM=IGYCRCTL,
// PARM='ABC,COBOL3,LANGLVL(2),NOSOURCE,
// procgrp-parms, elm-parms, XYZ'
```

Return Codes for CONPARMX

The following return codes can be issued during CONPARMX processing:

0– Processing successful.

16– WTO message issued with specific error condition.

All other return codes are issued by the parm1 program when called.

Options Member Syntax Rules

The options members contain the options values that can be passed by the PARM parameter to the CONPARMX utility. The syntax for the options members follows:

```
Column 1
V
program = 'options values[,]' [+ -]
          ["options values[,]"] [+ -]
               .
               .
               .
          ['options values[,]']
```

program

Specifies a program name and must begin in column 1.

options values

Specifies the options values that can be passed by the PARM parameter to the CONPARMX utility. The following rules apply to options values:

- Bracketed […] elements in the syntax diagram are optional.
- The options values can be enclosed within single or double quotes, can span multiple lines, and optionally end with a comma (,).
- A plus sign (+) or dash (-) at the end of a line indicates continuation of the options values on the next line. Only enter one of the continuation characters.
- An options values continuation line can begin in any column.
- A blank in column 1 is treated as a comment line, unless the previous line contains a continuation character.
- An asterisk (*) in column 1 always indicates a comment line and is ignored.

- Single or double quotes can be used to enclose the options member's options values. If an options value requires single quotes, enclose the value in double quotes. For example:

```
******************************** Top of Data ************************
----+----1----+----2----+----3----+----4----+----5----+----6----+----7--
CCNDRVR = 'ARCH(5),EXPMAC,RENT,'+
           "SEARCH(//'SYS1.SIEAHDR.H'),"+
           'RENT,'
******************************** Bottom of Data *********************
```

- The total length of the output options string using all options values coded for CONPARMX parm2 through parm6 must be less than 512 characters (CONPARMX limit) or the limit that is supported by the parm1 program, whichever is less.

Options Member Examples

Example: Sample Default Options Member

This example assumes this sample defaults options member:

```
******************************** Top of Data ************************
----+----1----+----2----+----3----+----4----+----5----+----6----+----7--
ASMA90 = 'RENT,TERM,XREF(SHORT),USING(MAP,WARN(13)),LIST(133),'
IGYCRCTL = 'OBJECT,APOST,AWO,DATA(24),FASTSRT,LIB,FLAG(W),'+
           'RESIDENT,LIST,RENT,TRUNC(BIN),'+
           'NODBCS,SOURCE,MAP,NOSEQ,XREF,NONUMBER,LIST,'
IEWL = 'LIST,MAP,RENT,REUS,NOLET,XREF,SIZE=(256K,64K),'
******************************** Bottom of Data *********************
```

This defaults options member would display this options string for ASMA90:

```
RENT,TERM,XREF(SHORT),USING(MAP,WARN(13)),LIST(133)
```

It would display this options string for IGYCRCTL:

```
OBJECT,APOST,AWO,DATA(24),FASTSRT,LIB,FLAG(W),RESIDENT,LIST,RENT,TRUNC(BIN),
NODBCS,SOURCE,MAP,NOSEQ,XREF,NONUMBER,LIST
```

Example: Sample Processor Group and Element Options Members

This example assumes that the parm7 value is set to Y. When parm7 is set to Y (default), for the processor group options member, the utility only includes options that are overridden from the default options member. For the element options member, the utility only includes options that are overridden from the processor group, default options members, or both.

Note: When parm7 is set to N, the utility includes all necessary options values. The options values that are first found to match the program name are used and placed in the output options string.

This example assumes this is the default options member:

```
******************************** Top of Data ************************
----+----1----+----2----+----3----+----4----+----5----+----6----+----7--
ASMA90 = 'RENT,TERM,XREF(SHORT),USING(MAP,WARN(13)),LIST(133),'
IGYCRCTL = 'OBJECT,APOST,AWO,DATA(24),FASTSRT,LIB,FLAG(W),'+
           'RESIDENT,LIST,RENT,TRUNC(BIN),'+
           'NODBCS,SOURCE,MAP,NOSEQ,XREF,NONUMBER,LIST,'
IEWL = 'LIST,MAP,RENT,REUS,NOLET,XREF,SIZE=(256K,64K),'
******************************** Bottom of Data ***********************
```

This example assumes this member is the processor group options member:

```
******************************** Top of Data ************************
----+----1----+----2----+----3----+----4----+----5----+----6----+----7--
ASMA90 = 'NORENT,'
******************************** Bottom of Data ***********************
```

Using this default options member with this processor group options member would result in the following output options string for ASMA90:

```
RENT,TERM,XREF(SHORT),USING(MAP,WARN(13)),LIST(133),NORENT
```

ASMA90 and most compiler programs use the last option specified if conflicting options are used. Consequently, in this example, the NORENT option overrides the RENT option specified earlier in the options list.

The CONRELE Utility

The CONRELE utility allows you 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.

The CONRELE utility parses and processes input data. CONRELE accepts user syntax from the NDVRIPT 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.

Note: You must use the CONRELE utility with an active component list (the component list for the element being processed).

You must include the CONRELE utility as a processor step and you must provide the input. You can include the optional NOECHO parameter to suppress CONRELE messages from appearing in C1MSGS1 output. The following sample processor JCL executes CONRELE and suppresses the CONRELE messages:

```
//STEPxx   EXEC  PGM=CONRELE,PARM='NOECHO'
//NDVRIPT  DD   DSN=&user.data.set,DISP=SHR
```

CONRELE Utility Commands

The CONRELE utility accepts the following commands from the NDVRIPT file:

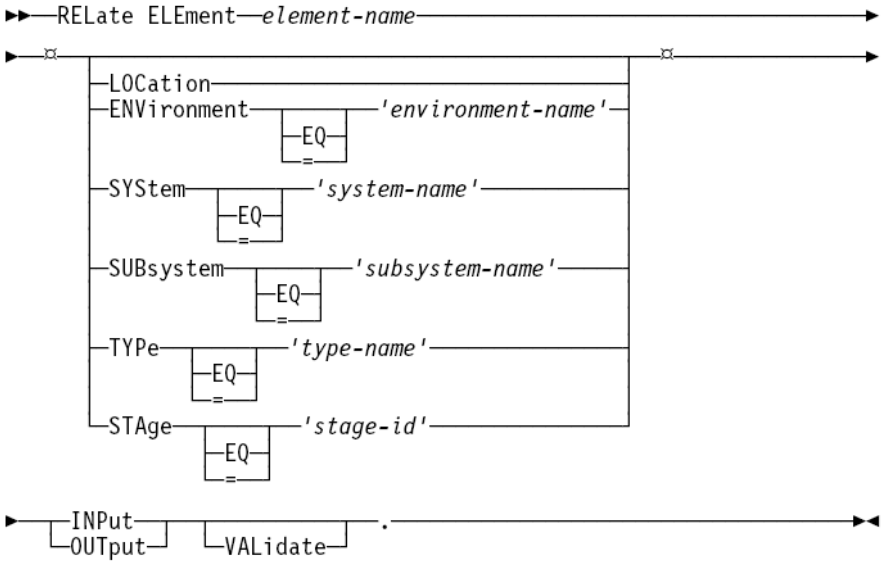
- RELATE ELEMENT-Relates an element to another element.
- RELATE MEMBER-Relates a data set member to an element.
- RELATE OBJECT-Relates an object such as a pathname or a filename for an object on another platform to an element.
- RELATE COMMENT-Adds comments to a component list.
- SET ERROR RETURN CODE-Returns an error code when the CONRELE utility finds errors in the input syntax (default 8).

The syntax for these commands follows.

RELATE ELEMENT Command Syntax

The RELATE ELEMENT command syntax is shown next.

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



element-name

The name of the element, maximum length of 10.

LOCATION

(Optional) Location of the element-name. If you do not include the location in your syntax the location defaults to the target location of the current CA Endevor SCM element.

INPUT

Input component

OUTPUT

Output component

VALIDATE

Verifies the elements' presence at the specified location in CA Endevor SCM. This is optional.

If the element doesn't exist, the RELATEd element is rejected.

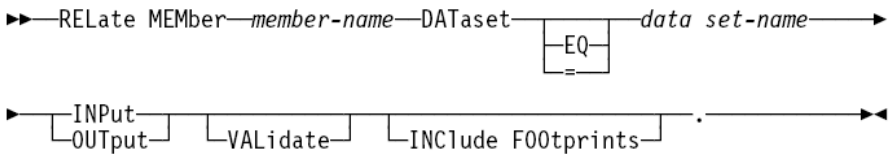
If you specify the VALIDATE option and the validation fails, the return code is set according to the SET ERROR RETURN CODE syntax.

- When the VALIDATE option is specified, the location names (environment, system, etc.) cannot be wildcarded. If they are omitted, the location names associated with the target element are used.
- When the VALIDATE option is NOT specified, CONRELE stores the location information "as specified". Omitted location information is taken from the target element's location.

Note: Package component validation only validates RELATED ELEMENTS, (not members, objects or comments) which were related using the CONRELE VALIDATE option.

RELATE MEMBER Command Syntax

The RELATE MEMBER command syntax is shown next:



member-name

The name of the member in a data set, the maximum member-name length is 10. A blank member is valid.

data set-name

The name of the data set that contains the member-name.

INPUT

Input component

OUTPUT

Output component

VALIDATE

CA Endevor SCM determines whether the specified data set exists. This is optional.

If you specify the VALIDATE option, CA Endevor SCM determines whether or not the data set exists.

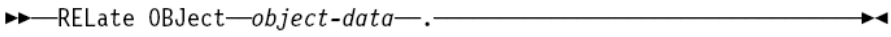
If the validation fails the default SET ERROR RETURN CODE is displayed. If you do not specify the VALIDATE option no validation occurs.

INCLUDE FOOTPRINTS

(Optional) If the member is footprinted, the footprint is stored in the component list.

RELATE OBJECT Command Syntax

The RELATE OBJECT command syntax is shown:

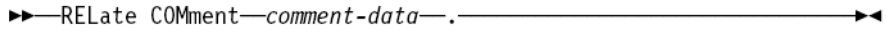


object-data

The name of object-data. The object-data name is a maximum length of 70 bytes. The object-data is not verified. Related objects are stored in a first-in-first-out (FIFO) sequence.

RELATE COMMENT Command Syntax

The RELATE COMMENT command syntax is shown next:

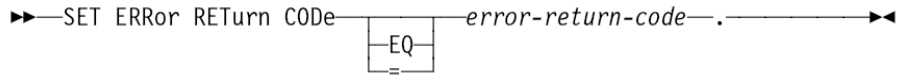


Comment-data

The comments should be enclosed in quotations marks and are stored in a first-in-first-out (FIFO) sequence. You can include an unlimited number of comments. The comments are not verified but you can search on the text.

SET ERROR RETURN CODE Command Syntax

The SET ERROR RETURN CODE command syntax is shown:



error-return-code

Specifies the error return code when the Validate option fails.

Default: 8.

Example of CONRELE Syntax

An example of the CONRELE syntax is shown:

```
SET ERROR RETURN CODE 0000.
 RELATE ELEMENT BGSQL600
   LOCATION
    ENVIRONMENT = TEST
    SYSTEM   = FINANCE
    SUBSYSTEM  = AP
    TYPE    = BGLOAD2
    STAGE    = 1
   INPUT.

 RELATE ELEMENT BGSQL723
   LOCATION
    ENVIRONMENT = TEST
    SYSTEM   = FINANCE
    SUBSYSTEM  = AP
    TYPE    = BGLOAD3
    STAGE    = 1
   INPUT.

 RELATE ELEMENT BGSQL601
   LOCATION
    ENVIRONMENT = TEST
    SYSTEM   = FINANCE
    SUBSYSTEM  = AP
    TYPE    = BGLOAD2
    STAGE    = 1
   OUTPUT.

 RELATE ELEMENT BGSQL64
   LOCATION
    ENVIRONMENT = TEST
    SYSTEM   = FINANCE
    SUBSYSTEM  = AP
    TYPE    = DB2COB3
    STAGE    = 1
   OUTPUT.
```

```
 RELATE ELEMENT BGSQL65
   LOCATION
    ENVIRONMENT = TEST
    SYSTEM   = FINANCE
    SUBSYSTEM  = AP
    TYPE    = DB2COB
    STAGE    = 1
  OUTPUT
 VALIDATE.
 RELATE MEMBER BC1PSQL1
    dataset = 'JSMITH.SRCLIB'
    INPUT.

 RELATE MEMBER BGSQL60
    dataset = 'DA1BG10.DBRMLIB'
    INPUT
    VALIDATE
    INCLUDE FOOTPRINTS.

RELATE MEMBER BC1PSQL3
    dataset = 'JSMITH.SRCLIB'
    OUTPUT.

RELATE MEMBER BGSQL70
    dataset = 'DA1BG10.DBRMLIB'
    INPUT
    VALIDATE
    INCLUDE FOOTPRINTS.

RELATE OBJECT 'D;\ENDEVOR\TEMP.DOC'.
 RELATE OBJECT 'D;\ENDEVOR\TEMP.DOC2'.
 RELATE OBJECT 'D;\ENDEVOR\TEMP.DOC3'.
 RELATE OBJECT 'D;\ENDEVOR\TEMP.DOC4'.
 RELATE OBJECT 'D;\ENDEVOR\TEMP.DOC5'.
 RELATE OBJECT 'D;\ENDEVOR\TEMP.DOC6'.
 RELATE OBJECT 'D;\ENDEVOR\TEMP.DOC7'.
 RELATE OBJECT 'D;\ENDEVOR\TEMP.DOC8'.
 RELATE OBJECT
'<-------------------------------------------------------------.'
 .
 RELATE COMMENT 'THIS IS A FREE FORM TEST'.
 RELATE COMMENT 'THIS IS A FREE FORM TEST2'.
 RELATE COMMENT 'THIS IS A FREE FORM TEST3'.
 RELATE COMMENT 'THIS IS A FREE FORM TEST4'.
 RELATE COMMENT 'THIS IS A FREE FORM TEST5'.
 RELATE COMMENT 'THIS IS A FREE FORM TEST6'.
 RELATE COMMENT 'THIS IS A FREE FORM TEST7'.
 RELATE COMMENT 'THIS IS A FREE FORM TEST8'.
 RELATE COMMENT 'THIS IS A FREE FORM TEST9'.
```

```
RELATE COMMENT 'THIS IS A FREE FORM TEST10'.
RELATE COMMENT 'THIS IS A FREE FORM TEST11'.
```

The CONSCAN Utility

The CONSCAN utility allows you to identify additional Automated Configuration Management (ACM) relationships between CA Endevor SCM Elements and objects contained within the element, such as data set or program names contained within a JCL jobstream or dynamic program call statements within a COBOL program.

User-defined selection criteria and scan rules are applied against the element source and CONRELE control statements are produced by the CONSCAN utility. These statements can be passed to the CONRELE utility, which updates the ACM component data for an element.

Note: For more information about the CONRELE utility, see The CONRELE Utility (see page 100).

Once these relationships are established, the element display component options or the ACMQ facility can be used to view this information in addition to standard ACM input and output component data.

Although CONSCAN can be executed as a stand-alone utility, it is intended to be executed as a step in a CA Endevor SCM processor, followed by a CONRELE step. If used as a stand-alone utility, none of the CA Endevor SCM processor symbolic parameters (&C1ELEM, etc.) are available and the processor IF, THEN, ELSE logic cannot be utilized.

CONSCAN Parameter Data Set

A user-defined data set must be created to hold the input control parameters necessary to execute this utility. This data set must be a card image file defined as a fixed blocked file with a record length of 80. It is recommended this file be a partitioned data set (PDS).

This parameter data set contains selection criteria and scan rules. It is recommended you create one PARMSCAN member in this library for each element TYPE that utilizes the CONSCAN utility.

PARMSCAN Parameter Statements

When coding PARMSCAN statements, keep these conditions in mind:

- All lines with an asterisk in column 1 are considered comments and are ignored.
- Positions 2-72 are used for control statement syntax in free format.

- Lines that end with a comma are continued.
- Literals that contain special characters must be enclosed within single quotes.
- Quotes within literals must be doubled.
- Only one statement is allowed per line.

CONSCAN validates all the PARMSCAN input for proper syntax. If a syntax error is detected, a return code of 8 is returned and error messages are written to the report file.

CONSCAN will not scan any source until all the statements in the PARMSCAN input pass the syntax checking rules.

The contents of the PARMSCAN input consists of three logical parts:

- Excluding source data
- Selecting source data
- Scan rule processing

Excluding Source Data

Exclusion groups define conditions that cause source data to be ignored. An unlimited number of exclusion groups are allowed per PARMSCAN member, but none are required. A COMMENT statement is used to identify the beginning of an exclusion group. A FIND statement, which defines element source exclusion criteria follows. If a match is found, the input source data is ignored. This may be the entire record or selected data within a record. An END must follow the FIND statement to identify the end of the data to be excluded and to terminate the exclusion group.

Exclusion Group Syntax

```
COMMENT
FIND1 STRING='string',POS=ANY/'nn'
END1 CARD/STRING='string',POS=ANY/'nn'
```

COMMENT

Indicates the beginning of an exclusion group.

FIND1

Indicates the beginning of the FIND statement.

STRING=

Indicates to scan this record for a string. The search is case sensitive; the search will only find a string value with a matching case.

'string',

Specified string. One to eight characters can be specified.

POS

Identifies the position to search for the delimiter string.

ANY

The specified string may occur anywhere in the source.

nn

The specified string must occur after this position in the source.

END1

END1 Indicates the end of an exclusion group.

CARD

Ignore from the FIND POS to the end of the source record.

STRING

Delimiter for the data to be ignored.

*'string'***,**

Indicates to include any characters following this string. 1-8 characters can be specified.

POS

Identifies the position to search for the delimiter string.

ANY

The specified string may occur anywhere in the source.

nn

The specified string must occur after this position in the source.

Examples: Exclusion Groups

Example 1: In this assembler example, CONSCAN searches for source records containing an asterisk '*' in position 1. If found, the remainder of this record is ignored.

```
COMMENT
FIND1 STRING='*',POS=1
END1  CARD
```

Example 2: In this JCL example, CONSCAN searches for source records containing '//*' starting in position 1. If found, the remainder of the record is ignored.

```
COMMENT
FIND1 STRING='//*',POS=1
END1 CARD
```

Example 3: In this example, CONSCAN searches for source records containing a '/*' followed by a '*/' in any position of the record. If found, these characters and any characters in between are ignored.

```
COMMENT
FIND1 STRING='/*',POS=ANY
END1 STRING='*/',POS=ANY
```

Selecting Source Data

Selection groups specify the conditions that are used to select source data. An unlimited number of selection groups are allowed per PARMSCAN member. A minimum of one group is required.

These are the required statements, and they must be in order:

1. SCANTYPE-Identifies the beginning of a selection group.
2. FIND-Defines the element source selection criteria. One FIND statement (FIND1) is required and an optional second FIND statement (FIND2) can be specified per group. If FIND1 and FIND2 are present, both conditions must be true (treated as an AND condition) for the source data to be selected.

3. START- Follows the last FIND statement and identifies the location of the data to extract and is placed on the generated CONRELE control statement. Only one START statement is allowed.
4. END-Must follow the START statement to terminate the extracted string and to terminate the selection group. One END statement is required (END1) and an optional second END statement (END2) can be specified per group. If END1 and END2 are present, either condition can be true (treated as an OR condition) for the termination to take place.

If a match is found against the FIND criteria, the data specified by the START criteria is extracted. There can be more than one match per input record. CONRELE control statements are generated for each match found. These control parameters are written to the data set specified on the ACMRELE DD statement and to the report file, SCANPRT.

Selection Group Syntax

```
SCANTYPE type
FINDn keyword,
   STRING='string'POS=nn/ANY
START TYPE=type,PARM=parm
ENDn TYPE=type,PARM=parm
```

SCANTYPE

Indicates the beginning of a selection group.

type

Identifies the type of relationship and is used to generate the CONRELE RELATE control statements. Valid values are:

- MEMBER
- ELEMENT
- OBJECT
- COMMENT

CONSCAN attempts to determine the type of relationship existing in the data. You may have specified a type of OBJECT, but the control statements are generated with a type of COMMENT. If you specify the type MEMBER and the FIND string is 'DSN=' several checks are performed. If no member exists in the extracted string, CONSCAN determines the length of the data set name and the CONRELE control statements are generated with a type of OBJECT.

FIND

Indicates the beginning of a FIND statement.

n

The valid values for n are 1 or 2. FIND2 cannot exist unless FIND1 exists.

keyword

Optional keywords used to identify additional information related to the string. Only one keyword is allowed:

AFTER

Used in conjunction with the POS parameter. Directs CONSCAN to begin the search for the string after POS=*nn*.

WORD

The string must be a word surrounded by spaces, parentheses () or greater than/less than symbols >< or preceded or followed by one of the following symbols: .,;+-/*.

REJECT

Specified on a FIND2 statement. Allows the user to code a FIND1 and not FIND2 condition. If FIND1 and FIND2 match and REJECT is present on the FIND2 statement, the source data is ignored. For REJECT to work properly, two selection groups are required and the FIND1 statement must be identical.

Note: For an example of the REJECT parameter, see Example 3.

STRING=

Indicates to start scanning this record for a string. The search is case sensitive; the search will only find a string value with a matching case.

'string',

Specified string. One to eight characters can be specified.

POS

Identifies the position to search for the delimiter string.

ANY

The specified string may occur anywhere in the source.

nn

The specified string must occur after this position in the source.

START

Defines where to start the collection of data when the FIND criteria is true. If two FIND statements are specified and both match, the start criteria are relative to the FIND2 statement.

TYPE=

Identifies the direction of the start collection. Valid values are:

- FORW
- BACK
- STRG
- DFLT

PARM=

Works in conjunction with the TYPE keyword.

parm

PARM defines the position to start the collection of data characters as follows:

- After the last character found by the FIND statements when TYPE is FORW.
- Before the first character found by the FIND statements when TYPE is BACK.

Note: The collection of data starts at the first non-blank character.

- If TYPE is STRG, specifies the number of characters after the FIND string.
- If TYPE is DFLT, starts the collection at the first not blank character after the FIND string.

END

Indicates the START collection string delimiter. When this condition is met, the string collection is terminated.

The valid values are 1 and 2. END2 cannot be specified unless END1 is specified. Code all END statements after the START statement. If END1 and END2 are present, they are treated as an OR condition.

TYPE=

Identifies the type of delimiter.

type

Works in conjunction with the PARM keyword. Valid values are:

- CHAR
- LENG
- SPAC
- STRG

PARM=

Works in conjunction with the TYPE keyword.

parm

Specifies:

- If TYPE is CHAR, which character(s) ends the collection string
- If TYPE is LENG, the length of the collection string.
- If TYPE is SPAC, PARM is ignored.
- If TYPE is STRG, specifies the number of additional strings to collect after the initial string before termination of the collection. Each string must be separated by a space. For example, if a START string is detected and PARM=3 is specified, the initial string plus three additional strings are collected. CONRELE control statements are generated for each of the four strings.

Generated CONRELE Control Statements

CONRELE control card statements are generated as output from CONSCAN. This file can be fed into the CONRELE utility and added as additional component information. As described above, four types of components can be created; MEMBER, ELEMENT, OBJECT, and COMMENT. A sample CONRELE control statement output is shown next:

```
RELATE COMMENT
  'WS-ABEND-PGM'
 .
 RELATE ELEMENT DTESUB
  LOCATION
  ENVIRONMENT = ' '
  SYSTEM = ' '
  SUBSYSTEM = ' '
  TYPE = ' '
  STAGE = ' '
  INPUT
 .
 RELATE MEMBER SCANCBL
   data set='iprfx.iqual.CSIQOPTN'
   INPUT
 .
 RELATE OBJECT
   'iprfx.iqual.LOADLIB'
```

Selection Group Examples

Example 1

In this JCL example, CONSCAN searches for source records containing data set or member names. The data set name begins in the first position after the FIND string (DSN=) and is terminated by either a space or a comma.

Source Statements	CONSCAN Statements	CONRELE Statements
//DD DSN=SYS1.PROCS(ABC) //DD DSN=SYS1.PROC2II	SCANTYPE MEMBER FIND1 STRING='DSN=', POS=ANY START TYPE=DFLT END1 TYPE=SPAC END2 TYPE=CHAR, PARM=','	RELATE MEMBER ABC DSN='SYS1.PROCS' INPUT. RELATE OBJECT DSN='SYS1.PROC2II'

Example 2

In this COBOL example, CONSCAN searches for source records containing dynamically called programs. The program name is the next word following the search string of CALL and terminated by a space or a comma.

Source Statements	CONSCAN Statements	CONRELE Statements

CALL DTESUB.	SCANTYPE ELEMENT
	FIND1 STRING='CALL',
	  POS=ANY
	START TYPE=DFLT
	END1 TYPE=SPAC
	END2 TYPE=CHAR,
	  PARM='.'
		RELATE ELEMENT DTESUB
		  LOCATION
		  ENVIRONMENT = ' '
		  SYSTEM = ' '
		  TYPE = ' '
		  STAGE = ' '
		  INPUT.

Example 3

In this JCL example, CONSCAN searches for source records containing PROC names. This requires two selection groups. The first selection group ensures program names are not selected. The second selection group selects the processors. In both cases, the search is terminated by a space.

Note: The FIND1 statement is identical in both selection groups.

Source Statements	CONSCAN Statements	CONRELE Statements

//PROC EXEC NDVR
//EXEC PGM=ABC
	SCANTYPE ELEMENT
	FIND1 STRING='EXEC',
	  POS=ANY
	FIND2 REJECT,
	  STRING='PGM',
	  POS=ANY
	START TYPE=DFLT
	END1 TYPE=SPAC
		ELATE ELEMENT NDVR
		  LOCATION
		  ENVIRONMENT = ' '
		  SYSTEM = ' '
		  SUBSYSTEM = ' '
		  TYPE = ' '
		  STAGE = ' '
		  INPUT.

*	SCANTYPE ELEMENT FIND1 STRING='EXEC', POS=ANY START TYPE=DFLT END1 TYPE=SPAC	

Example 4

In this assembler example, CONSCAN searches for source records containing links to other programs. Two FIND statements are required to identify these programs. Both conditions must be true to be selected. The search is terminated when a blank space or a comma is detected.

Source Statements	CONSCAN Statements	CONRELE Statements
LINK EP=CSECT1	SCANTYPE ELEMENT FIND1 STRING='LINK', POS=ANY FIND2 STRING='EP=', POS=ANY START TYPE=DFLT END1 TYPE=SPAC END2 TYPE=CHAR, PARM=','	RELATE ELEMENT CSECT1 LOCATION ENVIRONMENT = ' ' SYSTEM = ' ' SUBSYSTEM = ' ' TYPE = ' ' STAGE = ' ' INPUT.

Example 5

In this COBOL example, CONSCAN searches for source records containing calls to program XYZ. This program requires three parameters. The goal is to capture the data associated with each parameter and generate a RELATE control statement for each. The extract is terminated after the third parameter or when the closing ')' is detected.

Source Statements	CONSCAN Statements	CONRELE Statements
CALL XYZ,(PARM1 PARM2)	SCANTYPE COMMENT FIND1 STRING='XYZ,(', POS=ANY START TYPE=DFLT END1 TYPE=STRG, PARM=1 END2 TYPE=CHAR, PARM=')'	RELATE COMMENT 'PARM1 PARM2'.

Example 6

In this COBOL example, CONSCAN searches for source records containing object names which are followed by a special comment denoting the type of object. The object name begins in the 20 positions before the FIND string (object), is 16 characters in length and is terminated by a space.

Source Statements	CONSCAN Statements	CONRELE Statements
SERVICE-FIND-APP OBJECT	SCANTYPE OBJECT FIND1 AFTER, POS=7, STRING=' OBJ' FIND2 AFTER, POS=7, STRING='OBJECT' START TYPE=BACK, PARM=20 END1 TYPE=LENG,PARM=16	RELATE 'OBJECT'.

Additional examples can be found in the iprfx.iqual.CSIQOPTN installation library. Assembler examples are provided in member SCANASM, COBOL examples are in SCANCBL, and JCL examples in SCANJCL.

Scan Rule Processing

The element source is read one line at a time. Each selection group defined, from top to bottom, is applied against this source line. If a match is found against a group, the FIND criteria of this group is applied to the remainder of this record in order to check for any additional matches. If a record contains data that matches the FIND criteria of a group, this record is not processed against any of the subsequent selection groups. By default, CONRELE control statements are only generated by one selection group.

If you want your input records to be applied against all the selection groups, regardless of the outcome of previous selection groups, code **APPLY ALL** as the first line of your PARMSCAN control statements. This causes the input record to be applied against each selection group within the PARMSCAN library. Therefore, CONRELE control statements may be generated for one input record by more than one selection group.

Sample CONSCAN Utility Processor

```
//*-------------------------------------------------------------------*
//*
//*
//* NAME: CONSCAN
//*
//* FUNCTION: SCAN THE ELEMENT BASED ON THE PARMSCAN PARAMETERS TO
//* CREATE ADDITIONAL RELATIONSHIPS. THE SCAN RULES VARY
//* DEPENDING ON THE ENDEVOR TYPE.
//*
//* ADD THE RELATIONSHIP TO THE ELEMENT COMPONENT LIST
//* USING THE CONRELE UTILITY.
//*
//* THE SRCIN DD IS ONE OF THE FOLLOWING:
//* 1) FOR FORWARD DELTAS, THIS IS CONWRITE OUTPUT OF THE ELEMENT
//* 2) FOR REVERSE DELTAS, THIS IS BASE OUTPUT LIBRARY
//*-------------------------------------------------------------------*
//*
//*********************************************************************
//** SCAN CONTENTS OF ELEMENT USING THE CONSCAN UTILITY
//*********************************************************************
//CONSCAN EXEC PGM=CONSCAN,REGION=4*96K
//*********************************************************************
//* INPUT FILES
//*********************************************************************
//SRCIN DD DISP=(OLD,DELETE),DSN=&&ELMOUT
//PARMSCAN DD DISP=SHR,DSN=iprfx.iqual.CSIQOPTN(&C1ELTYPE)
//*********************************************************************
//* OUTPUT FILES
//*********************************************************************
//ACMRELE   DD   DISP=(NEW,PASS),DSN=&&RELEIN,SPACE=(TRK,(10,5))
//SCANPRT   DD   DISP=(OLD,PASS),DSN=&&SCOUT2
//SYSPRINT  DD   DISP=(OLD,PASS),DSN=&&SCOUT1
//*
//*********************************************************************
//* ADD RELATIONSHIPS TO ELEMENT COMPONENT LIST
//*********************************************************************
//CONRELE    EXEC   PGM=CONRELE
//NDVRIPT    DD     DISP=(OLD,DELETE),DSN=&&RELEIN
//*
```

Note: The SYSPRINT DD is required when running the CONSCAN JCL in a CA Endevor SCM processor or in a batch job.

The CONWRITE Utility

You can use the CONWRITE utility to write component list data or the current level of any element to an external location such as a data set or a database.

- The standard form of CONWRITE writes the current element to a user specified data set.
- The extended form of CONWRITE:
  - Writes component list data to an external file.
  - Processes WRITE ELEMENT control statements to write user specified elements to an external data set or USS directory.
  - Passes individual element data to a user exit program.

Both forms of CONWRITE can expand INCLUDE statements embedded in the element.

Writing Component List Data to an External Location

The CONWRITE utility provides you with the ability to take component list data and store it in an external data set or use the component list data as input to other processes. The component list can either be an active component list or an existing component list. (An active component list is actively being built in memory by a processor, and may be incomplete depending on when and where you request it. If this is a delete processor, the list is taken from disk.)

You can code a CONWRITE step anywhere in any processor. For example, you can code a CONWRITE step within a move processor. The default location of the component list is the target location of the MOVE action, but you can override this default.

Note: You should be aware when retrieving an active component list that the entire component list is not available until all steps are complete.

Component List Data

CONWRITE allows you to extract and use the following component list data:

- Input components
- Output components
- Symbolics
- Related input and related output data

- Related objects
- Related comments

Note: Use the CONRELE utility to create related input components, related output components, related objects, and related comments before using the CONWRITE utility.

Output Format

The output format of the extracted component list data does not contain binary data. Assembler DSECTS and COBOL copy statements are provided in the output source file of the install process. In addition, COBOL layouts for the data are provided in &uprfx.SOURCE(COBCOMP), and an example of a COBOL program displaying records field by field is provided in &uprfx.SOURCE(COBCOMPX). See the following elements for the layout of external component list records:

- $COMPDS ASMMAC - Assembler layouts for component list data
- COMPRECS EXAMPLE - COBOL copy statements for component list data
- CONCOMP EXAMPLE - COBOL program which displays fields within component list data.

Writing Elements to an External Location

When CONWRITE writes to an external data set or USS directory, it validates the external data set record length against the maximum record length defined in the element type record. If the external data set record length is less than the element type record length, CONWRITE truncates the element data records. If the external data set record length is greater than the element type record length, CONWRITE pads the element records with spaces. In both cases CONWRITE issues a warning message and sets the step return code to four. If either of these circumstances is not desired then code the MAXRC or COND statement on the processor JCL to terminate the processor.

The Standard Form of CONWRITE

The standard form of CONWRITE writes to the first DD statement after the EXEC statement that does not begin with C1INCL and is not the CONWLIB or CONWIN statement. The CONWRITE output can be passed to a subsequent processor step such as a compiler or assembler. STEPLIB is a reserved DD name for the CONWRITE utility.

Note: The standard form of CONWRITE does not support the extraction of a component list.

The following JCL is an example of the standard form of the CONWRITE utility:

```
//WRITE   EXEC PGM=CONWRITE,PARM='EXPINCL(N)'
//ELMOUT    DD DSN=&&SYSIN,DISP=(NEW,PASS,DELETE)
```

The Extended Form of CONWRITE

You can use the extended form of CONWRITE to extract elements or component list records for any CA Endevor SCM element. CONWRITE reads WRITE ELEMENT control statements from the CONWIN DD statement to determine which elements to be extracted. You can write the output element or component list records to either an external data set or you can pass it to a user specified exit program.

The following JCL is an example of the extended form of CONWRITE:

```
//WRITE  EXEC PGM=CONWRITE,MAXRC=4
//CONWLIB  DD DSN=user.loadlib,DISP=SHR
//CONWIN   DD *
```

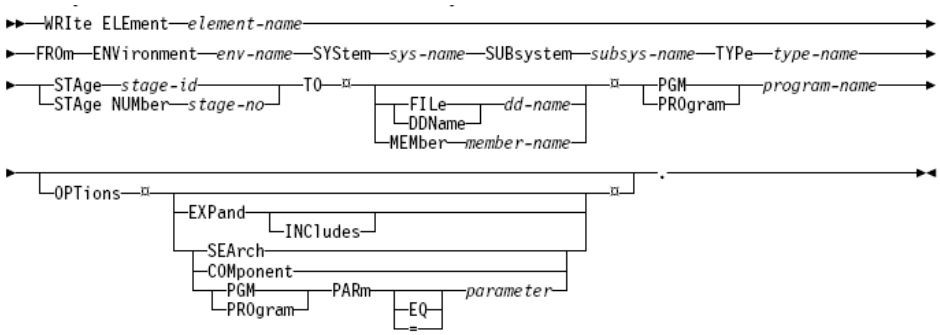
You create copies of more than one element or component list record using the CONWIN DD statement. You can also use this statement to pass the individual element or component list records to a user exit program.

Sample JCL for using the CONWIN DD statement to extract a component list with the extended form of CONWRITE is shown next:

```
//WRITE   EXEC   PGM=CONWRITE
//COMPOUT   DD   DSN=&user.data.set,DISP=(NEW,PASS),UNIT=SYSDA,
//               SPACE=(TRK,(3,5),RLSE),
//               DCB=(RECFM=VB,LRECL=256,BLKSIZE=0,DSORG=PS)
//CONWIN    DD    *
  WRITE ELEMENT &c1element
     FROM ENV &c1envmnt SYSTEM &c1system SUBSYSTEM &c1subsys
          TYPE &c1type STAGE &c1stgid
  TO DDN COMPOUT
  OPTION COMPONENT.
 /*
```

Command Syntax for the CONWRITE Utility

The syntax of the CONWRITE utility is shown as follows:



user.loadlib

The load library where a user exit program resides.

program-name

The name of a user exit program invoked for each element record. If the CONWLIB DD statement is specified the program is loaded from the library specified in the DD statement.

On entry to the user program, R1 points to this parameter list:

- **+0-**Address of a 256 byte work area. The area is initialized to zeroes for the first invocation of the exit.
- **+4-**Address of a half word containing the record length.
- **+8-**Address of the data record.
- **+12-**Address of a 100 byte message area. The message area is set to blanks before each user program call.
- **+16-**Address of the parameter specified on the 'program parm =' statement. The parameter is a halfword length field followed by the parameter string. If a 'program parm =' statement was not specified, the halfword length is set to zero.

parameter

A 1- to 70-character parameter passed to the user program.

Note: When you include a CONWIN DD statement in a CONWRITE step, CONWRITE ignores PARM information in the EXEC statement for that step.

There is no limit to the number of WRITE ELEMENT statements that can be included in the CONWIN DD statement.

If a syntax error is detected in the CONWIN DD input stream, CONWRITE issues an error message. All remaining WRITE ELEMENT statements are syntax-checked, but they are not executed.

For example, if link-edit control cards are separate from programs, and you want to use these control cards in a processor, you can fetch the control cards using the extended form of CONWRITE. The following example shows how the control cards can be processed:

Note: This example uses OPTION SEARCH to fetch the current version of LINKCARD's source.

```
//*********************************************************************
//**  EXAMPLE OF CONWRITE EXECUTION JCL WITH CONWIN DD STMT INPUT **
//*********************************************************************
//CONWRITE EXEC PGM=CONWRITE
//*
//* ELMSRC IS A TEMPORARY DATA SET USED TO CONTAIN ELM SOURCE CODE
//*
//ELMSRC  DD DSN=&&ELMSRC,DISP=(,PASS),
//           UNIT=VIO,SPACE=(TRK,5,1),
//           DCB=(RECFM=FB,LRECL=80,BLKSIZE=0,DSORG=PS)
//*
//* LNKSRC IS A TEMPORARY DATA SET USED TO CONTAIN ELM LINK-EDIT STMTS
//*
//LNKSRC  DD DSN=&&LNKSRC,DISP=(,PASS),
//           UNIT=VIO,SPACE=(TRK,1,1),
//           DCB=(RECFM=FB,LRECL=80,BLKSIZE=0,DSORG=PS)
//CONWIN  DD *
 WRITE ELEMENT &C1ELEMENT.
     FROM ENV &C1ENVMNT. SYSTEM &C1SYSTEM. SUBSYSTEM &C1SUBSYS.
          TYPE &C1ELTYPE.  STAGE &C1STGID.
     TO  DDN ELMSRC
     OPTION EXPAND INCLUDES .
 WRITE ELEMENT &C1ELEMENT.
     FROM ENV &C1ENVMNT. SYSTEM &C1SYSTEM. SUBSYSTEM &C1SUBSYS.
          TYPE LINKCARD  STAGE &C1STGID.
     TO  DDN LNKSRC
   OPTION SEARCH .
//*
//*********************************************************************
//**  PERFORM COMPILE                      **
//*********************************************************************
//COMPILE EXEC...
//SYSIN  DD DSN=&&ELMSRC,DISP=(OLD,DELETE)
//*
//*********************************************************************
//**  PERFORM LINK-EDIT                      **
//*********************************************************************
//LINKEDIT EXEC PGM=IEWL
//SYSLIN  DD DSN=&&LNKSRC,DISP=(OLD,DELETE)
```

Using CONWRITE to Expand INCLUDEs

If the element source references one or more INCLUDE members, you can either expand or not expand those members within CONWRITE using either of the following clauses.

The PARM=EXPINCL( ) Clause

Specify PARM=EXPINCL(Y) to expand INCLUDE members; specify PARM=EXPINCL(N) if you do not want to expand the INCLUDE members. The default is PARM=EXPINCL(N).

```
//STEPNAME EXEC PGM=CONWRITE,PARM=EXPINCL(Y)
//DDNAME     DD DSN=&&WRITEOUT,DISP=PASS,UNIT=SYSDA,
//              SPACE=(TRK,(3,5),RLSE),
//              DCB=(RECFM=FB,LRECL=80,BLKSIZE=0,DSORG=PS)
```

The EXPAND INCLUDES Option within a WRITE ELEMENT Statement

```
//CONWIN  DD *
 WRITE ELEMENT &C1ELEMENT.
       FROM ENV &C1ENVMNT. SYSTEM &C1SYSTEM. SUBSYSTEM &C1SUBSYS.
            TYPE &C1ELTYPE.  STAGE &C1STGID.
       TO  DDN ELMSRC
   OPTION EXPAND INCLUDES .
```

Note: CONWIN DD statements and PARM clauses are mutually exclusive. If you code a CONWIN DD statement, CONWRITE ignores any PARM clauses in the EXEC statement for the step.

By default, CONWRITE searches the environment map for INCLUDE members in the default INCLUDE libraries for specified types. This means that when expanding INCLUDEs for a Stage 1 element, CONWRITE looks first in the Stage 1 INCLUDE library for the member, then in INCLUDE libraries in successive stages on the map. When processing a Stage 2 element, CONWRITE looks for the INCLUDE member first in the Stage 2 library, then in successive stages on the map.

Alternatively, you can use C1INCL DD statements to specify up to 99 INCLUDE libraries within the CONWRITE step of a processor. When you do this, CA Endevor SCM accesses these statements in ascending order based on their DD names. The example below shows how to write C1INCL DD statements.

```
//STEPNAME EXEC PGM=CONWRITE,PARM=EXPINCL(Y)
//C1INCL01 DD DSN=include.library1,DISP=SHR
//C1INCL02 DD DSN=include.library2,DISP=SHR
//DDNAME   DD DSN=&&WRITEOUT,DISP=(NEW,PASS),
//            UNIT=SYSDA,SPACE=(TRK,(3,5),RLSE),
```

Note: These search rules are mutually exclusive. If you use C1INCL DD statements, CONWRITE does not search in the default INCLUDE libraries for the specified types. Also, all C1INCL DD statements must precede the CONWRITE output data set statement.

Writing Exit Programs to Use CONWRITE Input

The WRITE ELEMENT control statement allows CONWRITE to pass the output element, record by record, to a user-specified exit program. The program could, for example, process the data before the exit writes the record to an external data set. Specify the user program on the TO PROGRAM statement of the WRITE ELEMENT action. You can use the OPTION PROGRAM PARM EQ statement to pass a parameter string to the user program.

CONWRITE passes the user exit a five-word parameter list for each data record. Register 1 points to the parameter list. The parameter list and all parameters are in 24-bit addressable storage. The parameter list points to these fields:

- A 256-byte program work area. The work area is double word aligned and is initialized to binary zeroes for the first exit call. The work area is not reinitialized between exit calls or between WRITE ELEMENT statements within a step.
- A halfword field that contains the length of the record.
- The element data record.
- A 100-byte area in which the user program can place a message. The area is initialized to blanks prior to each invocation of the exit. The message will be printed if the exit sets a return code of four or eight and the message area contains non-blank data.
- The program parameter specified in the OPTION PROGRAM PARM EQ statement. The parameter is a halfword length field followed by the parameter specified. If the PROGRAM PARM EQ statement was not specified, CONWRITE sets the halfword length field to zero.

After CONWRITE processes all of the element data records, it calls the exit a final time with both the record length field and the record address parameter set to zero. CONWRITE expects one of the following return codes from the user program. The return code is passed back in Register 15.

0

Normal completion. CONWRITE continues processing the element.

4

Terminate the current WRITE ELEMENT operation. CONWRITE continues with the next WRITE ELEMENT statement.

8

Terminate CONWRITE processing. CONWRITE immediately terminates with a return code of 12. If an invalid return code is received, CONWRITE immediately terminates the current step with a return code of 12.

The exit program should be link-edited with the REUS attribute.

The ENUSSUTL Utility

The ENUSSUTL processor utility collects package backout information for USS processor output files. This backout information consists of backout records and backout files.

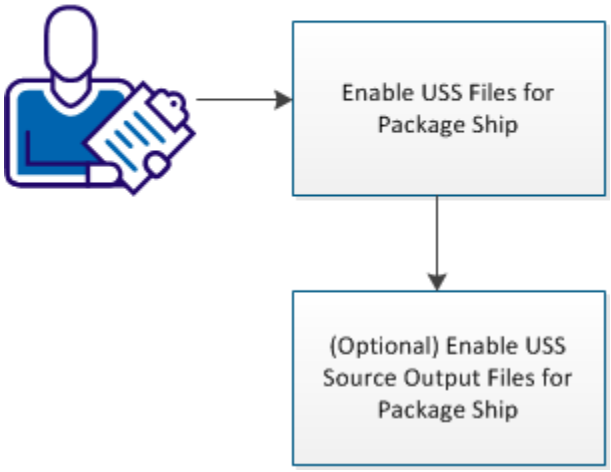
Package outputs for USS files can be transmitted to other sites using the Package Ship utility. However, the output cannot be shipped unless package backout information for USS files is available. To collect this information, the processor that is associated to the element type must include the ENUSSUTL utility to collect and associate the backout files to an element. When this utility is executed under Package processing, the utility enables these files for Package Backout and Package Ship.

How to Enable USS Supported Files for Package Ship

As a change manager (CA Endevor SCM administrator), you can enable USS supported files (HFS, zOS, or NFS files) for the Package Ship facility. Before USS supported files can be shipped, backout files and records must exist for these files. To create backout files and records, you add the ENUSSUTL utility to the processor associated to the element Type definition.

The following diagram shows how you enable USS files for Package Ship and, optionally, how you can enable USS source outputs for package ship.

How to Enable USS Files for Package Ship



Complete the following procedures to enable USS files for Package Ship:

1. Enable USS Files for Package Ship (see page 127).
2. (Optional) How to Enable USS Source Output Files for Package Ship (see page 131)—This process is required only If you have USS source output files that you intend to back out or ship.

More Information

The ENUSSUTL Utility (see page 133)

Enable USS Supported Files for Package Ship

Backout files and records are required to enable package outputs to be shipped using the Package Ship facility. For USS supported files, the ENUSSUTL utility creates the backout files and records. The processor that is associated to the element type must include the ENUSSUTL utility to collect and associate the backout files to the element.

Follow these steps:

1. Add the ENUSSUTL utility to the processor with the appropriate Copy and Select or Delete and Select statements.

   The ENUSSUTL syntax depends on whether it is used in a Move, Generate, or Delete processor. Move and Generate processors use Copy statements. Delete processors use Delete statements.

   - **Move and Generate processors**

     COPY syntax—Copy, Select group statements are used in Generate and Move processors. In a generate processor, copy the USS file from the CA Endevor SCM path location to a temporary path. Then use this file as input into the ENUSSUTL utility.

     ```
     Copy Indd dd-name  Outdd dd-name .
     Select File file-name [Newfile file-name].
     ```

     Copy Indd dd-name Outdd dd-name

     Identifies the dd-name copy source and target path locations.

     Indd dd-name

     Specifies the dd-name source location. CA Endevor SCM and user symbolics can be used on the JCL path name specification.

     Outdd dd-name

     Specifies the dd-name target location. CA Endevor SCM and user symbolics can be used on the JCL path name specification.

Select FIle file-name

Specifies the name of the file at the source location for the associated Copy statement. If the Newfile clause is not used, this file specification will be used at the target location. A file name can be up to 255 characters. See the SCL Reference Guide for information on how to specify a value that spans multiple 80 character lines. Multiple select statements can follow the copy statement. Copy and Select group statements are supported. Select statements are paired with the copy statement that precedes it. CA Endevor SCM and user symbolics can be used on the file name specification.

Newfile file-name

(Optional.) Specifies the copy file name to be used for the file at the target location. If this clause is not used, the file specification name is used. The name can be up to 255 characters. See the *SCL Reference Guide* for information on how to specify a value that spans multiple 80 character lines. CA Endevor SCM and user symbolics can be used on the file name specification.

Sample syntax for Move processors:

```
//COPY1   EXEC  PGM=ENUSSUTL
//INPUT   DD  PATH='input-pathame'
//OUTPUT  DD  PATH='output-pathname',
//        PATHMODE=(SIRWXU,SIRWXG,SIRWXO)
//ENUSSIN   DD  *
  COPY INDD 'INPUT'  OUTDD 'OUTPUT' .
  SELECT FILE 'fileone.ext' .
  SELECT FILE 'filetwo.ext' .
```

Sample syntax for Generate processors:

```
//BPXW1   EXEC  PGM=BPXBATCH
//STDPARM DD  *
//SH cp –B 'pathname.&C1ELELMNT255.' 'temp-pathname.&C1ELELMNT255.';
//STDOUT  DD  SYSOUT=*
//STDERR  DD  SYSOUT=*
//COPY1   EXEC  PGM=ENUSSUTL
//INPUT   DD  PATH='temp-pathame'
//OUTPUT  DD  PATH='pathname',
//        PATHMODE=(SIRWXU,SIRWXG,SIRWXO)
//ENUSSIN   DD  *
  COPY INDD 'INPUT'  OUTDD 'OUTPUT' .
  SELECT FILE &C1ELELMNT255. .
```

Rules for Copy syntax as are follows:

- Multiple SELECT statements can follow the COPY statement.
- Multiple COPY SELECT groups can be specified. Select statements are associated with the preceding Copy statement.

```
COPY INDD 'INPT1'  OUTDD 'OUTP1'  .
SELECT FILE 'fileone.ext'  .
SELECT FILE 'filetwo.ext'  .
COPY  INDD 'INP2'  OUTDD 'OUTP2'  .
SELECT FILE 'filethree.ext' .
```

- The JCL PATHOPTS parameter is ignored if used on the Copy DD JCL statements.
- The PATHMODE parameter should be used on the OUTDD DD statement. If unspecified, the default filemode is set to '000'.
- Concatenation of paths are not allowed on the INDD DD statement. The utility will check for this and if found will fail during syntax validation.
- You cannot copy to the same path and file name.
- By default, ENUSSUTL copies the USS file using the credentials of the Alternate ID, so the owner and group of the file after the copy is the Alternate ID. If you want to use the user's credentials instead code, ALTID=N on the processor step.

- **Delete processors**

  DELETE Syntax—Use the utility's Delete and Select group statements in a delete processor.

```
Delete FRomDD dd-name .
Select File file-name .
```

  Delete FRomdd dd-name

  Identifies the DELETE target path locations.

  FRomdd dd-name

  Specifies the FROM dd-name. CA Endevor SCM and user symbolics can be used on the JCL path name specification.

  Select FIle file-name .

  Specifies the name of the file to be deleted that is associated with the Delete statement. A file name can be up to 255 characters. See the *SCL Reference Guide* for information on how to specify a value that spans multiple 80 character lines. Multiple select statements can follow the delete statement. Delete and Select group statements are supported. Select statements are paired with the delete statement that precedes it. CA Endevor SCM and user symbolics can be used on the file name specification.

Sample syntax for Delete processors:

```
//DEL1    EXEC  PGM=ENUSSUTL
//FROM    DD  PATH='pathname',
//        PATHMODE=(SIRWXU,SIRWXG,SIRWXO)
//ENUSSIN   DD  *
  DELETE  FROMDD 'FROM'
  SELECT FILE 'fileone.ext' .
  SELECT FILE 'filetwo.ext' .
```

Rules for Delete syntax as are follows:

- Multiple SELECT statements can follow the DELETE statement.
- Multiple DELETE SELECT groups can be specified. Select statements are associated with the preceding Delete statement.

```
DELETE FROMDD 'FROM1'       .
SELECT FILE 'fileone.ext'   .
SELECT FILE 'filetwo.ext'   .
DELETE FROMDD 'FROM2'       .
SELECT FILE 'filethree.ext' .
```

- The JCL PATHOPTS parameter is ignored if used on the Delete DD JCL statements.
- Concatenation of paths are not allowed on the FROMDD DD statement. The utility checks for this and if found will fail during syntax validation.
- By default, ENUSSUTL deletes the USS file using the credentials of the Alternate ID, so the Alternate ID must have appropriate access to the from path. If you want to use the user's credentials instead code, ALTID=N on the processor step.

2. (Optional) Include the NOECHO parameter on the EXEC statement, if you want to suppress the echo of ENUSSUTL commands produced by the parser from appearing in the C1MSGS1 output.

   NOECHO

   Suppresses the SCL statements from appearing in the C1MSGS1 output.

   For example, the following EXEC statement includes the NOECHO parameter and will prevent the ENUSSUTL commands from appearing in the C1MSGS1 output:

   ```
   //STEPxx  EXEC PGM=ENUSSUTL,PARM='NOECHO'
   ```

How to Enable Backout of USS Source Output Files

Source Output USS files created by CA Endevor SCM through the Type definition cannot be backed out or shipped. To have this alternate file created and to have it available for Package Backout and Ship, you must change the Type definition and Processor. One way to change the processor is to use the CONWRITE and ENUSSUTL utilities in the processor. To use this method, complete the following steps:

1. Add the CONWRITE utility to the processor to extract the element to a temporary file. For more information about CONWRITE, see the *Extended Processors Guide*.
2. Add the ENUSSUTL utility to the processor after the CONWRITE utility. Use the Copy and Select statements to specify that the ENUSSUTL utility copy the temporary file to the targeted USS file. The targeted USS file is the source output library defined on the Type definition.

   By using the ENUSSUTL utility, a backout record and a backout file is created when an action is executed under package processing control against this element. For more information about using the ENUSSUTL utility, see Enable Supported Files for Package Ship.
3. Change the Type definition to remove the SOURCE OUTPUT USS library definition.

Note: If you do not intend to back out or ship your source output USS file, you do not need to change your processor or type definition.

Example: CONWRITE and ENUSSUTL in Processor for USS Source Output Files

This example shows how a processor can use the CONWRITE and ENUSSUTL utilities to create back out files and information for USS source output files. Thus enabling the USS files to be shipped. This partial processor uses the CONWRITE utility to create USS output from a CA Endevor SCM element and copy it to a temporary USS file. Then the ENUSSUTL utility copies the USS file to its real name and location. Then the BPXBATCH utility deletes the temporary USS file.

```
//GUSS  PROC USSDIR='/u/users/endeavor/&C1EN(1,1)&C1S#/'
                        •
                        •
                        •
//*************************************************************
//* Create USS output from endevor element to a temporary USS
//* file and then use ENUSSUTL t0 copy it to its real name
//* and location.
//* Delete the temporary USS file
//* - CONWRITE element.tmp
//* - ENUSSUTL copy element.tmp to element
//* - BPXBATCH delete element.tmp
//*************************************************************
//CONW1   EXEC  PGM=CONWRITE,MAXRC=0
//ELMOUT1 DD  PATH='&USSDIR',
//        PATHOPTS=(OWRONLY,OCREAT),
//        PATHMODE=(SIRWXU,SIRWXG,SIRWXO)
//CONWIN  DD *
 WRITE ELEMENT &C1ELMNT255
    FROM ENV &C1EN SYSTEM &C1SY  SUBSYSTEM &C1SU
         TYPE &C1TY STAGE &C1SI
    TO  DDN  ELMOUT1
        HFSFILE &C1ELMNT255..TMP
 .
//*************************************************************
//ENUSS1  EXEC  PGM=ENUSSUTL,MAXRC=4
//INPUT   DD  PATH='&USSDIR'
//OUTPUT  DD  PATH='&USSDIR',
//        PATHMODE=(SIRWXU,SIRWXG,SIRWXO)
//ENUSSIN DD  *
 COPY INDD 'INPUT' OUTDD 'OUTPUT' .
 S FILE '&C1ELMNT255..tmp
   NEWF '&C1ELMNT255'
 .
//*************************************************************
//BPXB1  EXEC  PGM=BPXBATCH,MAXRC=0,COND=(4,LT)
//STDPARM DD  *
SH rm –r '&USSDIR.&C1ELMNT255..tmp' ;
//STDOUT  DD  SYSOUT=*
//STDERR  DD  SYSOUT=*
//*
```

The ENUSSUTL Utility

The ENUSSUTL processor utility collects package backout information for USS processor output files. This backout information consists of backout records and backout files.

Package outputs for USS files can be transmitted to other sites using the Package Ship utility. However, the output cannot be shipped unless package backout information for USS files is available. To collect this information, the processor that is associated to the element type must include the ENUSSUTL utility to collect and associate the backout files to an element. When this utility is executed under Package processing, the utility enables these files for Package Backout and Package Ship.

Generate and Move processors—The execution of the ENUSSUTL utility program in a Generate and Move processor copies user-selected USS files to the user-specified USS directory. Backout files and backout records are created for the USS files, if executed under a package.

- Backout files are stored at the target location.
- Backout records are stored in the CA Endevor SCM Package data set.

Delete processors—The execution of the ENUSSUTL utility in a Delete processor deletes user-selected USS files from a user-specified USS directory. Backout files and backout records are created for these USS files, if executed under a package.

- Backout files are stored in the same user-specified USS directory.
- Backout records are stored in the CA Endevor SCM Package data set.

How ENUSSUTL Works in a Move Processor

The ENUSSUTL utility with the Copy and Select commands in a Move processor, creates package backout records and files as follows:

1. If the select source file exists at the target location, an eight-character backout file name is generated and used as the backout saved file name. The selected file is renamed at the target location to the backout saved file name.
2. The selected Indd path and file is copied to the Outdd path and file.
3. A USS Backout record is written to the package dataset.

How ENUSSUTL Works in a Generate Processor

The ENUSSUTL utility with the Copy and Select commands in a Generate processor creates backout records and files as described by the Move processor. Before this utility is invoked in a Generate processor a step is required that copies the USS files to another temporary directory. These files can then be selected for input into the ENUSSUTL utility.

How ENUSSUTL Works in a Delete Processor

The ENUSSUTL utility with the Delete and Select commands in a Delete processor creates backout records and files as follows:

- If the select source file does not exist at the target location, no backout or rename processing is done.
- If the select source file exists at the target location, an eight-character backout file name is generated and used as the backout saved file name. The existing file is renamed to the backout saved file name in the same location.
- The selected file is deleted.
- A USS Backout record is written to the package dataset.

The LEXTRCTR and BC1PCCSP Utilities

The CSP 4.1 generate program produces link cards that must be passed dynamically to the linkage editor during processor execution. CA Endevor SCM for CSP uses the LEXTRCTR utility to pass these link cards.

The purpose of LEXTRCTR is to parse and capture all the pieces of a CSP application when the user is managing at the application versus the component level. BC1PCCSP then attaches the compiler for each piece of the application identified by LEXTRCTR.

Note: For an example of the use of LEXTRCTR and BC1PCCSP, see the appendix "Sample Processors (see page 155)".

Chapter 4: Classifying and Managing Processors

This section contains the following topics:

Classifying Processors

Processors are identified by the same logical structure used for other elements: they are located in a particular environment and stage, and are classified by system, subsystem, and type. This section discusses the classification scheme.

You must classify processors as type PROCESS.

For elements of type PROCESS, CA Endevor SCM reserves a processor group, also called PROCESS. Type PROCESS has a predefined set of processors-GPPROCSS and DPPROCSS--which are defined automatically in this processor group:

- **GPPROCSS** is the generate and move processor in Stage 1 and Stage 2. It checks the processor syntax and creates an executable form of the processor in the processor load library. If a processor contains syntax errors, GPPROCSS does not create the executable form. In this situation, refer to the listing created by GPPROCSS in the processor listing library, to check your errors.
- **DPPROCSS** is the delete processor in Stage 1 and Stage 2. It deletes the executable form of the processor, should you delete the processor source from CA Endevor SCM.

You can change only two items in processor group PROCESS:

- The description of the processor group.
- Whether the processor is executable in foreground as well as batch.

After defining the type 'process', you can perform any CA Endevor SCM action against the type 'process'.

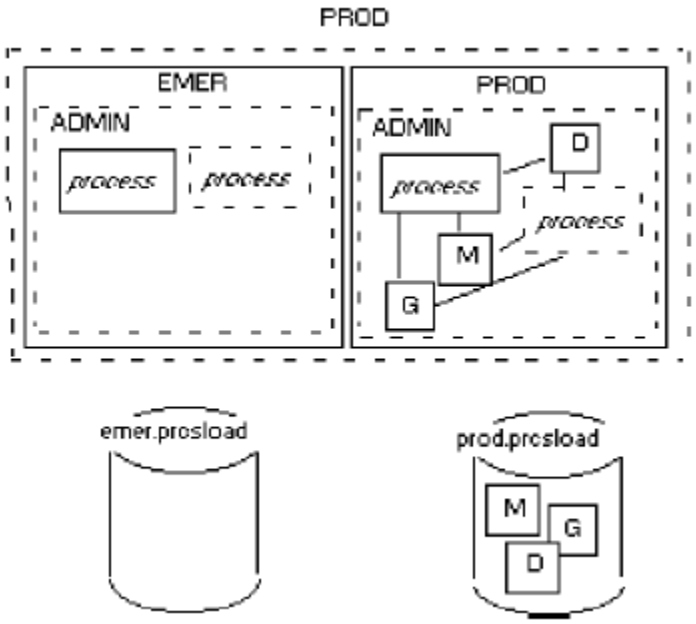
Note: For more information about action processing, see the *User Guide*.

This section describes how to implement and maintain processors.

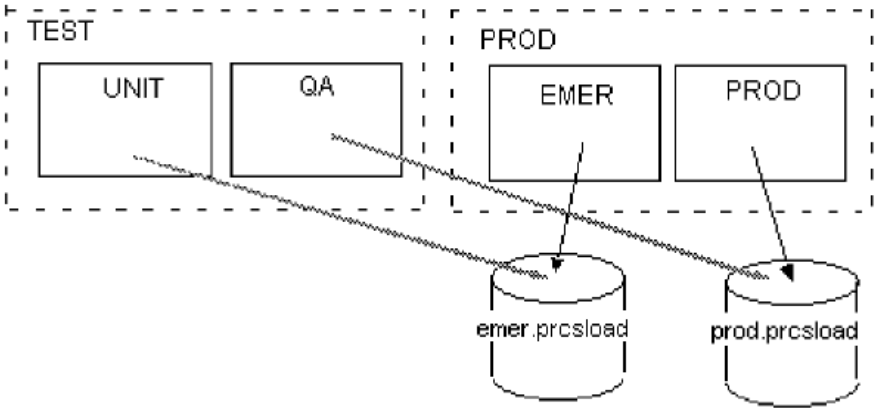
Implement Processors

To implement processors

1. Define a system, for example ADMIN, in your production environment, specifying ENDEVOR.EMER.PRCSLOAD as the Stage 1 processor load library and ENDEVOR.PROD.PRCSLOAD as the Stage 2 processor load library for system. Remember to define the required processor type, PROCESS, to this system. Allocating listing libraries is optional.
2. Define a subsystem, for example PROCESS, within system ADMIN. Use subsystem PROCESS to store and maintain all your processors. The following diagram shows the resulting configuration:



3. Have all systems in all environments reference ENDEVOR.EMER.PRCSLOAD as their Stage 1 processor load library, and ENDEVOR.PROD.PRCSLOAD as their Stage 2 processor load library. However, keep in mind that you maintain processors only in system ADMIN in environment PROD.



Note: Process types cannot be mapped.

Maintain Processors

To maintain processors managed according to the previous example:

1. Copy your current administration system to create a test system.
2. Modify the PRCSLOAD and LIST entries in the test system definitions.
3. Modify the Base/Delta entries for types.
4. Add the processor to the test system and run your test programs.
5. After the tests have completed successfully, transfer the processor to your administration system.

Where CA Endevor SCM Looks for Processors

After determining the processor group for an action, CA Endevor SCM looks for the requisite processor first in Stage 1, then in Stage 2.

Chapter 5: Using Processor Groups

This section contains the following topics:

Processor Group Overview

There are three main processor types, namely delete, generate, and move. Instead of associating one of each type of processor with each element type, CA Endevor SCM allows you to combine the processors into processor groups, and to associate one or more of these groups with each element type. A processor group:

- Identifies the delete, generate, and move processor that CA Endevor SCM should use to process a particular element type. (A processor group can identify less than three processors if a particular element type does not require all of them. For example, a non-executable element type may only require a move processor, so its processor group can omit a generate and delete processor.)
- Contains the symbolic overrides for the processors' JCL.

  Note: For more information about symbols, see Symbols Parameters (see page 30).

You can associate any number of processor groups with a given element type. This is useful when elements of one type may require slightly different processing. For example, a site may have COBOL programs coded in batch COBOL and CICS COBOL. In this case, the processor group capability of CA Endevor SCM allows you to create a single COBOL type with two processor groups, one to handle each variation of COBOL code. Furthermore, symbols allow you to use the same processor for both types, changing only the symbolic overrides in the two processor groups.

If the stage 1 is the entry stage, then the processor group must be defined to both stage 1 and stage 2. If stage 2 is the entry stage (ENTRYSTG#=2 in C1DEFLTS) then it is only necessary to define the processor group in stage 2.

Suggested Naming Conventions for Processor Groups

Processor group names can have up to eight characters. The following abbreviations do not represent a complete list, and are offered as guidelines only. They are shown by position and description:

1-3

Language Type

- ASM = Assembler
- COB = COBOL
- CII = COBOL 2
- EAS = Easytrieve
- FOR = Fortran
- PLI = PL-1
- RPG = RPG
- TRA = Transform
- UTL = Utility

4

Database environment

- D = DB2/DL1
- S = IDMS
- I = IMS
- N = None

5

Operating environment

- B = Batch
- C = CICS
- S = IDMS-DC
- I = IMS-DC
- N = None

6

Output type

- A = Impact analysis SCL
- L = Load module
- K = NCAL load module
- O = Object module
- N = None
- P = PDS
- R = Reports
- S = Listing

7, 8

User-defined. Can be used for sequence number, stage identifier, option, etc.

Example

The example later in this section discusses two processor groups: COBNBL and COBNBL01. The identifier COBNBL indicates that the processor group processes COBOL elements (COB), that the elements do not have a database type (N), that the processor is for batch execution (B), and that the output of processor will be a load module (L). The identifier 01 in processor group COBNBL01 is a sequence number indicating that COBNBL is a default processor group, with processor COBNBL01 as a derivative processor group.

User-Defined Symbols

CA Endevor SCM supports user-defined symbols in processors. The symbols defined for each processor in a processor group appear on the Processor Group Symbolics panel. You can use this panel to view and/or override user-defined symbols for this processor group.

To specify a default processor group for an element type, enter the group's name in the DFLT PROC GRP field on the Type Definition panel. After that, each element of that type that you create is automatically associated with the type's symbolic overrides. If the element requires special processing, however, you can override the default symbols at execution time.

Note: For more information, see Change or Display Processor Group Symbols (see page 144).

Processor Group Information

You create, update, and display processor group information using the Processor Group Definition panel. You can:

- Change the definition of a processor group by selecting option **6** from the Environment Options Menu.
- Change the default processor group for a given type, or assign a new processor group to a new type, by changing the value in the DFLT PROC GRP field on a Type Definition panel.

Create, Update, and Display Processor Group Information (Environment Options Menu)

To create, update, or display processor group information from the Environment Options Menu:

1. Select option **6** and press Enter. CA Endevor SCM displays the Processor Group Request panel.
2. On the Processor Group request panel, type:
   - An option (**Blank, #, C**, or **U**).
   - An environment name, if different from the displayed name.
   - A system name or mask (not used for CREATE).
   - A type name or mask (not used for CREATE).
   - A stage ID.
   - A processor group name or mask (not used for CREATE).

   Press ENTER.
3. If CA Endevor SCM displays a:
   - System Selection List, select a system and press Enter.
   - Type Selection List, select a type and press Enter.

- Processor Group Selection List, select a processor group and press ENTER. You can:
  - Display information about the group by typing an **S** to the left of the group's name.
  - Update information about the group by typing a **U** to the left of the group's name.
  - Delete the group by typing **#** to the left of the group's name.
- Processor Group Definition panel, type or change information as necessary on the Processor Group Definition panel and press Enter to save the changes.

  Note: For more information about this panel, see The Processor Group Definition Panel (see page 147).

Create, Update, and Display Processor Group Information (Type Definition Panel)

To create, update, or display processor group information from the Type Definition panel:

1. Access the Type Definition panel in update or create mode by selecting option **C** or **U** from the Type Request panel. For details, see the chapter, “Defining Inventory Structure” in the *Administration Guide.*
2. Change the DFLT PROC GRP value and press Enter. CA Endevor SCM displays the Processor Group Definition panel, in either create or update mode.
3. Type or change information as necessary on the Processor Group Definition panel and press Enter. CA Endevor SCM displays the message GROUP CREATED in the upper right corner.

   Note: For more information about this panel, see The Processor Group Definition Panel (see page 147).
4. To return to the Type Definition panel, press End.

Change or Display Processor Group Symbols

To change or display the symbols for a processor, access the Processor Group Definition panel, type either **S** (browse) or **U** (Update) in the SELECTION field next to the processor, and press ENTER. The processing option you have selected is displayed in the upper left corner of this panel.

You can change the value in the -/O (Default/Override) and VALUE fields when the processing mode of this panel is Update.

- If the value in the -/O field is **- (dash)**, you can change the default value of the symbol by typing **O** in the --/O field, and typing new information in the VALUE field, and pressing ENTER.
- If the value in the -/O field is **O**, you can:
  - Enter a new override value by typing it in the VALUE field and pressing ENTER.
  - Restore the default value for the symbol by typing **- (dash)** in the -/O field and pressing ENTER.

Once you have established a default value and an override value for a symbol you can toggle back and forth between them by typing **-- (dash)** or **O** in the -/O field and pressing ENTER.

Note: If the dash is specified the symbols original value is restored when you end your CA Endevor SCM session.

Change or Display Processor Group Symbols - Example

Assume that you usually store the listings from compile/link-edits of your COBOL programs, but that periodically you print the listings rather than store them. You decide to set up a type COBOL in Stage 1 and Stage 2 that references a default processor group, COBNBL, to compile, link-edit, and store COBOL listings. You write these processors using symbols. You then use the same processors as the base for a second processor group (COBNBL01) that, by changing the symbols, can be invoked when necessary to print the listings. To set up these two processor groups:

1. Add the processors for the processor groups to CA Endevor SCM.
2. When you create type COBOL, override the default value ***NOPROC*** in the DFLT PROC GRP field of the Type Definition panel with the name you have selected for the processor group that stores listings, in this case **COBNBL**.

   When you press Enter, a Processor Group Definition panel is displayed.

3. Create the default processor group by typing the following information on the Processor Group Definition panel:
   - A description of the processor group in the DESCRIPTION field.
   - The element names for the generate, move, and/or delete processors that will make up the processor group.
   - One of the following values for each processor:

     Y-if you want to allow the processor to run in foreground.

     N-if you do not want the processor to run in foreground.

     Press ENTER to create the default processor group.
4. To view a list of the symbols for any of these processors, type **S** next to a processor and press Enter.
5. Exit to the Environment Options Menu by pressing END.
6. To create the second processor group that you need for your COBOL inventory, access a Processor Group Definition panel through option **6** of the Environment Options Menu. Specify the name of the second processor group (COBNBL01) on the Processor Group Request panel.

   Type the information for the new processor group on the Processor Group Definition panel.

   Note: We have used the same generate, move, and delete processors for both processor groups.

   Type **U** (update symbolics) next to the generate processor (GCIINBL) on the Processor Group Definition panel and press Enter. On the Processor Group Symbolics panel, type **O** in the -/O field next to the symbol LISTLIB, then type new information in the VALUE field for this symbol.
7. Press ENTER. You have created a second processor group to print compile listings by simply changing the symbols referenced by the GCIINBL processor.

Display Processors

You can access the Processor Display panel from the Processor Group Definition panel by typing **L** (List) in the selection field next to the processor in which you are interested, then pressing ENTER.

Note: For more information about this panel, see The Processor Display Panel (see page 152).

The Processor Group Selection List

CA Endevor SCM displays the Processor Group Selection List when you specify a name mask or an incomplete group name on the Processor Group Request panel. It lists the processor groups currently defined for the specified system, stage, and type.

The panel fields are described, except for SELECTION, they are display-only.

Current Env

Name of the current environment.

Stage ID

Name of the current stage.

System

Name of the current system.

Type

Name of the type to which the processor group(s) apply.

Next Env

Name of the environment in the next map location.

Stage ID

Name of the stage at the next map location.

System

Name of the system at the next map location.

Type

Name of the type at the next map location.

Selection (no title)

Used to select a processor group for display, deletion, or update. Place an **S** (display), **#** (delete), or **U** (update) next to the processor group you want to process.

Processor Group

Name of the processor group.

Processor Group Description

Description of the processor group.

The Processor Group Definition Panel

When you update and/or create processor groups for all types other than type PROCESS, CA Endevor SCM displays the Processor Group Definition panel.

The fields that appear on a Processor Group Definition panel depend on the type to which the processor group applies and the processing option that you have selected.

- Processor Group Definition panels for CREATE and UPDATE provide three output management information options (**S**--browse symbolics, **U**--update symbolics, **L**--list processor) and allow you to enter information in the DESCRIPTION, PROCESSOR, and FOREGROUND EXECUTION fields.
- Processor Group Definition panels for DISPLAY and DELETE provide two output management information options (**S**--browse symbolics, **L**--list processor), and do not allow you to enter information in any fields.
- Processor Group Definition panels for type PROCESS and for the default processor group *NOPROC* provide no output management information options and allow you to enter information only in the DESCRIPTION and FOREGROUND EXECUTION fields. The PROCESSOR O/P TYPE field is available for the default proc group: *NOPROC*

Note: In effect, there is only one processor group for type PROCESS. These “processors that process processors” (GPPROCSS and DPPROCSS) are hard coded in CA Endevor SCM. The only allowed update actions against this processor group are changing the description of the processor group and/or allowing the processors to run in foreground.

If you run an action in foreground that normally would result in a processor being executed, but that processor cannot be run in foreground, you receive a message stating that fact. In this situation, you must submit the job in batch.

The remaining information in this section relates to the Processor Group Definition panel.

The use of the Processor Group Definition panel varies by processing option:

Display

Display a processor group definition.

Delete

View a processor group definition and verify that you want to delete it. To cancel the request, press End.

Create

Define a new processor group. To cancel the request, press End.

Update

Modify a processor group definition. To cancel the request, press END.

After you have entered the necessary information on the panel, press ENTER to perform the requested processing.

Identification Fields

The first six fields on this panel identify the processor group. If you have accessed this panel to display, delete, or update this processor group, information appears in all six fields. If you have accessed this panel to create a processor group, you must type information in the DESCRIPTION field.

Current Env

Display-only. Name of the current environment.

Stage ID

Display-only. Name of the stage in which the processor groups on the list are defined.

System

Display-only. Name of the system in which the processor groups on the list are defined.

Type

Display-only. Name of the type to which the processor group applies.

Next Env

Display-only. Name of the environment at the next map location.

Stage ID

Display-only. Name of the stage at the next map location.

System

Display-only. Name of the system at the next map location.

Type

Name of the type at the next map location. You can change the type name when you access this panel in create or update mode.

Processor Group

Name of the processor group.

Processor O/P Type

This 16 character field's initial default value is a concatenation of the element's type and processor group. The processor "output type" works in conjunction with element registration to enable duplicate element names across systems, subsystems, or element types. This feature prevents like-named modules from overlaying each other in output libraries because the addition of the output type makes each like-named element unique. You can change the output type.

This feature is turned on at the system definition level.

Note: For more information about element registration, see the *Administration Guide*.

Description

Description of the processor group.

Next Prcs Group

Name of the processor group at the next map location. If this processor group was defined to CA Endevor SCM:

- Before release 3.6 was installed, *DEFAULT appears in this field.
- After release 3.6 was installed, the processor group name appears in this field.

You can override these default values in update mode.

Updated

- Identifies the date, time, and user ID of the last user to update the processor group definition. When creating a new processor group definition this field is blank.

Output Management Information Fields

There are four groups of OUTPUT MANAGEMENT INFORMATION fields: Move/Transfer processor selection fields, option fields, processor identification fields, and foreground execution fields.

Move/Transfer Processor Selection

The MOVE action executes a move processor, which copies the inventory from the source to the target location. If you want to recompile inventory as part of the MOVE action, specify **G** in the PROCESSOR TO USE FOR MOVE ACTION field, to tell CA Endevor SCM to execute the generate, not the move processor.

The TRANSFER action executes a generate processor at the target location. If you want to transfer component lists as part of the TRANSFER action, specify **M** in the PROCESSOR TO USE FOR TRANSFER ACTION field, to tell CA Endevor SCM to execute the move, not the generate, processor.

Note: You can select either a move or a generate processor only when transferring from one CA Endevor SCM location to another. When transferring from an archive data set to CA Endevor SCM, or when using a move processor with a TRANSFER action, CA Endevor SCM does not allow you to rename elements at the target location.

Option Fields

The options that appear on Processor Group Definition panels may be used as indicated next. These options are not available for type Process or for processor groups named *NOPROC*.

S-Browse Symbolics

Appears on Processor Group Definition panels for display, delete, create, and update processing. Used to access a Processor Group Symbolics panel.

L-List Processor

Appears on Processor Definition panels for display, delete, create, and update processing. Used to access a Processor Display panel.

U-Update Symbolics

Appears on Processor Definition panels for create and update processing only. Used to access a Processor Group Symbolics panel.

Processor Identification Fields

The GENERATE PROCESSOR, DELETE PROCESSOR, and MOVE PROCESSOR fields on this panel identify the generate, delete, and move processors that make up this processor group.

- If you have accessed this panel to display, delete, or update a processor group, information displays in these fields.
- If you have accessed this panel to create a new processor group, you must type the names of the processors that you want to include in this new group in these fields.

  If these fields contain processor names when you access the Processor Group Definition panel in create mode, you can override these names with new values. If you want to use one or more of the processors that appear on this panel, first make sure that they are defined for the system and stage for which you are creating the processor group.

Note: These are required fields. If you do not want to identify one or more processors in this processor group, you must enter the value ***NOPROC*** in those fields. (CA Endevor SCM converts one or more blanks in these fields to ***NOPROC***.)

Foreground Execution Fields

The foreground execution fields on this panel indicate whether the generate, delete, or move processors identified in the respective PROCESSOR fields can be executed in foreground. The acceptable values are:

Y-the processor can be executed in foreground.

N-the processor cannot be executed in foreground.

When a processor runs as part of a package, the package foreground execution flag (PKGTSO= in the C1DEFLTS Table) overrides the foreground execution flag in the processor group. This means that if a package can be executed in foreground, all processors in that package will execute, regardless of the value in the foreground execution field for the processor group.

The Processor Group Symbolics Panel

Symbolics panel allows you to change the default and override values for the processor's symbolic parameters.

Identification Fields

The first eight fields on this panel identify the inventory area to which the processor group is defined (CURRENT ENV, STAGE ID, SYSTEM, and TYPE), the processor group (PROCESSOR GROUP and DESCRIPTION), a processor within the group (PROCESSOR), and where it is stored (LOAD LIBRARY).

Immediately below the identification fields there are three lines of instructions for working with this panel.

Symbolic Identification Fields

The three remaining fields on this panel provide information about the symbols defined for this processor.

Symbolic

Name of the symbol in the PROC statement of the processor.

- (dash)/O

Indicates the status of the value of the symbol.

- **(dash)**-Default value for the symbol. This is the value assigned to the symbol in the PROC statement contained in the processor.

O-Override value for the symbol.

Value

Value to be assigned to the symbol during the next run of the processor.

The Processor Display Panel

The Processor Display panel allows you to view the JCL for the processors in a processor group. You can access Processor Display panels from Processor Group Definition panels. To access a Processor Display panel, type **L** (List) in the SELECTION field next to the processor in which you are interested and press Enter. CA Endevor SCM displays a Processor Display panel.

Note that the ISPF browse facility is used to display processors, so you can use ISPF browse commands on the COMMAND line.

This panel displays:

- PROCESSOR FOOTPRINT fields, which provide footprint information about the processor.
- A processor listing.

All fields are display-only.

Appendix A: Sample Processors

This section contains the following topics:

Sample Processor Overview

The processor examples in this appendix demonstrate the use of CA Endevor SCM utilities, keywords, and user-defined symbols within the processors. These capabilities allow you to more easily convert your own JCL and PROCs into CA Endevor SCM processors. User-defined symbols can then be overridden by the administrator during processor group definition.

How to Convert PROCs to Processors

To convert an existing JCL PROC into a processor, use the following guidelines.

1. Start with the current JCL or PROC.
2. Copy the current PROC and make the following changes:
   a. Add the **BC1PDSIN** utility as the first step.

      The BC1PDSIN utility is used to initialize temporary data sets. These temporary data sets are then written to by subsequent compile, assemble, or link-edit steps, instead of SYSOUT=* (for example, SYSPRINT). The CONLIST step will then bundle these temporary data sets together and either store them in a listings data set or print them.

      There is a sample BC1PDSIN step in the iprfx.iqual.JCL library as member BC1PDSIN. It can be copied into your PROCs and modified to include or exclude data sets as appropriate.

      Note: For more information about this utility, see The BC1PDSIN Utility.

b. Add the **CONWRITE** utility as the second step.

The CONWRITE utility is used to get a copy of the current level of the controlled base/delta source and write it to a temporary data set. The temporary data set can then be passed as input to a subsequent compile, assemble or link-edit step. For users of CA Panvalet and CA Librarian, the CONWRITE utility will expand ++INCLUDE or -INC statements in the source.

Note: For more information about this utility, see The CONWRITE Utility (see page 119).

There is a sample CONWRITE step in the iprfx.iqual.JCL library as member CONWRITE. This sample CONWRITE step uses the &EXPINC. and &MONITOR. symbols. You can define the following values for these symbols in your PROC statements:

- EXPINC=N

  If you do not want to expand ++INCLUDE or -INC statements.

- EXPINC=Y

  If you want to expand ++INCLUDE and -INC statements.

- MONITOR= COMPONENTS

  If you want to monitor selected data sets in order to build component lists.

- MONITOR=NONE

  If you do not have ACM or do not want the data set monitored.

c. Add **MAXRC=nnn** to each step.

The MAXRC keyword is added to each step on the EXEC statement. It tells CA Endevor SCM of the highest acceptable OS return code (RC) for the step. If the utility returns an OS return code higher than the value coded, CA Endevor SCM will set a failed flag on for the element and issue a CA Endevor SCM return code of 12.

d. Add **MONITOR=COMPONENTS** on appropriate DSN= parameters.

The MONITOR keyword is added to each data set name in a DD statement. This keyword is for ACM users. It instructs ACM to monitor the specific data set for input or output components and to build a component list with the information captured. Add this keyword to copylibs, loadlibs and listing data sets. Non-ACM users can code the MONITOR=NONE keyword in anticipation of adding ACM at a later point in time.

e. Add the **CONLIST** utility as the last step.

   The CONLIST utility is used to concatenate the output listings from several temporary data sets written to by the compile, assemble, and link-edit steps. It then places a banner page on the front of the combined listing and either prints or stores the listing. For more information, see The CONLIST Utility (see page 81).

   There is a sample CONLIST step in the iprfx.iqual.JCL library as member CONLIST. It can be copied into your PROCs and modified to include or exclude data sets as appropriate. This sample CONLIST step uses the &LISTLIB symbol. If you add LISTLIB=listing library name to your PROC statement, CA Endevor SCM stores the listing in the listing library name as coded. If you add LISTLIB=NO to your PROC statement CA Endevor SCM prints the listing.

f. Ensure that all user symbols are defined on the PROC statement.

Generate Processors

This section contains sample generate processors. Samples are provided in iprfx.iqual.CSIQSAMP. These processors are:

GASMNBL

An Assembler compile and link-edit processor.

GCIIDBL

A processor that performs a DB2 pre-compile, compiles, link-edits a COBOL II program, and binds the DB2 plan.

GCIINBL

A COBOL II and COBOL/370 compile and link-edit processor.

GLNKNBL

A link-edit-only processor (composite link).

LOADONLY

This processor handles “sourceless” load modules or binary files that you want CA Endevor SCM to manage It is used as a generate process for ADD and UPDATE, and a move process for MOVE and TRANSFER.

Note: For more information about load module support, see the *Utilities Guide*.

These processors are designed to execute at Stage 1. To use a processor at Stage 2, override during processor group definition the &CSYSLIB1, &LSYSLIB1, &LISTLIB and &LOADLIB symbols with the appropriate Stage 2 data set names.

For ACM users, each processor is coded to monitor the appropriate data sets. This option has been coded using the MONITOR=&MONITOR syntax. The MONITOR symbol defaults to **COMPONENTS** as defined on each PROC statement. If you do not have or use ACM, you can override the value for the MONITOR symbol to read **NONE**.

These processors also offer two alternatives for handling listings. If you override the value of the LISTLIB parameter to read **no**, the listings will be printed rather than stored.

The names of the sample processors in this appendix reflect the conventions discussed (on page xx). Each sample is introduced with an expansion of its identifier. For example, GCIINBL is introduced as **G**enerate COBOL **II**, **N**o database, for **B**atch, and create a **L**oad module.

The GASMNBL Generate Processor

Generate **Assembler**, **N**o database, for **B**atch, and create a **L**oad module.

```
//*******************************************************************
//**                                                               **
//**  ASSEMBLE AND LINK-EDIT PROCESSOR                             **
//**                                                               **
//*******************************************************************
//*
//GASMNBL PROC LISTLIB='&PROJECT..&GROUP.&C1ST..LISTLIB',
//        LOADLIB='&PROJECT..&GROUP.&C1ST..LOADLIB',
//        MACLIB='SYSMACLIB',
//        PROJECT='IPRFX.IQUAL',
//        GROUP='SMPL',
//        STG3='EMER',
//        STG4='PROD',
//        EXPINC=N,
//        LSYSLIB1='&PROJECT..&GROUP.&C1ST..LOADLIB',
//        LSYSLIB2='&PROJECT..&GROUP.&C1ST2..LOADLIB',
//        LSYSLIB3='&PROJECT..&GROUP.&STG3..LOADLIB',
//        LSYSLIB4='&PROJECT..&GROUP.&STG4..LOADLIB',
//        MACLIB1='&PROJECT..&GROUP.&C1ST..MACLIB',
//        MACLIB2='&PROJECT..&GROUP.&C1ST2..MACLIB',
//        MACLIB3='&PROJECT..&GROUP.&STG3..MACLIB',
//        MACLIB4='&PROJECT..&GROUP.&STG4..MACLIB',
//        MEMBER=&C1ELEMENT.,
//        MONITOR=COMPONENTS,
//        PARMASM='NODECK,OBJECT,NOTERM,XREF,NOUSING',
//        PARMLNK='LIST,MAP,RENT,XREF',
//        SYSOUT=*,
//        WRKUNIT=TDISK
//*********************************************************************
//*  ALLOCATE TEMPORARY LISTING DATASETS                              *
//*********************************************************************
//INIT   EXEC PGM=BC1PDSIN,MAXRC=0
//C1INIT01 DD DSN=&&ASMLIST,DISP=(,PASS),
//            UNIT=&WRKUNIT.,SPACE=(TRK,(100,100),RLSE),
//            DCB=(RECFM=FBA,LRECL=125,BLKSIZE=0,DSORG=PS)
//C1INIT02 DD DSN=&&LNKLIST,DISP=(,PASS),
//            UNIT=&WRKUNIT.,SPACE=(TRK,(10,10),RLSE),
//            DCB=(RECFM=FBA,LRECL=125,BLKSIZE=0,DSORG=PS)
```

```
//*********************************************************************
//* READ SOURCE AND EXPAND INCLUDES                                   *
//*********************************************************************
//CONWRITE EXEC PGM=CONWRITE,COND=(0,LT),MAXRC=0,
// PARM='EXPINCL(&EXPINC)'.
//ELMOUT  DD DSN=&&ELMOUT,DISP=(,PASS),
//           UNIT=&WRKUNIT.,SPACE=(TRK,(100,100),RLSE),
//           DCB=(RECFM=FB,LRECL=80,BLKSIZE=0),
//           MONITOR=&MONITOR.
//*********************************************************************
//*   ASSEMBLE THE ELEMENT                                          **
//*********************************************************************
//ASM   EXEC PGM=ASMA90,COND=(0,LT),MAXRC=4,
// PARM='&PARMASM'.
//SYSLIB   DD DSN=&MACLIB1.,
//            MONITOR=&MONITOR.,
//            DISP=SHR
//         DD DSN=&MACLIB2.,
//            MONITOR=&MONITOR.,
//            DISP=SHR
//         DD DSN=&MACLIB3.,
//            MONITOR=&MONITOR.,
//            DISP=SHR
//         DD DSN=&MACLIB4.,
//            MONITOR=&MONITOR.,
//            DISP=SHR
//         DD DSN=&MACLIB.,
//            DISP=SHR
//SYSIN    DD DSN=&&ELMOUT,DISP=(OLD,DELETE)
//SYSLIN   DD DSN=&&SYSLIN,DISP=(,PASS,DELETE),
//            UNIT=&WRKUNIT.,SPACE=(TRK,(100,100),RLSE),
//            DCB=(RECFM=FB,LRECL=80,BLKSIZE=0),
//            FOOTPRNT=CREATE
//SYSPUNCH DD DUMMY
//SYSUT1   DD UNIT=&WRKUNIT.,SPACE=(CYL,(1,1))
//SYSUT2   DD UNIT=&WRKUNIT.,SPACE=(CYL,(1,1))
//SYSUT3   DD UNIT=&WRKUNIT.,SPACE=(CYL,(1,1))
//SYSPRINT DD DSN=&&ASMLIST,DISP=(OLD,PASS)
//*********************************************************************
//*   LINK EDIT THE ELEMENT                                         **
//*********************************************************************
//LKED   EXEC PGM=IEWL,COND=(4,LT),MAXRC=4,
// PARM='&PARMLNK'.
//SYSLIN   DD DSN=&&SYSLIN,DISP=(OLD,DELETE)
//SYSLMOD  DD DSN=&LOADLIB(&MEMBER).,
//            MONITOR=&MONITOR.,
//            FOOTPRNT=CREATE,
//            DISP=SHR
//SYSLIB   DD DSN=&LSYSLIB1.,
```

```
//            MONITOR=&MONITOR.,
//            DISP=SHR
//         DD DSN=&LSYSLIB2.,
//            MONITOR=&MONITOR.,
//            DISP=SHR
//         DD DSN=&LSYSLIB3.,
//            MONITOR=&MONITOR.,
//            DISP=SHR
//         DD DSN=&LSYSLIB4.,
//            MONITOR=&MONITOR.,
//            DISP=SHR
//SYSUT1   DD UNIT=&WRKUNIT.,SPACE=(CYL,(1,1))
//SYSPRINT DD DSN=&&LNKLIST,DISP=(OLD,PASS)
//*********************************************************************
//*   STORE THE LISTINGS IF: &LISTLIB=LISTING. LIBRARY NAME         *
//*********************************************************************
//STORLIST EXEC PGM=CONLIST,MAXRC=0,PARM=STORE,COND=EVEN,
//              EXECIF=(&LISTLIB.,NE,NO)
//C1LLIBO    DD DSN=&LISTLIB.,DISP=SHR,
//              MONITOR=&MONITOR.
//C1BANNER   DD UNIT=&WRKUNIT.,SPACE=(TRK,(1,1)),
//              DCB=(RECFM=FBA,LRECL=121,BLKSIZE=0)
//LIST01     DD DSN=&&ASMLIST,DISP=(OLD,DELETE)
//LIST02     DD DSN=&&LNKLIST,DISP=(OLD,DELETE)
//*********************************************************************
//*   PRINT THE LISTINGS IF: &LISTLIB=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=0,DSORG=PS)
//C1PRINT    DD SYSOUT=&SYSOUT.,
//              DCB=(RECFM=FBA,LRECL=133,BLKSIZE=0,DSORG=PS)
//LIST01     DD DSN=&&ASMLIST,DISP=(OLD,DELETE)
//LIST02     DD DSN=&&LNKLIST,DISP=(OLD,DELETE)
```

The GCIIDBL Generate Processor

Generate **C**obol **II,** for **D**B2, for **B**atch, and create a **L**oad module.

```
//*******************************************************************
//**                                                               **
//** PERFORMS A DB2 PRECOMPILE, COBOL2 COMPILE AND LINK EDIT       **
//** BINDS THE DB2 APPL PLAN                                       **
//**                                                               **
//*******************************************************************
//GCIIDBL PROC LISTLIB='&PROJECT..&GROUP.&C1ST..LISTLIB',
//             CLECOMP='SYSCLECOMP',                        *SIGYCOMP
//             CLERUN='SYSCLERUN',                          *SCEERUN
//             CLELKED='SYSCLELKED',                        *SCEELKED
//             CIILIB='SYSCIILIB',
//             CIICOMP='SYSCIICOMP',
//             DB2SYS='DB2SYSTEM',
//             DB2LOADL='SYSDB2LIB',
//             DBRMLIB='&PROJECT..&GROUP.&C1ST..DBRMLIB',
//             PROJECT='IPRFX.IQUAL',
//             GROUP='SMPL',
//             STG1='&C1ST.',    CURRENT ENV STAGE 2 NAME
//             STG2='&C1ST2.',   CURRENT ENV STAGE 2 NAME
//             STG3='EMER',      EMER STAGE
//             STG4='PROD',      PROD STAGE
//             CSYSLIB1='&PROJECT..&GROUP.&STG1..COPYLIB',
//             CSYSLIB2='&PROJECT..&GROUP.&STG2..COPYLIB',
//             CSYSLIB3='&PROJECT..&GROUP.&STG3..COPYLIB',
//             CSYSLIB4='&PROJECT..&GROUP.&STG4..COPYLIB',
//             EXPINC=N,
//             LOADLIB='&PROJECT..&GROUP.&C1ST..LOADLIB',
//             LSYSLIB1='&LOADLIB'.,
//             LSYSLIB2='&PROJECT..&GROUP..&STG2..LOADLIB',
//             LSYSLIB3='&PROJECT..&GROUP..&STG3..LOADLIB',
//             LSYSLIB4='&PROJECT..&GROUP..&STG4..LOADLIB',
//             MEMBER=&C1ELEMENT.,
//             MONITOR=COMPONENTS,
//             PARMLIB='&PROJECT..&GROUP.&C1ST..PARMLIB',
//             PARMPC='HOST(COB2),APOST,APOSTSQL',
//             PARMCOB='LIB,NOSEQ,OBJECT,APOST,',
//             PARMLNK='LIST,MAP,XREF',
//             PLAN=&MEMBER.,
//             SYSOUT=A,
//             WRKUNIT=TDISK
```

```
//*********************************************************************
//*   ALLOCATE TEMPORARY LISTING DATASETS                             *
//*********************************************************************
//INIT      EXEC PGM=BC1PDSIN,MAXRC=0
//C1INIT01    DD DISP=(,PASS),DSN=&&DB2PLIST,
//               UNIT=&WRKUNIT.,SPACE=(TRK,(10,10)),
//               DCB=(RECFM=FBA,LRECL=133,BLKSIZE=0)
//C1INIT02    DD DISP=(,PASS),DSN=&&COBLIST,
//               UNIT=&WRKUNIT.,SPACE=(TRK,(10,10)),
//               DCB=(RECFM=FBA,LRECL=133,BLKSIZE=0)
//C1INIT03    DD DISP=(,PASS),DSN=&&LNKLIST,
//               UNIT=&WRKUNIT.,SPACE=(TRK,(10,10)),
//               DCB=(RECFM=FBA,LRECL=125,BLKSIZE=0)
//C1INIT04    DD DISP=(,PASS),DSN=&&PARMLIST,
//               UNIT=&WRKUNIT.,SPACE=(TRK,(10,10)),
//               DCB=(RECFM=FBA,LRECL=133,BLKSIZE=0)
//*********************************************************************
//* READ SOURCE AND EXPAND INCLUDES                                   *
//*********************************************************************
//CONWRITE EXEC PGM=CONWRITE,COND=(0,LT),MAXRC=0,
// PARM='EXPINCL(&EXPINC)'.
//C1INCL01   DD DSN=&CSYSLIB1.,DISP=SHR,
//              MONITOR=&MONITOR.
//C1INCL02   DD DSN=&CSYSLIB2.,DISP=SHR,
//              MONITOR=&MONITOR.
//C1INCL03   DD DSN=&CSYSLIB3.,DISP=SHR,
//              MONITOR=&MONITOR.
//C1INCL04   DD DSN=&CSYSLIB4.,DISP=SHR,
//              MONITOR=&MONITOR.
//ELMOUT     DD DSN=&&ELMOUT,DISP=(,PASS),
//              UNIT=&WRKUNIT.,SPACE=(TRK,(10,10),RLSE),
//              DCB=(RECFM=FB,LRECL=80,BLKSIZE=0),
//              MONITOR=&MONITOR.
//*********************************************************************
//*  DB2 PRECOMPILIER PROCESSING                                      *
//*********************************************************************
//PRECOMP EXEC PGM=DSNHPC,COND=(0,NE),MAXRC=4,
// PARM='&PARMPC'.
//STEPLIB   DD DSN=&DB2LOADL.,DISP=SHR
//DBRMLIB   DD DSN=&DBRMLIB(&MEMBER).,DISP=SHR,
//             MONITOR=&MONITOR.,
//             FOOTPRNT=CREATE
//SYSIN     DD DSN=&&ELMOUT,DISP=OLD
//SYSCIN    DD DSN=&&PREOUT,DISP=(,PASS),
//             UNIT=&WRKUNIT.,SPACE=(TRK,(10,10),RLSE),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=0)
//SYSLIB    DD DSN=&CSYSLIB1.,
//             MONITOR=&MONITOR.,
//             DISP=SHR
```

```
//          DD DSN=&CSYSLIB2.,
//             MONITOR=&MONITOR.,
//             DISP=SHR
//          DD DSN=&CSYSLIB3.,
//             MONITOR=&MONITOR.,
//             DISP=SHR
//          DD DSN=&CSYSLIB4.,
//             MONITOR=&MONITOR.,
//             DISP=SHR
//SYSTERM   DD SYSOUT=&SYSOUT.
//SYSPRINT  DD DSN=&&DB2PLIST,DISP=(OLD,PASS)
//SYSUT1    DD UNIT=&WRKUNIT.,SPACE=(CYL,(1,1))
//SYSUT2    DD UNIT=&WRKUNIT.,SPACE=(CYL,(1,1))
//*****************************************************************
//**  COMPILE THE ELEMENT                                        **
//*****************************************************************
//COMPILE EXEC PGM=IGYCRCTL,COND=(0,LT),MAXRC=4,
// PARM='&PARMCOB'.
//* TEST PROCESSOR GROUP IF CIINBL THEN ALLOCATE COBOL2 LIBRARIES
//*           IF CLENBL ALLOCATE COBOL/MVS RUNTIME LIBS
// IF &C1PRGRP=CIINBL. THEN
//STEPLIB  DD DSN=&CIICOMP.,DISP=SHR
//         DD DSN=&CIILIB.,DISP=SHR
// ELSE
//* PROCESSOR GROUP IS COBOL/LE
//STEPLIB  DD DSN=&CLECOMP.,DISP=SHR
//         DD DSN=&CLERUN.,DISP=SHR
// ENDIF
//*****************************************************************
//*   COPYLIB CONCATENATIONS                                      **
//*****************************************************************
//SYSLIB   DD DSN=&CSYSLIB1.,
//            MONITOR=&MONITOR.,
//            DISP=SHR
//         DD DSN=&CSYSLIB2.,
//            MONITOR=&MONITOR.,
//            DISP=SHR
//         DD DSN=&CSYSLIB3.,
//            MONITOR=&MONITOR.,
//            DISP=SHR
//         DD DSN=&CSYSLIB4.,
//            MONITOR=&MONITOR.,
//            DISP=SHR
//SYSIN    DD DSN=&&PREOUT,DISP=(OLD,DELETE)
//SYSLIN   DD DSN=&&SYSLIN,DISP=(,PASS,DELETE),
//            UNIT=&WRKUNIT.,SPACE=(TRK,(10,10),RLSE),
//            DCB=(RECFM=FB,LRECL=80,BLKSIZE=0),
//            FOOTPRNT=CREATE
//SYSUT1   DD UNIT=&WRKUNIT.,SPACE=(CYL,(5,3))
```

```
//SYSUT2   DD UNIT=&WRKUNIT.,SPACE=(CYL,(5,3))
//SYSUT3   DD UNIT=&WRKUNIT.,SPACE=(CYL,(5,3))
//SYSUT4   DD UNIT=&WRKUNIT.,SPACE=(CYL,(5,3))
//SYSUT5   DD UNIT=&WRKUNIT.,SPACE=(CYL,(5,3))
//SYSUT6   DD UNIT=&WRKUNIT.,SPACE=(CYL,(5,3))
//SYSUT7   DD UNIT=&WRKUNIT.,SPACE=(CYL,(5,3))
//SYSPRINT DD DSN=&&COBLIST,DISP=(OLD,PASS)
//*****************************************************************
//**  LINK EDIT THE ELEMENT                                      **
//*****************************************************************
//LKED   EXEC PGM=IEWL,COND=(4,LT),MAXRC=4,
// PARM='&PARMLNK'.
//SYSLIN   DD DSN=&&SYSLIN,DISP=(OLD,DELETE)
//SYSLMOD  DD DSN=&LOADLIB(&MEMBER).,
//            MONITOR=&MONITOR.,
//            FOOTPRNT=CREATE,
//            DISP=SHR
//SYSLIB   DD DSN=&LSYSLIB1.,
//            MONITOR=&MONITOR.,
//            DISP=SHR
//         DD DSN=&LSYSLIB2.,
//            MONITOR=&MONITOR.,
//            DISP=SHR
//         DD DSN=&LSYSLIB3.,
//            MONITOR=&MONITOR.,
//            DISP=SHR
//         DD DSN=&LSYSLIB4.,
//            MONITOR=&MONITOR.,
//            DISP=SHR
//         DD DSN=&DB2LOADL.,
//            DISP=SHR
//* IF PROCESSOR GROUP IS CIINBL THEN ALLOC COB2 CALL LIBRARY COB2LIB
// IF &C1PRGRP=CIINBL. THEN
//         DD DSN=&CIILIB.,
//            DISP=SHR
// ELSE
//* IF PROCESSOR GROUP IS COBOL/LE THEN ALLOC LE CALL LIBRARY SCEELKED
//         DD DSN=&CLELKED.,
//            DISP=SHR
// ENDIF
//SYSUT1   DD UNIT=&WRKUNIT.,SPACE=(CYL,(1,1))
//SYSPRINT DD DSN=&&LNKLIST,DISP=(OLD,PASS)
//*****************************************************************
//* BIND APPLICATION PLAN IF EXECUTING IN FOREGROUND
//* NOTE: ATTEMPTING TO RUN THIS STEP IN BG WILL RESULT IN RC=5
//*****************************************************************
//BINDFG EXEC PGM=BC1PTMP0,MAXRC=5,COND=(4,LT),
// PARM='&PARMLIB(&C1ELEMENT)'.
//STEPLIB  DD DSN=&DB2LOADL.,DISP=SHR
```

```
//DBRMLIB  DD DSN=&DBRMLIB.,DISP=SHR
//SYSUDUMP DD SYSOUT=&SYSOUT.
//*********************************************************************
//* BIND APPLICATION PLAN IF EXECUTING IN BACKGROUND                  *
//*********************************************************************
//BINDBG EXEC PGM=IKJEFT01,COND=((5,NE,BINDFG),(5,LT)),MAXRC=7
//STEPLIB  DD DSN=&DB2LOADL.,DISP=SHR
//DBRMLIB  DD DSN=&DBRMLIB.,DISP=SHR
//SYSTSPRT DD DSN=&&PARMLIST,DISP=(OLD,PASS)
//SYSTSIN  DD DSN=&PARMLIB(&C1ELEMENT).,DISP=(OLD,PASS)
//*********************************************************************
//*   STORE THE LISTINGS IF: &LISTLIB=LISTING. LIBRARY NAME         *
//*********************************************************************
//STORLIST EXEC PGM=CONLIST,MAXRC=0,PARM=STORE,COND=EVEN,
//      EXECIF=(&LISTLIB.,NE,NO)
//C1LLIBO   DD DSN=&LISTLIB.,DISP=SHR,
//           MONITOR=&MONITOR.
//C1BANNER  DD UNIT=&WRKUNIT.,SPACE=(TRK,(1,1)),
//             DCB=(RECFM=FBA,LRECL=121,BLKSIZE=6171)
//LIST01    DD DSN=&&DB2PLIST,DISP=(OLD,DELETE)
//LIST02    DD DSN=&&COBLIST,DISP=(OLD,DELETE)
//LIST03    DD DSN=&&LNKLIST,DISP=(OLD,DELETE)
//LIST04    DD DSN=&&PARMLIST,DISP=(OLD,DELETE)
//*********************************************************************
//*   PRINT THE LISTINGS IF: &LISTLIB=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=&&DB2PLIST,DISP=(OLD,DELETE)
//LIST02    DD DSN=&&COBLIST,DISP=(OLD,DELETE)
//LIST03    DD DSN=&&LNKLIST,DISP=(OLD,DELETE)
//LIST04    DD DSN=&&PARMLIST,DISP=(OLD,DELETE)
//**
```

The GCIINBL Generate Processor

Generate **C**obol **II**, **N**o database, for **B**atch, and create a **L**oad module. This processor uses If-Then-Else logic to specify the proper system libraries using the COBOL version selected during installation. Values are:

- CII-COBOL II
- CLE-COBOL/LE
- COBOL-COBOL for z/OS

```
//*****************************************************************
//**                                                             **
//**  COBOL2 AND COBOL/MVS COMPILE AND LINK-EDIT PROCESSOR
//**                                                             **
//*****************************************************************
//GCIINBL PROC LISTLIB='&PROJECT..&GROUP.&C1ST..LISTLIB',
//             CLECOMP='SYSCLECOMP',                        *SIGYCOMP
//             CLERUN='SYSCLERUN',                          *SCEERUN
//             CLELKED='SYSCLELKED',                        *SCEELKED
//             CIILIB='SYSCIILIB',                          *COB2LIB
//             CIICOMP='SYSCIICOMP',                        *COB2COMP
//             PROJECT='IPRFX.IQUAL',
//             GROUP='SMPL',
//             STG3='EMER',             SMPLPROD/EMER STAGE
//             STG4='PROD',             SMPLPROD/PROD STAGE
//             CSYSLIB1='&PROJECT..&GROUP.&C1ST..COPYLIB',
//             CSYSLIB2='&PROJECT..&GROUP.&C1ST2..COPYLIB',
//             CSYSLIB3='&PROJECT..&GROUP.&STG3..COPYLIB',
//             CSYSLIB4='&PROJECT..&GROUP.&STG4..COPYLIB',
//             EXPINC=N,
//             LOADLIB='&PROJECT..&GROUP.&C1ST..LOADLIB',
//             LSYSLIB1='&LOADLIB'.,
//             LSYSLIB2='&PROJECT..&GROUP.&C1ST2..LOADLIB',
//             LSYSLIB3='&PROJECT..&GROUP.&STG3..LOADLIB',
//             LSYSLIB4='&PROJECT..&GROUP.&STG4..LOADLIB',
//             MEMBER=&C1ELEMENT.,
//             MONITOR=COMPONENTS,
//             PARMCOB='LIB,NOSEQ,OBJECT,APOST,',
//             PARMLNK='LIST,MAP,XREF',
//             SYSOUT=*,
//             WRKUNIT=TDISK
```

```
//*********************************************************************
//*   ALLOCATE TEMPORARY LISTING DATASETS                              *
//*********************************************************************
//INIT     EXEC PGM=BC1PDSIN,MAXRC=0
//C1INIT01   DD DSN=&&COBLIST,DISP=(,PASS),
//              UNIT=&WRKUNIT.,SPACE=(TRK,(10,10)),
//              DCB=(RECFM=FBA,LRECL=133,BLKSIZE=0,DSORG=PS)
//C1INIT02   DD DSN=&&LNKLIST,DISP=(,PASS),
//              UNIT=&WRKUNIT.,SPACE=(TRK,(10,10)),
//              DCB=(RECFM=FBA,LRECL=125,BLKSIZE=0,DSORG=PS)
//*********************************************************************
//* READ SOURCE AND EXPAND INCLUDES                                   *
//*********************************************************************
//CONWRITE EXEC PGM=CONWRITE,COND=(0,LT),MAXRC=0,
// PARM='EXPINCL(&EXPINC)'.
//ELMOUT     DD DSN=&&ELMOUT,DISP=(,PASS),
//              UNIT=&WRKUNIT.,SPACE=(TRK,(100,100),RLSE),
//              DCB=(RECFM=FB,LRECL=80,BLKSIZE=0),
//              MONITOR=&MONITOR.
//*********************************************************************
//**  COMPILE THE ELEMENT                                            **
//*********************************************************************
//COMPILE EXEC PGM=IGYCRCTL,COND=(0,LT),MAXRC=4,
// PARM='&PARMCOB'.
//* TEST PROCESSOR GROUP IF CIINBL THEN ALLOCATE COBOL2 LIBRARIES
//*           IF CLENBL ALLOCATE COBOL/MVS RUNTIME LIBS
// IF &C1PRGRP=CIINBL. THEN
//STEPLIB   DD DSN=&CIICOMP.,DISP=SHR
//          DD DSN=&CIILIB.,DISP=SHR
// ELSE
//* PROCESSOR GROUP IS COBOL/LE
//STEPLIB   DD DSN=&CLECOMP.,DISP=SHR
//          DD DSN=&CLERUN.,DISP=SHR
// ENDIF
//*********************************************************************
//*   COPYLIB CONCATENATIONS                                          **
//*********************************************************************
//SYSLIB   DD DSN=&CSYSLIB1.,
//            MONITOR=&MONITOR.,
//            DISP=SHR
//         DD DSN=&CSYSLIB2.,
//            MONITOR=&MONITOR.,
//            DISP=SHR
//         DD DSN=&CSYSLIB3.,
//            MONITOR=&MONITOR.,
//            DISP=SHR
//         DD DSN=&CSYSLIB4.,
//            MONITOR=&MONITOR.,
//            DISP=SHR
```

```
//SYSIN    DD DSN=&&ELMOUT,DISP=(OLD,PASS)
//SYSLIN   DD DSN=&&SYSLIN,DISP=(,PASS,DELETE),
//            UNIT=&WRKUNIT.,SPACE=(TRK,(100,100),RLSE),
//            DCB=(RECFM=FB,LRECL=80,BLKSIZE=0),
//            FOOTPRNT=CREATE
//SYSUT1   DD UNIT=&WRKUNIT.,SPACE=(CYL,(5,3))
//SYSUT2   DD UNIT=&WRKUNIT.,SPACE=(CYL,(5,3))
//SYSUT3   DD UNIT=&WRKUNIT.,SPACE=(CYL,(5,3))
//SYSUT4   DD UNIT=&WRKUNIT.,SPACE=(CYL,(5,3))
//SYSUT5   DD UNIT=&WRKUNIT.,SPACE=(CYL,(5,3))
//SYSUT6   DD UNIT=&WRKUNIT.,SPACE=(CYL,(5,3))
//SYSUT7   DD UNIT=&WRKUNIT.,SPACE=(CYL,(5,3))
//SYSPRINT DD DSN=&&COBLIST,DISP=(OLD,PASS)
//*****************************************************************
//**  LINK EDIT THE ELEMENT                                      **
//*****************************************************************
//LKED   EXEC PGM=IEWL,COND=(4,LT),MAXRC=4,
// PARM='&PARMLNK'.
//SYSLIN   DD DSN=&&SYSLIN,DISP=(OLD,DELETE)
//SYSLMOD  DD DSN=&LOADLIB(&MEMBER).,
//            MONITOR=&MONITOR.,
//            FOOTPRNT=CREATE,
//            DISP=SHR
//SYSLIB   DD DSN=&LSYSLIB1.,
//            MONITOR=&MONITOR.,
//            DISP=SHR
//         DD DSN=&LSYSLIB2.,
//            MONITOR=&MONITOR.,
//            DISP=SHR
//         DD DSN=&LSYSLIB3.,
//            MONITOR=&MONITOR.,
//            DISP=SHR
//         DD DSN=&LSYSLIB4.,
//            MONITOR=&MONITOR.,
//            DISP=SHR
//* IF PROCESSOR GROUP IS CIINBL THEN ALLOC COB2 CALL LIBRARY COB2LIB
// IF &C1PRGRP=CIINBL. THEN
//        DD DSN=&CIILIB.,
//           DISP=SHR
// ELSE
//* IF PROCESSOR GROUP IS COBOL/LE THEN ALLOC LE CALL LIBRARY SCEELKED
//         DD DSN=&CLELKED.,
//            DISP=SHR
// ENDIF
//SYSUT1   DD UNIT=&WRKUNIT.,SPACE=(CYL,(1,1))
//SYSPRINT DD DSN=&&LNKLIST,DISP=(OLD,PASS)
//*****************************************************************
//*   STORE THE LISTINGS IF: &LISTLIB=LISTING. LIBRARY NAME         *
//*****************************************************************
```

```
//STORLIST EXEC PGM=CONLIST,MAXRC=0,PARM=STORE,COND=EVEN,
//      EXECIF=(&LISTLIB.,NE,NO)
//C1LLIBO DD DSN=&LISTLIB.,DISP=SHR,
//      MONITOR=&MONITOR.
//C1BANNER DD UNIT=&WRKUNIT.,SPACE=(TRK,(1,1)),
//      DCB=(RECFM=FBA,LRECL=121,BLKSIZE=0)
//LIST01  DD DSN=&&COBLIST,DISP=(OLD,DELETE)
//LIST02  DD DSN=&&LNKLIST,DISP=(OLD,DELETE)
//*****************************************************************
//*   PRINT THE LISTINGS IF: &LISTLIB=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=0,DSORG=PS)
//C1PRINT  DD SYSOUT=&SYSOUT.,
//            DCB=(RECFM=FBA,LRECL=133,BLKSIZE=0,DSORG=PS)
//LIST01   DD DSN=&&COBLIST,DISP=(OLD,DELETE)
//LIST02   DD DSN=&&LNKLIST,DISP=(OLD,DELETE)
//**
```

The GLNKNBL Generate Processor

Generate **Lin**k-edit only **N**o database for **B**atch and create a **L**oad module.

```
//*******************************************************************
//**                                                               **
//** LINK-EDIT ONLY PROCESSOR (COMPOSITE LINK) FOR TYPE LINKCARD   **
//**  (USING ASSEMBLER OR COBOL OBJECT MODULES)                    **
//**                                                               **
//*******************************************************************
//*
//GLNKNBL PROC LISTLIB='&PROJECT..&GROUP.&STG1..LISTLIB',
//             LOADLIB='&PROJECT..&GROUP.&STG1..LOADLIB',
//             CII='C??',
//             CIILIB='SYSCIILIB',                          **COB2LIB
//             CLELIB='SYSCLELIB',                         **SCEELKED
//             EXPINC=N,
//             PROJECT='IPRFX.IQUAL',
//             GROUP='SMPL',
//             LSYSLIB1='&PROJECT..&GROUP.&C1ST..LOADLIB',
//             LSYSLIB2='&PROJECT..&GROUP.&C1ST2..LOADLIB',
//             LSYSLIB3='&PROJECT..&GROUP.&STG3..LOADLIB',
//             LSYSLIB4='&PROJECT..&GROUP.&STG4..LOADLIB',
//             STG3='EMER',
//             STG4='PROD',
//             MEMBER=&C1ELEMENT.,
//             MONITOR=COMPONENTS,
//             OBJLIB1='&PROJECT..&GROUP.&C1ST..OBJLIB',
//             OBJLIB2='&PROJECT..&GROUP.&C1ST2..OBJLIB',
//             OBJLIB3='&PROJECT..&GROUP.&STG3..OBJLIB',
//             OBJLIB4='&PROJECT..&GROUP.&STG4..OBJLIB',
//             PARMLNK='LIST,MAP,XREF',
//             WRKUNIT=TDISK
//*********************************************************************
//*  ALLOCATE TEMPORARY LISTING DATASETS                              *
//*********************************************************************
//INIT   EXEC PGM=BC1PDSIN,MAXRC=0
//C1INIT01  DD DSN=&&LNKLIST,DISP=(,PASS),
//        UNIT=&WRKUNIT.,SPACE=(CYL,(1,2),RLSE),
//        DCB=(RECFM=FBA,LRECL=125,BLKSIZE=0,DSORG=PS)
//*******************************************************************
//**   LINK EDIT THE ELEMENT                                       **
//*******************************************************************
//LKED    EXEC PGM=IEWL,COND=(0,LT),MAXRC=0,
// PARM='&PARMLNK'.
//SYSPRINT  DD DSN=&&LNKLIST,DISP=(OLD,PASS)
```

```
//SYSUT1    DD UNIT=TDISK,SPACE=(CYL,(1,1))
//SYSLIB    DD DSN=&LSYSLIB1.,DISP=SHR,
//             MONITOR=COMPONENTS
//          DD DSN=&LSYSLIB2.,DISP=SHR,
//             MONITOR=COMPONENTS
//          DD DSN=&LSYSLIB3.,DISP=SHR,
//             MONITOR=COMPONENTS
//          DD DSN=&LSYSLIB4.,DISP=SHR,
//             MONITOR=COMPONENTS
// IF &CII=CII. THEN
//          DD DSN=&CIILIB.,DISP=SHR
// ELSE
//          DD DSN=&CLELIB.,DISP=SHR
// ENDIF
//OBJLIB    DD DSN=&OBJLIB1.,DISP=SHR,
//             MONITOR=COMPONENTS
//          DD DSN=&OBJLIB2.,DISP=SHR,
//             MONITOR=COMPONENTS
//          DD DSN=&OBJLIB3.,DISP=SHR,
//             MONITOR=COMPONENTS
//          DD DSN=&OBJLIB4.,DISP=SHR,
//             MONITOR=COMPONENTS
//SYSLIN    DD DSN=&C1BASELIB(&C1ELEMENT).,DISP=SHR
//SYSLMOD   DD DSN=&LOADLIB.,DISP=SHR,
//             FOOTPRNT=CREATE,
//             MONITOR=COMPONENTS
//*******************************************************************
//*   STORE THE LISTINGS IF: &LISTING=LISTING. LIBRARY NAME         *
//*******************************************************************
//STORLIST EXEC PGM=CONLIST,MAXRC=0,PARM=STORE,COND=EVEN,
//      EXECIF=(&LISTLIB.,NE,NO)
//C1LLIBO  DD DSN=&LISTLIB.,DISP=SHR,
//           MONITOR=&MONITOR.
//C1BANNER DD UNIT=&WRKUNIT.,SPACE=(TRK,(1,1)),
//            DCB=(RECFM=FBA,LRECL=121,BLKSIZE=6171)
//LIST01   DD DSN=&&LNKLIST,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=*,
//            DCB=(RECFM=FBA,LRECL=133,BLKSIZE=0,DSORG=PS)
//LIST01   DD DSN=&&LNKLIST,DISP=(OLD,DELETE)
//*
```

The LOADONLY Generate Processor

The LOADONLY processor can be used as a generate or a move processor. The processing varies depending on the type of processor requested.

- **Generate**-Copies the requested element, file, or member from the user's file into and CA Endevor SCM managed library.
- **Move**-Copies the member from one stage's output library to the next stage's output library.

```
//*********************************************************************
//*                                                                   *
//* COPY SOURCELES LOAD MODULES FROM USER DATA SETS TO STAGE1         *
//*                                                                   *
//*********************************************************************
//*
//LOADONLY PROC LISTLIB='&PROJECT..&GROUP.SMPL&C1ST..LISTLIB',
//              LOADLIB1='&PROJECT..&GROUP.SMPL&C1ST..LOADLIB',
//              LOADLIB2='&PROJECT..&GROUP.SMPL&STG2..LOADLIB',
//              PROJECT='IPRFX',
//              GROUP='IQUAL',
//              STG2='&C1SSTAGE.', FROM STAGE FOR TRANSFER/MOVE
//              MONITOR=COMPONENTS,
//              SYSOUT=*,
//              WRKUNIT=TDISK
//*********************************************************************
//*  ALLOCATE TEMPORARY LISTING DATASETS                              *
//*********************************************************************
//INIT   EXEC PGM=BC1PDSIN
//C1INIT01 DD DSN=&&COPYLIST,
//            DISP=(NEW,PASS,DELETE),
//            UNIT=&WRKUNIT.,
//            SPACE=(CYL,(1,2),RLSE),
//            DCB=(RECFM=FBA,LRECL=133,BLKSIZE=0,DSORG=PS)
//*********************************************************************
//*                                                                   *
//* ONLY PERFORM COPY FROM USER LOADLIB WHEN ADD OR UPDATE IS REQUESTED*
//*                                                                   *
//*********************************************************************
```

```
//IF1 IF ((&C1ACTION=ADD). OR (&C1ACTION=UPDATE)). THEN
//ADD    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=&C1USRDSN.,DISP=SHR
//OUTDD    DD DSN=&LOADLIB1.,DISP=SHR,FOOTPRNT=CREATE
//SYSIN    DD *
 COPY  I=INDD,O=OUTDD
 SELECT MEMBER=((&C1USRMBR.,&C1ELEMENT.,R))
//END1 ENDIF
//*********************************************************************
//*                                                                   *
//* IF TRANSFER,COPY FROM SOURCE LOADLIB TO CURRENT LOADLIB           *
//*                                                                   *
//*********************************************************************
//IF2 IF (&C1ACTION=TRANSFER). OR (&C1ACTION=MOVE). THEN
//TRANSFER 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,FOOTPRNT=VERIFY
//OUTDD DD DSN=&LOADLIB2.,DISP=SHR,FOOTPRNT=CREATE
//SYSIN    DD *
 COPY  I=INDD,O=OUTDD
 SELECT MEMBER=((&C1ELEMENT.,,R))
//END2 ENDIF
//*********************************************************************
//*      STORE THE LISTINGS IF: &LISTLIB=LISTING. LIBRARY           *
//*********************************************************************
//STORLIST EXEC PGM=CONLIST,PARM='STORE',COND=EVEN,
//      EXECIF=(&LISTLIB.,NE,NO)
//C1LLIBO    DD DSN=&LISTLIB.,DISP=SHR
//C1BANNER   DD DSN=&&BANNER,DISP=(,PASS,DELETE),
//              UNIT=&WRKUNIT.,SPACE=(TRK,(1,1)),
//              DCB=(RECFM=FBA,LRECL=121,BLKSIZE=0,DSORG=PS)
//LIST01     DD DSN=&&COPYLIST,DISP=(OLD,DELETE)
//*********************************************************************
//*   PRINT THE LISTINGS IF: &LISTLIB=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=0,DSORG=PS)
//C1PRINT   DD SYSOUT=&SYSOUT.,
//             DCB=(RECFM=FBA,LRECL=133,BLKSIZE=0,DSORG=PS)
//LIST01    DD DSN=&&COPYLIST,DISP=(OLD,DELETE)
//*
//**
```

Delete Processors

Delete processors clean up (delete) what generate processors create. Since most generate processors create a load module and possibly a listing, delete processors can be generic. A single delete processor can be used to clean up the work created by several generate processors.

This section contains two sample delete processors. Samples are provided in the iprfx.iqual.CSIQSAMP. These processors are:

DLODDNL

Delete **Lo**a**d** modules, for **D**B2, **N**o (All) operating environments, **N**o (All) output types.

DLODNNL

Delete **Lo**a**d** modules, **N**o (All) databases, **N**o (All) operating environments, **N**o (All) output types. These processors can be used as Stage 1 or Stage 2 delete processors. To change the PROC statement symbol values, override the data set names during processor group definition.

If you do not want to store listings, change the &LISTLIB symbol to NO.

The DLODDNL Delete Processor

Delete **LO**a**d** modules, for **D**B2, **N**o (All) operating environments, **N**o (All) output types.

```
//*********************************************************************
//*                                                                   *
//* DELETE COBOL LOAD AND LISTING MODULES & DBRM                      *
//*                                                                   *
//*********************************************************************
//*
//DLODDNL PROC DBRMLIB='&PROJECT..&GROUP.&STG1..DBRMLIB',
//             LISTLIB='&PROJECT..&GROUP.&STG1..LISTLIB',
//             LOADLIB='&PROJECT..&GROUP.&STG1..LOADLIB',
//             PROJECT='IPRFX.IQUAL',
//             GROUP='SMPL',
//             STG1='&C1STAGE.'   CURRENT STAGE
//*
//DELLOD  EXEC PGM=CONDELE,MAXRC=12
//C1LIB     DD DSN=&LOADLIB.,DISP=SHR
//*
//DELDBRM EXEC PGM=CONDELE,MAXRC=12
//C1LIB     DD DSN=&DBRMLIB.,DISP=SHR
//*
//DELLIST EXEC PGM=CONLIST,PARM='DELETE',MAXRC=12,COND=EVEN,
//       EXECIF=(&LISTLIB.,NE,NO)
//C1LLIBI   DD DSN=&LISTLIB.,DISP=SHR
```

The DLODNNL Delete Processor

Delete **LO**a**d** modules, **N**o (All) databases, **N**o (All) operating environments and **L**istings

```
//*********************************************************************
//*                                                                   *
//* DELETE LOAD/OBJECT AND LISTING MODULES                            *
//*                                                                   *
//*********************************************************************
//*
//DLODNNL PROC LISTLIB='&PROJECT..&GROUP.&STG1..LISTLIB',
//             LOADLIB='&PROJECT..&GROUP.&STG1..LOADLIB',
//             PROJECT='IPRFX.IQUAL',
//             GROUP='SMPL',
//             STG1='&C1STAGE.'   CURRENT STAGE
//*
//DELMOD  EXEC PGM=CONDELE,MAXRC=12
//C1LIB     DD DSN=&LOADLIB.,DISP=SHR
//*********************************************************************
//* DELETE THE LISTING IF: &LISTLIB=LISTING. LIBRARY NAME             *
//*********************************************************************
//CONLIST EXEC PGM=CONLIST,PARM='DELETE',MAXRC=12,COND=EVEN,
//      EXECIF=(&LISTLIB.,NE,NO)
//C1LLIBI   DD DSN=&LISTLIB.,DISP=SHR
```

Move Processors

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. This section includes two sample move processors that accomplish this.
- Recreating the load module at the target stage. This occurs when a generate processor is used as a move processor.

  Note: For more information, see Generate Processors (see page 157).
- If you want to use a generate processor as a move processor, make sure to specify **G** in the PROCESSOR TO USE FOR THE MOVE ACTION field on the definition panel for the APPROPRIATE PROCESSOR GROUP.

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 sync with Master Control File information for the element.

This section contains two sample move processors that are provided in *iprfx.iqual.CSIQSAMP*. These processors copy load modules, and their associated component lists and listings, from a source to a target stage. By using symbolic overrides for LOADLIB1 and LOADLIB2, these processors can be used for almost every type in your inventory structure. If the &LISTLIB2 symbol is overridden during processor group definition to read **NO**, CA Endevor SCM copies the load modules and component lists, but prints the listings, instead of copying them to the target stage.

If ACM is not installed, or if you do not wish to copy component lists to the target stage, you can override the MONITOR symbol with any value except **components**.

The MLODDNL Move Processor

Move **Lo**a**d** Modules, for **D**B2, **N**o (All) operating environments, to **L**oad modules.

```
//*********************************************************************
//*                                                                   *
//* COPY LOAD MODULES FROM STAGE 1 TO STAGE 2 AND THEIR ASSOCIATED    *
//* COMPONENT LIST AND LISTINGS. ALSO COPY DBRM MEMBERS AND BIND      *
//* APPLICATION PLAN.                                                 *
//*                                                                   *
//*********************************************************************
//*
//MLODDNL PROC DBRMLIB1='&PROJECT..&GROUP.&STG1..DBRMLIB',
//             DBRMLIB2='&PROJECT..&GROUP.&STG2..DBRMLIB',
//             DB2SYS='DB2SYSTEM',
//             DB2LOADL='SYSDB2LIB',

//             EDB2AUTH='IPRFX.IQUAL.EDB2.AUTHLIB',
//             EDB2CONL='IPRFX.IQUAL.EDB2.LOADLIB',
//             LISTLIB='&PROJECT..&GROUP.&STG1..LISTLIB',
//             LISTLIB1='&PROJECT..&GROUP.&STG1..LISTLIB',
//             LISTLIB2='&PROJECT..&GROUP.&STG2..LISTLIB',
//             LOADLIB1='&PROJECT..&GROUP.&STG1..LOADLIB',
//             LOADLIB2='&PROJECT..&GROUP.&STG2..LOADLIB',
//             PARMLIB='&PROJECT..&GROUP.&STG2..PARMLIB',
//             PROJECT='IPRFX.IQUAL',
//             GROUP='SMPL',
//             STG1='&C1SSTAGE.',   CURRENT STAGE
//             STG2='&C1STAGE.',  TO STAGE
//             PLAN=&C1ELEMENT.,
//             MONITOR=COMPONENTS,
//             SYSOUT=A,
//             WRKUNIT=PDISK
//*********************************************************************
//*  ALLOCATE TEMPORARY LISTING DATASET                               *
//*********************************************************************
//INIT   EXEC PGM=BC1PDSIN,MAXRC=0
//C1INIT01 DD DISP=(,PASS),DSN=&&COPY1LST,
//            UNIT=&WRKUNIT.,SPACE=(TRK,(5,2),RLSE),
//            DCB=(RECFM=FBA,LRECL=121,BLKSIZE=6171)
//C1INIT02 DD DISP=(,PASS),DSN=&&COPY2LST,
```

```
//            UNIT=&WRKUNIT.,SPACE=(TRK,(5,2),RLSE),
//            DCB=(RECFM=FBA,LRECL=121,BLKSIZE=6171)
//C1INIT03 DD DISP=(,PASS),DSN=&&PARMLIST,
//            UNIT=&WRKUNIT.,SPACE=(TRK,(5,2),RLSE),
//            DCB=(RECFM=FBA,LRECL=121,BLKSIZE=6171)
//*********************************************************************
//* COPY THE LOAD MODULE                                              *
//*********************************************************************
//BSTCOPY EXEC PGM=BSTCOPY,MAXRC=04,COND=(0,LT)
//SYSPRINT  DD DSN=&&COPY1LST,DISP=OLD
//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 THE DBRM MODULE                                              *
//*********************************************************************
//BSTCOPY EXEC PGM=BSTCOPY,MAXRC=04,COND=(4,LT)
//SYSPRINT  DD DSN=&&COPY2LST,DISP=OLD
//SYSUT3    DD UNIT=&WRKUNIT.,SPACE=(TRK,(1,1))
//SYSUT4    DD UNIT=&WRKUNIT.,SPACE=(TRK,(1,1))
//INDD      DD DSN=&DBRMLIB1.,DISP=SHR
//OUTDD     DD DSN=&DBRMLIB2.,DISP=SHR,MONITOR=&MONITOR.
//SYSIN     DD *
 COPY O=OUTDD,I=INDD
 SELECT MEMBER=((&C1ELEMENT.,,R))
//*********************************************************************
//* MOVE THE COMPONENT LIST                                           *
//*********************************************************************
//*
//*
//*********************************************************************
//* BIND APPLICATION PLAN IF EXECUTING IN FOREGROUND
//* NOTE: ATTEMPTING TO RUN THIS STEP IN BG WILL RESULT IN RC=5
//*********************************************************************
//BINDFG EXEC PGM=BC1PTMP0,MAXRC=5,COND=(4,LT),
// PARM='&PARMLIB(&C1ELEMENT)'.
//STEPLIB  DD DSN=&DB2LOADL.,DISP=SHR
//DBRMLIB  DD DSN=&DBRMLIB2.,DISP=SHR
//SYSTSPRT DD DSN=&&PARMLIST,DISP=(OLD,PASS)
//SYSUDUMP DD SYSOUT=&SYSOUT.
//*********************************************************************
//* BIND APPLICATION PLAN IF EXECUTING IN BACKGROUND                  *
//*********************************************************************
//BINDBG EXEC PGM=IKJEFT01,COND=((5,NE,BINDFG),(5,LT)),MAXRC=7
//STEPLIB  DD DSN=&DB2LOADL.,DISP=SHR
```

```
//DBRMLIB  DD DSN=&DBRMLIB2.,DISP=SHR
//SYSTSPRT DD DSN=&&PARMLIST,DISP=(OLD,PASS)
//SYSTSIN  DD DSN=&PARMLIB(&C1ELEMENT).,DISP=(OLD,PASS)
//*********************************************************************
//* FOOTPRINT DB2 PLAN                                                *
//*********************************************************************
//FOOTDB2 EXEC PGM=BC1PCAF,COND=(8,LE),MAXRC=0,
// PARM='&DB2SYS.,BC1PSQL1,BC1PDBFP'
//STEPLIB   DD DSN=&EDB2AUTH.,DISP=SHR
//          DD DSN=&EDB2CONL.,DISP=SHR
//DBRMLIB   DD DSN=&DBRMLIB2.,DISP=SHR
//BSTIPT    DD DSN=&PARMLIB(&C1ELEMENT).,DISP=(OLD,PASS)
//*********************************************************************
//*   COPY THE LISTINGS IF: &LISTING=LISTING. LIBRARY NAME          *
//*********************************************************************
//CONLIST EXEC PGM=CONLIST,MAXRC=0,PARM=COPY,COND=EVEN,
//     EXECIF=(&LISTLIB.,NE,NO)
//C1LLIBI  DD DSN=&LISTLIB1.,DISP=SHR
//C1LLIBO  DD DSN=&LISTLIB2.,DISP=SHR,MONITOR=&MONITOR.
//C1BANNER DD UNIT=&WRKUNIT.,SPACE=(TRK,(1,1)),
//             DCB=(RECFM=FBA,LRECL=121,BLKSIZE=6171)
//LIST01   DD DSN=&&COPY1LST,DISP=(OLD,DELETE)
//LIST02   DD DSN=&&COPY2LST,DISP=(OLD,DELETE)
//LIST03   DD DSN=&&PARMLIST,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)
//C1PRINT  DD SYSOUT=&SYSOUT.,
//            DCB=(RECFM=FBA,LRECL=133,BLKSIZE=1330,DSORG=PS)
//LIST01  DD DSN=&&COPY1LST,DISP=(OLD,DELETE)
//LIST02  DD DSN=&&COPY2LST,DISP=(OLD,DELETE)
//LIST03  DD DSN=&&PARMLIST,DISP=(OLD,DELETE)
//MOVECL  EXEC PGM=BC1PMVCL,MAXRC=04,COND=(4,LT)
//*
_
```

The MLODNNL Move Processor

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)
//*
```

Other Processors

These processors are provided as an example of how to manage other types of source. Use these processors as a guide. The following samples are provided in the iprfx.iqual.CSIQSAMP.

ALIASMOV

Moves alias load modules associated with the current element

BINDPLAN

Stand-alone Generate processor for bind cards

CICSMAP

Generate processor for CICS maps and copybooks

EASYTRIE

Generate processor for CA Easytrieve IQ Online Query

GCPYIMP

Ascertains a copybook's impact by executing a list action (using ACMQ) and providing a list of elements using the copybook

IDEAL01

Sample processor for CA Ideal for CA Datacom

IMSDBD

Sample IMS processor

JCLCHK

Sample processor for Unicenter CA JCLCheck

JOBSCAN

Sample JOBSCAN Processor

SDFIIMAP

Generate Processor for SDF II maps

CSP Processors

This appendix contains examples of Generate processors that use the LEXTRCTR and BC1PCCSP utilities.

- GELCL
- GELPCLB
- GELTCL
- GELPTCLB
- GCSP41LN (for CSP Extension Facility elements)

The GELCL CSP Processor

```
//*********************************************************************
//** GENERATE, COMPILE, LINK-EDIT CSP APPLS, TABLES AND MAP GROUPS    **
//*********************************************************************
//*
//GELCL  PROC CGHLQ='USER',
//     COBCOMP='SYS1.COB2COMP',
//     COBLIB='SYS1.COB2LIB',
//     CONLIB='iprfx.iqual.CSIQLOAD',
//     CRS='CRS210',
//     CSP='CSP410',
//     CSPVSAM='CSP410',
//     CSPCUST='CSP410',
//     CSPUSER='CSPUSER',
//     DATA='31',
//     ENV='MVSBATCH',
//     LISTLIB=NULLFILE,
//     LOADLIB=NULLFILE,
//     MAP='NONE',
//     NLS='ENU',
//     RESLIB='IMSVS.RESLIB',  (UNCOMMENT BELOW IF NECESSARY)
//     SOUT='*',
//     TABLE='NONE',
//     USR1MSL=NULLFILE,
//     USR2MSL=NULLFILE,
//     WSPC=500
//*
//* CSP PARMS: (PARM VALUES SHOULD BE SPECIFIED VIA PROC GROUPS)
//*
//*   CGHLQ   = COBOL GENERATION USER DATA SET HIGH-LEVEL QUALIFIER
//*   COBCOMP = COBOL COMPILER LIBRARY
//*   COBLIB  = COBOL RUN TIME LIBRARY
//*   CRS     = CSP/370RS HIGH LEVEL QUALIFIER
//*   CSP     = CSP/370AD HIGH LEVEL QUALIFIER
//*   CSPVSAM = HIGH LEVEL QUALIFIER FOR VSAM DATA SETS
//*   CSPCUST = HIGH LEVEL QUALIFIER FOR CUSTOMIZED LIBRARIES
//*   CSPUSER = CSP/370AD USER WORK DATA SET HIGH LEVEL QUALIFIER
//*   DATA    = COMPILE OPTION FOR PLACING WORKING STORAGE
//*             ABOVE 16M LINE
```

```
//*   ENV     = COBOL GENERATION USER DATA SET ENVIRONMENT QUALIFIER
//*             (SHOULD BE EQUAL TO GENERATION TARGET ENVIRONMENT)
//*   MAP     = NONE/ALL - FOR APPLICATION, GEN MAPS WITH APPL
//*   NLS     = NLS CODE (NATIONAL LANGUAGE DATA SET QUALIFIER)
//*   RESLIB  = IMS RESLIB LIBRARY
//*   SOUT    = SYSOUT ASSIGNMENT
//*   TABLE   = NONE/ALL - FOR APPLICATION, GEN TABLES WITH APPL
//*   WSPC    = PRIMARY AND SECONDARY SPACE ALLOCATION
//* Endevor PARMS: (PARM VALUES SHOULD BE SPECIFIED VIA PROC GROUPS)
//*   CONLIB  = Endevor RUN TIME LIBRARY
//*   LISTLIB = Endevor LISTINGS LIBRARY
//*   LOADLIB = Endevor/CSP OUTPUT LOAD LIBRARY
//*   USR1MSL = READ/WRITE MSL
//*   USR2MSL = READ-ONLY MSL
//*
//*********************************************************************
//*  INITIALIZE GENERATE/COMPILE/LINK-EDIT LISTING DATA SETS          *
//*********************************************************************
//*
//INITLIST EXEC PGM=BC1PDSIN,MAXRC=0
//C1INIT01 DD DSN=&&EZECOUT,DISP=(,PASS),
//            UNIT=tdisk,SPACE=(TRK,(5,5)),
//            DCB=(RECFM=FBA,LRECL=121,BLKSIZE=6171)
//C1INIT02 DD DSN=&&EZEPRINT,DISP=(,PASS),
//            UNIT=tdisk,SPACE=(TRK,(5,5)),
//            DCB=(RECFM=VBA,LRECL=654,BLKSIZE=6000)
//C1INIT03 DD DSN=&&COBPRINT,DISP=(,PASS),
//            UNIT=tdisk,SPACE=(TRK,(30,15)),
//            DCB=(RECFM=FBA,LRECL=133,BLKSIZE=6251)
//C1INIT04 DD DSN=&&CB2PRINT,DISP=(,PASS),
//            UNIT=tdisk,SPACE=(TRK,(5,5)),
//            DCB=(RECFM=FBA,LRECL=133,BLKSIZE=6251)
//C1INIT05 DD DSN=&&LNKPRINT,DISP=(,PASS),
//            UNIT=tdisk,SPACE=(TRK,(5,5)),
//            DCB=(RECFM=FBA,LRECL=121,BLKSIZE=3630)
//*
//COBGEN EXEC PGM=BC1PDCGB,COND=(0,NE),MAXRC=4
//*-----------------------
//*  LOADLIB AND MESSAGE FILES
//*-----------------------
//STEPLIB DD DISP=SHR,DSN=&CRS..SELALMD
//        DD DISP=SHR,DSN=&CSP..SEZELMD
//*       DD DISP=SHR,DSN=EDC.SEDCLINK
//*       DD DISP=SHR,DSN=PLI.SIBMLINK
//        DD DISP=SHR,DSN=&COBLIB.
//        DD DISP=SHR,DSN=&CONLIB.
//*
//EZECOMM DD DISP=SHR,DSN=&CSPVSAM..&NLS..EZECOMM
//*
```

```
//*-------------------------------------------------------
//*  THE FOLLOWING MESSAGE FILES SHOULD BE PRE-ALLOCATED
//*-------------------------------------------------------
//*EZEDMSG DD DISP=SHR,DSN=&CSPVSAM..FZEMSG
//*EZEEMSG DD DISP=SHR,DSN=&CSPVSAM..&NLS..EZEMSG
//*EZETES  DD DISP=SHR,DSN=&CSPVSAM..&NLS..FZETUTOR
//*EZEMAP  DD DISP=SHR,DSN=&CSPVSAM..&NLS..FZEMAPDS
//*----------------------
//*  WORK FILES
//*----------------------
//EZEWORK  DD DISP=OLD,DSN=&CSPUSER..EZEWORK
//EZEWRK1  DD UNIT=tdisk,SPACE=(CYL,(2,1)),
//            DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//*----------------------
//*  COMMAND AND LOG FILES
//*----------------------
//EZECOUT  DD DSN=&&EZECOUT,DISP=(OLD,PASS)
//*----------------------
//*  PRINT FILES
//*----------------------
//EZEPRINT DD DSN=&&EZEPRINT,DISP=(OLD,PASS)
//EZECPRT  DD SYSOUT=&SOUT.
//SYSPRINT DD SYSOUT=&SOUT.
//*----------------------
//*  COBOL GENERATION CONTROL FILES
//*----------------------
//EZEOPT   DD DISP=SHR,DSN=&CSPCUST..PDS.EZEOPT
//*EZEOPT  DD DISP=SHR,DSN=&CSPVSAM..VSAM.EZEOPT
//EZEJTMP  DD DISP=SHR,DSN=&CSPCUST..EZESAMP
//EZEWORD  DD DISP=SHR,DSN=&CSPCUST..EZESAMP(EZEWORDS)
//EZESNAP  DD SYSOUT=&SOUT.
//*----------------------
//*  INTERNAL READER FILE FOR PREPARATION JCL SUBMISSION
//*----------------------
//EZEJCL  DD DSN=&&EZEJCL,DISP=(,PASS),
//           UNIT=tdisk,SPACE=(TRK,(1,1)),
//           DCB=(RECFM=FB,LRECL=80,BLKSIZE=6160)
//*----------------------
//*----------------------
//*  ENVIRONMENT DEPENDENT CONTROL AND OUTPUT FILES
//*
//*  THE FOLLOWING FILES MAY OR MAY NOT BE REQUIRED, DEPENDING
//*  ON FUNCTIONS USED AT SPECIFIC INSTALLATIONS. THE DD CARDS
//*  ARE SHOWN AS COMMENTS IN THE PROCEDURE. THE GENERATOR USES
//*  DYNAMIC ALLOCATION TO ALLOCATE EXISTING DATA SETS WITH THE
//*  NAMES AS SHOWN IN THE DD CARDS, WHERE &CGHLQ. IS THE DATA SET
//*  QUALIFIER FROM THE USERID COBOL GENERATION OPTION AND &ENV.
//*  IS THE TARGET EXECUTION ENVIRONMENT.
//*
```

```
//*  TO USE DATA SETS WITH OTHER NAMES, MODIFY THE PROCEDURE OR THE
//*  JCL TO ALLOCATE THE DATA SETS WITH THE APPROPRIATE DD NAMES
//*  AS SHOWN BELOW. THE GENERATOR CHECKS TO SEE IF A DATA SET
//*  HAS ALREADY BEEN ALLOCATED TO THE DD NAME BEFORE ATTEMPTING TO
//*  DYNAMICALLY ALLOCATE A DATA SET WITH THE STANDARD NAME.
//*
//*----------------------
//*  COBOL SOURCE FILE
//*----------------------
//EZESRC  DD DISP=(NEW,PASS),DSN=&&EZESRC,
//           UNIT=tdisk,SPACE=(CYL,(5,5,10),RLSE),
//           DCB=(RECFM=FB,LRECL=80,BLKSIZE=6160)
//*
//EZEDUMMY DD DUMMY,DISP=SHR,DSN=&CGHLQ..&ENV..EZEDUMMY
//*----------------------
//*----------------------
//*  USER DEFINED LINK EDIT CONTROL STATEMENTS
//*----------------------
//EZELINK DD DUMMY
//*----------------------
//*  USER DEFINED BIND CONTROL STATEMENTS
//*----------------------
//EZEBIND DD DUMMY
//*-------------------------
//*  IMS COBOL COPY LIBRARY
//*-------------------------
//*EZECOPY DD DISP=SHR,DSM=&CGHLQ..&ENV..EZECOPY
//*----------------------
//*  OBJECT LIBRARIES FOR GENERATED MAP GROUPS
//*----------------------
//EZEFOBJ DD DISP=(NEW,PASS),DSN=&&EZEFOBJ,
//           UNIT=tdisk,SPACE=(TRK,(5,5,1),RLSE),
//           DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120)
//*EZEPOBJ DD DISP=SHR,DSN=&CGHLQ..&ENV..EZEPOBJ
//*----------------------
//*  TSO CLIST LIBRARY
//*----------------------
//EZECLST DD DISP=SHR,DSN=NULLFILE
//*----------------------
//*  PARTS LIST LIBRARY
//*----------------------
//EZEPCTL DD DISP=SHR,DSN=NULLFILE
//*----------------------
//*  MFS SOURCE LIBRARY
//*----------------------
//*EZEMFS  DD DISP=SHR,DSN=&CGHLQ..&ENV..EZEMFS
//*----------------------
//*  JCL LIBRARIES
//*----------------------
```

```
//*EZEJCLX DD DISP=SHR,DSN=&CGHLQ..&ENV..EZEJCLX
//*EZEJCLP DD DISP=SHR,DSN=&CGHLQ..&ENV..EZEJCLP
//USR1MSL  DD DISP=SHR,DSN=&USR1MSL.
//USR2MSL  DD DISP=SHR,DSN=&USR2MSL.
//NDVRIPT  DD *
 M=USR1MSL ROMSL=USR2MSL CMDIN=EZECIN CMDOUT=EZECOUT
//EZECIN  DD *
 GENERATE MEMBER(&C1ELEMENT). SYSTEM(&ENV).
      TABLES(&TABLE). MAPS(&MAP). BATCH(NO) ;
//EZEPARM  DD UNIT=tdisk,SPACE=(TRK,1),
//            DCB=(RECFM=FB,LRECL=80,BLKSIZE=80)
//*
//*********************************************************************
//*      EXTRACT THE LINKAGE-EDITOR CONTROL STATEMENTS FROM
//*      THE SEQUENTIAL FILE CREATED BY THE GENERATE STEP
//*********************************************************************
//*
//EXTRLNK EXEC PGM=LEXTRCTR,COND=(5,LT),MAXRC=0
//STEPLIB DD DISP=SHR,DSN=&CONLIB.
//INPUT   DD DSN=&&EZEJCL,DISP=(OLD,PASS)
//OUTPUT  DD DSN=&&CONTROL,DISP=(,PASS),
//           UNIT=tdisk,SPACE=(TRK,(1,1)),
//           DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120)
//*
//*********************************************************************
//** ELACL - COBOL COMPILE AND LINK-EDIT
//*********************************************************************
//*
//*
//*********************************************************************
//*      COMPILE THE COBOL PROGRAM
//*********************************************************************
//*
//C        EXEC PGM=IGYCRCTL,COND=(4,LT),MAXRC=4,
//         PARM=(NOSEQ,QUOTE,OFFSET,LIB,RES,RENT,NODYNAM,DBCS,OPT,
//         'TRUNC(BIN)','NUMPROC(NOPFD)',NOCMPR2,'DATA(&DATA)').
//STEPLIB  DD DISP=SHR,DSN=&COBCOMP.
//SYSIN    DD DISP=(OLD,PASS),DSN=&&EZESRC(&C1ELEMENT)
//SYSLIB   DD DISP=SHR,DSN=&CRS..SELACOPY,MONITOR=COMPONENTS
//SYSLIN   DD DISP=(NEW,PASS),DSN=&&LOADSET(&C1ELEMENT),
//            UNIT=tdisk,SPACE=(800,(&WSPC.,&WSPC.,10)),
//            FOOTPRNT=CREATE,
//            DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120)
//SYSPRINT DD DSN=&&COBPRINT,DISP=(OLD,PASS)
//SYSUT1   DD SPACE=(800,(&WSPC.,&WSPC).,,,ROUND),UNIT=tdisk
//SYSUT2   DD SPACE=(800,(&WSPC.,&WSPC).,,,ROUND),UNIT=tdisk
//SYSUT3   DD SPACE=(800,(&WSPC.,&WSPC).,,,ROUND),UNIT=tdisk
//SYSUT4   DD SPACE=(800,(&WSPC.,&WSPC).,,,ROUND),UNIT=tdisk
//SYSUT5   DD SPACE=(800,(&WSPC.,&WSPC).,,,ROUND),UNIT=tdisk
```

```
//SYSUT6   DD SPACE=(800,(&WSPC.,&WSPC).,,,ROUND),UNIT=tdisk
//SYSUT7   DD SPACE=(800,(&WSPC.,&WSPC).,,,ROUND),UNIT=tdisk
//*
//*********************************************************************
//*      COMPILE ANY ADDITIONAL MAPS AND TABLES                       *
//*********************************************************************
//*
//C2       EXEC PGM=BC1PCCSP,COND=(4,LT),MAXRC=4,
//         PARM=(NOSEQ,QUOTE,OFFSET,LIB,RES,RENT,NODYNAM,DBCS,OPT,
//         'TRUNC(BIN)','NUMPROC(NOPFD)',NOCMPR2,'DATA(&DATA)').
//COBCOMP  DD DISP=SHR,DSN=&COBCOMP.
//SYSLIB   DD DISP=SHR,DSN=&CRS..SELACOPY,MONITOR=COMPONENTS
//SYSIN    DD UNIT=tdisk,SPACE=(6400,(&WSPC.,&WSPC)).,
//            DCB=(RECFM=FB,LRECL=80,BLKSIZE=6400)
//SYSLIN   DD UNIT=tdisk,SPACE=(3120,(&WSPC.,&WSPC)).,
//            DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120)
//SYSPRINT DD UNIT=tdisk,SPACE=(3630,(&WSPC.,&WSPC)).,
//            DCB=(RECFM=FBA,LRECL=133,BLKSIZE=6251)
//GENCNTL  DD DISP=(OLD,PASS),DSN=&&CONTROL
//EZESRC   DD DISP=(OLD,PASS),DSN=&&EZESRC
//OBJLIB   DD DISP=(OLD,PASS),DSN=&&LOADSET
//PRINTLIB DD DISP=(OLD,PASS),DSN=&&CB2PRINT
//GENINFO  DD *
ELEMENT=&C1ELEMENT.       KEEP THESE CARDS IN ORDER,
 COMPILER=IGYCRCTL       LEFT JUST WITH NO BLANKS
//SYSUT1  DD SPACE=(800,(&WSPC.,&WSPC).,,,ROUND),UNIT=tdisk
//SYSUT2  DD SPACE=(800,(&WSPC.,&WSPC).,,,ROUND),UNIT=tdisk
//SYSUT3  DD SPACE=(800,(&WSPC.,&WSPC).,,,ROUND),UNIT=tdisk
//SYSUT4  DD SPACE=(800,(&WSPC.,&WSPC).,,,ROUND),UNIT=tdisk
//SYSUT5  DD SPACE=(800,(&WSPC.,&WSPC).,,,ROUND),UNIT=tdisk
//SYSUT6  DD SPACE=(800,(&WSPC.,&WSPC).,,,ROUND),UNIT=tdisk
//SYSUT7  DD SPACE=(800,(&WSPC.,&WSPC).,,,ROUND),UNIT=tdisk
//*
//*********************************************************************
//*      LINK-EDIT THE COBOL PROGRAM AND ANY MAPS/TABLES              *
//*      IF THE RETURN CODE ON ALL PREVIOUS STEPS IS 4 OR LESS        *
//*********************************************************************
//*
//L        EXEC PGM=IEWL,COND=(5,LT),MAXRC=0,
//         PARM='RENT,REUS,LIST,XREF,MAP,AMODE(31),RMODE(ANY)'
//SYSLIB   DD DISP=SHR,DSN=&COBLIB.,MONITOR=COMPONENTS
//*        DD DISP=SHR,DSN=&RESLIB.,MONITOR=COMPONENTS
//OBJLIB   DD DISP=(OLD,DELETE),DSN=&&LOADSET,MONITOR=COMPONENTS
//         DD DISP=(OLD,DELETE),DSN=&&EZEFOBJ,MONITOR=COMPONENTS
//SELALMD  DD DISP=SHR,DSN=&CRS..SELALMD,MONITOR=COMPONENTS
//SYSLMOD  DD DISP=SHR,DSN=&LOADLIB.,
//            FOOTPRNT=CREATE,MONITOR=COMPONENTS
//SYSPRINT DD DSN=&&LNKPRINT,DISP=(OLD,PASS)
//SYSUT1   DD SPACE=(1024,(&WSPC.,&WSPC)).,UNIT=tdisk
```

```
//SYSLIN   DD DSN=&&CONTROL,DISP=(OLD,DELETE)
//*
//*********************************************************************
//*  STORE OUTPUT LISTINGS IN LIST LIBRARY                            *
//*********************************************************************
//CONLIST  EXEC PGM=CONLIST,PARM=STORE,MAXRC=0,COND=EVEN
//C1LLIBO  DD DSN=&LISTLIB.,DISP=SHR,MONITOR=COMPONENTS
//C1BANNER DD UNIT=tdisk,SPACE=(TRK,1),
//            DCB=(RECFM=FBA,LRECL=121)
//LIST01   DD DSN=&&EZECOUT,DISP=(OLD,DELETE)
//LIST02   DD DSN=&&EZEPRINT,DISP=(OLD,DELETE)
//LIST03   DD DSN=&&EZEJCL,DISP=(OLD,DELETE)
//LIST04   DD DSN=&&COBPRINT,DISP=(OLD,DELETE)
//LIST05   DD DSN=&&CB2PRINT,DISP=(OLD,DELETE)
//LIST06   DD DSN=&&LNKPRINT,DISP=(OLD,DELETE)
//*
```

The GELPCLB CSP Processor

```
//*********************************************************************
//** GENERATE, DB2 PRE-PROCESS, COMPILE, LINK-EDIT AND BIND          **
//**  CSP APPLICATIONS                                              **
//*********************************************************************
//*
//GELPCLB PROC CGHLQ='USER',
//         BINDDSN='&C1USERID..NDVRCSP.BINDDSN',
//         COBCOMP='SYS1.COB2COMP',
//         COBLIB='SYS1.COB2LIB',
//         CONLIB='iprfx.iqual.CSIQLOAD',
//         CRS='CRS210',
//         CSP='CSP410',
//         CSPVSAM='CSP410',
//         CSPCUST='CSP410',
//         CSPUSER='CSPUSER',
//         DATA='31',
//         DBRMLIB='NULLFILE',
//         DSNEXIT='DSN220.DSNEXIT',
//         DSNLOAD='DSN220.DSNLOAD',
//         ENV='MVSBATCH',
//         LISTLIB='NULLFILE',
//         LOADLIB='NULLFILE',
//         MAP='NONE',
//         NLS='ENU',
//         RESLIB='IMSVS.RESLIB',  (UNCOMMENT BELOW IF NECESSARY)
//         SOUT='*',
//         TABLE='NONE',
//         USR1MSL='NULLFILE',
//         USR2MSL='NULLFILE',
//         WSPC=500
//*
//*
//* CSP PARMS: (PARM VALUES SHOULD BE SPECIFIED VIA PROC GROUPS)
//*
//*    CGHLQ   = COBOL GENERATION USER DATA SET HIGH-LEVEL QUALIFIER
//*    COBCOMP = COBOL COMPILER LIBRARY
//*    COBLIB  = COBOL RUN TIME LIBRARY
//*    CRS     = CSP/370RS HIGH LEVEL QUALIFIER
//*    CSP     = CSP/370AD HIGH LEVEL QUALIFIER
//*    CSPVSAM = HIGH LEVEL QUALIFIER FOR VSAM DATA SETS
//*    CSPCUST = HIGH LEVEL QUALIFIER FOR CUSTOMIZED LIBRARIES
```

```
//*    CSPUSER = CSP/370AD USER WORK DATA SET HIGH LEVEL QUALIFIER
//*    DATA    = COMPILE OPTION FOR PLACING WORKING STORAGE
//*         ABOVE 16M LINE
//*    DSNEXIT = DB2 DSNEXIT LIBRARY
//*    DSNLOAD = DB2 DSNLOAD LIBRARY
//*    ENV     = COBOL GENERATION USER DATA SET ENVIRONMENT QUALIFIER
//*              (SHOULD BE EQUAL TO GENERATION TARGET ENVIRONMENT)
//*    MAP     = NONE/ALL - FOR APPLICATION, GEN MAPS WITH APPL
//*    NLS     = NLS CODE (NATIONAL LANGUAGE DATA SET QUALIFIER)
//*    RESLIB  = IMS RESLIB LIBRARY
//*   SOUT     = SYSOUT ASSIGNMENT
//*   TABLE    = NONE/ALL - FOR APPLICATION, GEN TABLES WITH APPL
//*   WSPC     = PRIMARY AND SECONDARY SPACE ALLOCATION
//*
//* Endevor PARMS: (PARM VALUES SHOULD BE SPECIFIED VIA PROC GROUPS)
//*
//*   BINDDSN  = 'PERMANENTLY-ALLOCATED' TEMPORARY BIND CNTL CARD LIB
//*   CONLIB   = Endevor RUN TIME LIBRARY
//*   DBRMLIB  = DB2 DBRMLIB LIBRARY
//*   LISTLIB  = Endevor/CSP LISTINGS LIBRARY
//*   LOADLIB  = Endevor/CSP OUTPUT LOAD LIBRARY
//*   USR1MSL  = READ/WRITE MSL
//*   USR2MSL  = READ-ONLY MSL
//*
//*********************************************************************
//*  INITIALIZE GENERATE/COMPILE/LINK-EDIT LISTING DATA SETS          *
//*********************************************************************
//*
//INITLIST EXEC PGM=BC1PDSIN,MAXRC=0
//C1INIT01 DD DSN=&&EZECOUT,DISP=(,PASS),
//            UNIT=tdisk,SPACE=(TRK,(5,5)),
//            DCB=(RECFM=FBA,LRECL=121,BLKSIZE=6171)
//C1INIT02 DD DSN=&&EZEPRINT,DISP=(,PASS),
//            UNIT=tdisk,SPACE=(TRK,(5,5)),
//            DCB=(RECFM=VBA,LRECL=654,BLKSIZE=6000)
//C1INIT03 DD DSN=&&PRPPRINT,DISP=(,PASS),
//            UNIT=tdisk,SPACE=(TRK,(15,5)),
//            DCB=(RECFM=FBA,LRECL=133,BLKSIZE=6251)
//C1INIT04 DD DSN=&&COBPRINT,DISP=(,PASS),
//            UNIT=tdisk,SPACE=(TRK,(15,5)),
//            DCB=(RECFM=FBA,LRECL=133,BLKSIZE=6251)
//C1INIT05 DD DSN=&&CB2PRINT,DISP=(,PASS),
//            UNIT=tdisk,SPACE=(TRK,(15,5)),
//            DCB=(RECFM=FBA,LRECL=133,BLKSIZE=6251)
//C1INIT06 DD DSN=&&LNKPRINT,DISP=(,PASS),
//            UNIT=tdisk,SPACE=(TRK,(5,5)),
//            DCB=(RECFM=FBA,LRECL=121,BLKSIZE=3630)
//C1INIT07 DD DSN=&&BNDPRINT,DISP=(,PASS),
//            UNIT=tdisk,SPACE=(TRK,(5,2)),
```

```
//            DCB=(RECFM=FBA,LRECL=121,BLKSIZE=6171)
//*********************************************************************
//* INVOKE IDCAMS TO ENSURE THE BIND CNTL CARD O/P DATA SET IS DELETED*
//*********************************************************************
//IDCAMS   EXEC PGM=IDCAMS
//SYSPRINT DD DUMMY
//SYSIN    DD *
 DELETE '&BINDDSN'.
 SET MAXCC = 0
//COBGEN EXEC PGM=BC1PDCGB,COND=(0,NE),MAXRC=4
//*----------------------
//*  LOADLIB AND MESSAGE FILES
//*----------------------
//STEPLIB  DD DISP=SHR,DSN=&CRS..SELALMD
//         DD DISP=SHR,DSN=&CSP..SEZELMD
//*        DD DISP=SHR,DSN=EDC.SEDCLINK
//*        DD DISP=SHR,DSN=PLI.SIBMLINK
//         DD DISP=SHR,DSN=&COBLIB.
//         DD DISP=SHR,DSN=&CONLIB.
//*
//EZECOMM  DD DISP=SHR,DSN=&CSPVSAM..&NLS..EZECOMM
//*
//*-------------------------------------------------------
//*  THE FOLLOWING MESSAGE FILES SHOULD BE PRE-ALLOCATED
//*-------------------------------------------------------
//*EZEDMSG DD DISP=SHR,DSN=&CSPVSAM..FZEMSG
//*EZEEMSG DD DISP=SHR,DSN=&CSPVSAM..&NLS..EZEMSG
//*EZETES  DD DISP=SHR,DSN=&CSPVSAM..&NLS..FZETUTOR
//*EZEMAP  DD DISP=SHR,DSN=&CSPVSAM..&NLS..FZEMAPDS
//*----------------------
//*  WORK FILES
//*----------------------
//EZEWORK DD DISP=OLD,DSN=&CSPUSER..EZEWORK
//EZEWRK1 DD UNIT=tdisk,SPACE=(CYL,(2,1)),
//           DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//*----------------------
//*  COMMAND AND LOG FILES
//*----------------------
//EZECOUT DD DSN=&&EZECOUT,DISP=(OLD,PASS)
//*----------------------
//*  PRINT FILES
//*----------------------
//EZEPRINT DD DSN=&&EZEPRINT,DISP=(OLD,PASS)
//EZECPRT  DD SYSOUT=&SOUT.
//SYSPRINT DD SYSOUT=&SOUT.
//*----------------------
//*  COBOL GENERATION CONTROL FILES
//*----------------------
//EZEOPT   DD DISP=SHR,DSN=&CSPCUST..PDS.EZEOPT
```

```
//*EZEOPT  DD DISP=SHR,DSN=&CSPVSAM..VSAM.EZEOPT
//EZEJTMP  DD DISP=SHR,DSN=&CSPCUST..EZESAMP
//EZEWORD  DD DISP=SHR,DSN=&CSPCUST..EZESAMP(EZEWORDS)
//EZESNAP  DD SYSOUT=&SOUT.
//*----------------------
//*  INTERNAL READER FILE FOR PREPARATION JCL SUBMISSION
//*----------------------
//EZEJCL  DD DSN=&&EZEJCL,DISP=(,PASS),
//           UNIT=tdisk,SPACE=(TRK,(1,1)),
//           DCB=(RECFM=FB,LRECL=80,BLKSIZE=6160)
//*EZEJCL DD SYSOUT=(,INTRDR)
//*----------------------
//*----------------------
//*  ENVIRONMENT DEPENDENT CONTROL AND OUTPUT FILES
//*
//*  THE FOLLOWING FILES MAY OR MAY NOT BE REQUIRED, DEPENDING
//*  ON FUNCTIONS USED AT SPECIFIC INSTALLATIONS. THE DD CARDS
//*  ARE SHOWN AS COMMENTS IN THE PROCEDURE. THE GENERATOR USES
//*  DYNAMIC ALLOCATION TO ALLOCATE EXISTING DATA SETS WITH THE
//*  NAMES AS SHOWN IN THE DD CARDS, WHERE &CGHLQ. IS THE DATA SET
//*  QUALIFIER FROM THE USERID COBOL GENERATION OPTION AND &ENV.
//*  IS THE TARGET EXECUTION ENVIRONMENT.
//*
//*  TO USE DATA SETS WITH OTHER NAMES, MODIFY THE PROCEDURE OR THE
//*  JCL TO ALLOCATE THE DATA SETS WITH THE APPROPRIATE DD NAMES
//*  AS SHOWN BELOW. THE GENERATOR CHECKS TO SEE IF A DATA SET
//*  HAS ALREADY BEEN ALLOCATED TO THE DD NAME BEFORE ATTEMPTING TO
//*  DYNAMICALLY ALLOCATE A DATA SET WITH THE STANDARD NAME.
//*
//*----------------------
//*  COBOL SOURCE FILE
//*----------------------
//EZESRC  DD DISP=(NEW,PASS),DSN=&&EZESRC,
//           UNIT=tdisk,SPACE=(CYL,(5,5,10),RLSE),
//           DCB=(RECFM=FB,LRECL=80,BLKSIZE=6160)
//*
//EZEDUMMY DD DUMMY,DISP=SHR,DSN=&CGHLQ..&ENV..EZEDUMMY
//*----------------------
//*----------------------
//*  USER DEFINED LINK EDIT CONTROL STATEMENTS
//*----------------------
//EZELINK DD DUMMY
//*----------------------
//*  USER-DEFINED BIND CONTROL STATEMENTS
//*----------------------
//EZEBIND DD DSN=&BINDDSN.,DISP=(,CATLG,DELETE),
//           UNIT=tdisk,SPACE=(TRK,(1,1,1)),
//           DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120)
//*-------------------------
```

```
//*  IMS COBOL COPY LIBRARY
//*-------------------------
//*EZECOPY DD DISP=SHR,DSM=&CGHLQ..&ENV..EZECOPY
//*----------------------
//*  OBJECT LIBRARIES FOR GENERATED MAP GROUPS
//*----------------------
//EZEFOBJ DD DISP=(NEW,PASS),DSN=&&EZEFOBJ,
//           UNIT=tdisk,SPACE=(TRK,(5,5,1),RLSE),
//           DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120)
//*EZEPOBJ DD DISP=SHR,DSN=&CGHLQ..&ENV..EZEPOBJ
//*----------------------
//*  TSO CLIST LIBRARY
//*----------------------
//EZECLST DD DISP=SHR,DSN=NULLFILE
//*----------------------
//*  PARTS LIST LIBRARY
//*----------------------
//EZEPCTL DD DISP=SHR,DSN=NULLFILE
//*----------------------
//*  MFS SOURCE LIBRARY
//*----------------------
//*EZEMFS  DD DISP=SHR,DSN=&CGHLQ..&ENV..EZEMFS
//*----------------------
//*  JCL LIBRARIES
//*----------------------
//*EZEJCLX DD DISP=SHR,DSN=&CGHLQ..&ENV..EZEJCLX
//*EZEJCLP DD DISP=SHR,DSN=&CGHLQ..&ENV..EZEJCLP
//USR1MSL  DD DISP=SHR,DSN=&USR1MSL.
//USR2MSL  DD DISP=SHR,DSN=&USR2MSL.
//NDVRIPT  DD *
 M=USR1MSL ROMSL=USR2MSL CMDIN=EZECIN CMDOUT=EZECOUT
//EZECIN  DD *
 GENERATE MEMBER(&C1ELEMENT). SYSTEM(&ENV).
      TABLES(&TABLE). MAPS(&MAP). BATCH(NO) ;
//EZEPARM  DD UNIT=tdisk,SPACE=(TRK,1),
//            DCB=(RECFM=FB,LRECL=80,BLKSIZE=80)
//*
//*********************************************************************
//*      EXTRACT THE LINKAGE-EDITOR CONTROL STATEMENTS FROM           **
//*      THE SEQUENTIAL FILE CREATED BY THE GENERATE STEP             **
//*********************************************************************
//*
//EXTRLNK EXEC PGM=LEXTRCTR,COND=(5,LT),MAXRC=0
//STEPLIB DD  DISP=SHR,DSN=&CONLIB.
//INPUT   DD  DSN=&&EZEJCL,DISP=(OLD,PASS)
//OUTPUT  DD  DSN=&&CONTROL,DISP=(,PASS),
//            UNIT=tdisk,SPACE=(TRK,(1,1)),
//            DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120)
//*********************************************************************
```

```
//** ELAPCLB - COBOL PRECOMPILE/COMPILE AND LINK-EDIT                **
//*********************************************************************
//*
//*****************************************************************
//*      PRECOMPILE THE COBOL PROGRAM                              **
//*****************************************************************
//*
//P        EXEC PGM=DSNHPC,MAXRC=4,
//         PARM=('HOST(COB2),APOSTSQL,QUOTE')
//STEPLIB  DD DISP=SHR,DSN=&DSNEXIT.
//         DD DISP=SHR,DSN=&DSNLOAD.
//DBRMLIB  DD DISP=OLD,DSN=&DBRMLIB.(&C1ELEMENT).
//SYSCIN   DD DISP=(NEW,PASS),DSN=&&DSNHOUT,UNIT=tdisk,
//            SPACE=(800,(&WSPC.,&WSPC)).
//SYSLIB   DD DISP=SHR,DSN=&CRS..SELACOPY,MONITOR=COMPONENTS
//SYSPRINT DD DSN=&&PRPPRINT,DISP=(OLD,PASS)
//SYSUT1   DD SPACE=(800,(&WSPC.,&WSPC).,,,ROUND),UNIT=tdisk
//SYSUT2   DD SPACE=(800,(&WSPC.,&WSPC).,,,ROUND),UNIT=tdisk
//SYSIN    DD DISP=(OLD,PASS),DSN=&&EZESRC(&C1ELEMENT)
//*
//*
//*********************************************************************
//*      COMPILE THE COBOL PROGRAM                                   **
//*********************************************************************
//*
//C        EXEC PGM=IGYCRCTL,COND=(4,LT),MAXRC=4,
//         PARM=(NOSEQ,QUOTE,OFFSET,LIB,RES,RENT,NODYNAM,DBCS,OPT,
//         'TRUNC(BIN)','NUMPROC(NOPFD)',NOCMPR2,'DATA(&DATA)').
//STEPLIB  DD DISP=SHR,DSN=&COBCOMP.
//SYSIN    DD DISP=(OLD,DELETE),DSN=&&DSNHOUT
//SYSLIB   DD DISP=SHR,DSN=&CRS..SELACOPY,MONITOR=COMPONENTS
//SYSLIN   DD DISP=(NEW,PASS),DSN=&&LOADSET(&C1ELEMENT),UNIT=tdisk,
//            SPACE=(800,(&WSPC.,&WSPC.,10)),FOOTPRNT=CREATE,
//            DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120)
//SYSPRINT DD DSN=&&COBPRINT,DISP=(OLD,PASS)
//SYSUT1   DD SPACE=(800,(&WSPC.,&WSPC).,,,ROUND),UNIT=tdisk
//SYSUT2   DD SPACE=(800,(&WSPC.,&WSPC).,,,ROUND),UNIT=tdisk
//SYSUT3   DD SPACE=(800,(&WSPC.,&WSPC).,,,ROUND),UNIT=tdisk
//SYSUT4   DD SPACE=(800,(&WSPC.,&WSPC).,,,ROUND),UNIT=tdisk
//SYSUT5   DD SPACE=(800,(&WSPC.,&WSPC).,,,ROUND),UNIT=tdisk
//SYSUT6   DD SPACE=(800,(&WSPC.,&WSPC).,,,ROUND),UNIT=tdisk
//SYSUT7   DD SPACE=(800,(&WSPC.,&WSPC).,,,ROUND),UNIT=tdisk
//*
//*********************************************************************
//*      COMPILE ANY ADDITIONAL MAPS AND TABLES                      **
//*********************************************************************
//*
//C2       EXEC PGM=BC1PCCSP,COND=(4,LT),MAXRC=4,
//         PARM=(NOSEQ,QUOTE,OFFSET,LIB,RES,RENT,NODYNAM,DBCS,OPT,
```

```
//        'TRUNC(BIN)','NUMPROC(NOPFD)',NOCMPR2,'DATA(&DATA)').
//COBCOMP  DD DISP=SHR,DSN=&COBCOMP.
//SYSLIB   DD DISP=SHR,DSN=&CRS..SELACOPY,MONITOR=COMPONENTS
//SYSIN    DD UNIT=tdisk,SPACE=(6400,(&WSPC.,&WSPC)).,
//            DCB=(RECFM=FB,LRECL=80,BLKSIZE=6400)
//SYSLIN   DD UNIT=tdisk,SPACE=(3120,(&WSPC.,&WSPC)).,
//            DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120)
//SYSPRINT DD UNIT=tdisk,SPACE=(3630,(&WSPC.,&WSPC)).,
//            DCB=(RECFM=FBA,LRECL=133,BLKSIZE=6251)
//GENCNTL  DD DISP=(OLD,PASS),DSN=&&CONTROL
//EZESRC   DD DISP=(OLD,PASS),DSN=&&EZESRC
//OBJLIB   DD DISP=(OLD,PASS),DSN=&&LOADSET
//PRINTLIB DD DISP=(OLD,PASS),DSN=&&CB2PRINT
//GENINFO DD *
ELEMENT=&C1ELEMENT.       KEEP THESE CARDS IN ORDER,
 COMPILER=IGYCRCTL       LEFT JUST WITH NO BLANKS
//SYSUT1  DD SPACE=(800,(&WSPC.,&WSPC).,,,ROUND),UNIT=tdisk
//SYSUT2  DD SPACE=(800,(&WSPC.,&WSPC).,,,ROUND),UNIT=tdisk
//SYSUT3  DD SPACE=(800,(&WSPC.,&WSPC).,,,ROUND),UNIT=tdisk
//SYSUT4  DD SPACE=(800,(&WSPC.,&WSPC).,,,ROUND),UNIT=tdisk
//SYSUT5  DD SPACE=(800,(&WSPC.,&WSPC).,,,ROUND),UNIT=tdisk
//SYSUT6  DD SPACE=(800,(&WSPC.,&WSPC).,,,ROUND),UNIT=tdisk
//SYSUT7  DD SPACE=(800,(&WSPC.,&WSPC).,,,ROUND),UNIT=tdisk
//*
//*********************************************************************
//*      LINK-EDIT THE COBOL PROGRAM                                  **
//*      IF THE RETURN CODE ON ALL PREVIOUS STEPS IS 4 OR LESS        **
//*********************************************************************
//*
//L        EXEC PGM=IEWL,COND=(5,LT),MAXRC=0,
//         PARM='RENT,REUS,LIST,XREF,MAP,AMODE(31),RMODE(ANY)'
//SYSLIB   DD DISP=SHR,DSN=&COBLIB.,MONITOR=COMPONENTS
//*        DD DISP=SHR,DSN=&RESLIB.,MONITOR=COMPONENTS
//         DD DISP=SHR,DSN=&DSNLOAD.,MONITOR=COMPONENTS
//OBJLIB   DD DISP=(OLD,DELETE),DSN=&&LOADSET,MONITOR=COMPONENTS
//         DD DISP=(OLD,DELETE),DSN=&&EZEFOBJ,MONITOR=COMPONENTS
//SELALMD  DD DISP=SHR,DSN=&CRS..SELALMD,MONITOR=COMPONENTS
//SYSLMOD  DD DISP=SHR,DSN=&LOADLIB.,
//            FOOTPRNT=CREATE,MONITOR=COMPONENTS
//SYSPRINT DD DSN=&&LNKPRINT,DISP=(OLD,PASS)
//SYSUT1   DD SPACE=(1024,(&WSPC.,&WSPC)).,UNIT=tdisk
//SYSLIN   DD DSN=&&CONTROL,DISP=(OLD,DELETE)
//*
//*********************************************************************
//* BIND APPLICATION PLAN IN FOREGROUND                               **
//* NOTE: ATTEMPTING TO RUN THIS STEP IN BG WILL RESULT IN RC=5       **
//*********************************************************************
//BINDFG   EXEC PGM=BC1PTMP0,COND=(8,LT),MAXRC=5,
//         PARM='&BINDDSN(&C1ELEMENT)'.
```

```
//STEPLIB  DD DISP=SHR,DSN=&DSNLOAD.
//DBRMLIB  DD DISP=SHR,DSN=&DBRMLIB.,MONITOR=COMPONENTS
//         DD DISP=SHR,DSN=&CRS..SELADBRM,MONITOR=COMPONENTS
//SYSTSPRT DD DISP=(OLD,PASS),DSN=&&BNDPRINT
//*********************************************************************
//* BIND APPLICATION PLAN IN BACKGROUND                               *
//*********************************************************************
//BINDBG   EXEC PGM=IKJEFT01,COND=((8,LT),(5,NE,BINDFG)),MAXRC=0
//STEPLIB  DD DISP=SHR,DSN=&DSNLOAD.
//SYSTSIN  DD DISP=SHR,DSN=&BINDDSN.(&C1ELEMENT).
//DBRMLIB  DD DISP=SHR,DSN=&DBRMLIB.,MONITOR=COMPONENTS
//         DD DISP=SHR,DSN=&CRS..SELADBRM,MONITOR=COMPONENTS
//SYSTSPRT DD DISP=(OLD,PASS),DSN=&&BNDPRINT
//*********************************************************************
//* FOOTPRINT DB2 PLAN                                                *
//*********************************************************************
//FOOTDB2 EXEC PGM=BC1PCAF,COND=(8,LT),MAXRC=0,
//        PARM='DB2T,BC1PSQL1,BC1PDBFP'
//STEPLIB DD DISP=SHR,DSN=&DSNLOAD.
//DBRMLIB DD DISP=SHR,DSN=&DBRMLIB.
//        DD DISP=SHR,DSN=&CRS..SELADBRM
//BSTIPT  DD DISP=SHR,DSN=&BINDDSN.(&C1ELEMENT).
//*********************************************************************
//* INVOKE IDCAMS TO ENSURE THE BIND CNTL CARD O/P DATA SET IS DELETED*
//*********************************************************************
//IDCAMS   EXEC PGM=IDCAMS
//SYSPRINT DD DUMMY
//SYSIN    DD *
 DELETE '&BINDDSN'.
 SET MAXCC = 0
//*********************************************************************
//*  STORE OUTPUT LISTINGS IN LIST LIBRARY                            *
//*********************************************************************
//CONLIST  EXEC PGM=CONLIST,PARM=STORE,MAXRC=0,COND=EVEN
//C1LLIBO  DD DSN=&LISTLIB.,DISP=SHR,MONITOR=COMPONENTS
//C1BANNER DD UNIT=tdisk,SPACE=(TRK,1),
//            DCB=(RECFM=FBA,LRECL=121)
//LIST01   DD DSN=&&EZECOUT,DISP=(OLD,DELETE)
//LIST02   DD DSN=&&EZEPRINT,DISP=(OLD,DELETE)
//LIST03   DD DSN=&&EZEJCL,DISP=(OLD,DELETE)
//LIST04   DD DSN=&&PRPPRINT,DISP=(OLD,DELETE)
//LIST05   DD DSN=&&COBPRINT,DISP=(OLD,DELETE)
//LIST06   DD DSN=&&CB2PRINT,DISP=(OLD,DELETE)
//LIST07   DD DSN=&&LNKPRINT,DISP=(OLD,DELETE)
//LIST08   DD DSN=&&BNDPRINT,DISP=(OLD,DELETE)
//*
```

The GELTCL CSP Processor

```
//*********************************************************************
//** GENERATE, CICS TRANLATE, COMPILE AND LINK-EDIT CSP APPLICATIONS   *
//*********************************************************************
//*
//GELTCL  PROC CGHLQ='USER',
//         COBCICS='SYS1.COB2CICS',
//         COBCOMP='SYS1.COB2COMP',
//         COBLIB='SYS1.COB2LIB',
//         CONLIB='iprfx.iqual.CSIQLOAD',
//         CRS='CRS210',
//         CSP='CSP410',
//         CSPVSAM='CSP410',
//         CSPCUST='CSP410',
//         CSPUSER='CSPUSER',
//         DATA='31',
//         DBCS=,
//         DFHLOAD='SYS5.CICS212.LOADLIB',
//         ENV='MVSCICS',
//         LISTLIB=NULLFILE,
//         LOADLIB=NULLFILE,
//         MAP='NONE',
//         NLS='ENU',
//         RESLIB='IMSVS.RESLIB',  (UNCOMMENT BELOW IF NECESSARY)
//         SOUT='*',
//         SP=,
//         SUFF=1$,
//         TABLE='NONE',
//         USR1MSL=NULLFILE,
//         USR2MSL=NULLFILE,
//         WSPC=500
//*
//*
//* CSP PARMS: (PARM VALUES SHOULD BE SPECIFIED VIA PROC GROUPS)
//*
//*   CGHLQ   = COBOL GENERATION USER DATA SET HIGH-LEVEL QUALIFIER
//*   COBCICS = COBOL CICS RUN TIME LIBRARY
//*   COBCOMP = COBOL COMPILER LIBRARY
//*   COBLIB  = COBOL RUN TIME LIBRARY
//*   CRS     = CSP/370RS HIGH LEVEL QUALIFIER
//*   CSP     = CSP/370AD HIGH LEVEL QUALIFIER
//*   CSPVSAM = HIGH LEVEL QUALIFIER FOR VSAM DATA SETS
//*   CSPCUST = HIGH LEVEL QUALIFIER FOR CUSTOMIZED LIBRARIES
```

```
//*   CSPUSER = CSP/370AD USER WORK DATA SET HIGH LEVEL QUALIFIER
//*   DATA    = COMPILE OPTION FOR PLACING WORKING STORAGE
//*             ABOVE 16M LINE
//*   DBCS    = DOUBLE BYTE CHARACTER SUPPORT
//*   DFHLOAD = CICS TRANSLATOR LIBRARY
//*   ENV     = COBOL GENERATION USER DATA SET ENVIRONMENT QUALIFIER
//*             (SHOULD BE EQUAL TO GENERATION TARGET ENVIRONMENT)
//*   MAP     = NONE/ALL - FOR APPLICATION, GEN MAPS WITH APPL
//*   NLS     = NLS CODE (NATIONAL LANGUAGE DATA SET QUALIFIER)
//*   RESLIB  = IMS RESLIB LIBRARY
//*   SOUT    = SYSOUT ASSIGNMENT
//*   SP      = CICS/ESA SYSTEM PROGRAMMING TRANSLATOR OPTION
//*   SUFF    = CICS TRANSLATOR PROGRAM NAME SUFFIX
//*   TABLE   = NONE/ALL - FOR APPLICATION, GEN TABLES WITH APPL
//*   WSPC    = PRIMARY AND SECONDARY SPACE ALLOCATION
//*
//* Endevor PARMS: (PARM VALUES SHOULD BE SPECIFIED VIA PROC GROUPS)
//*   CONLIB  = Endevor RUN TIME LIBRARY
//*   LISTLIB = Endevor/CSP LISTINGS LIBRARY
//*   LOADLIB = Endevor/CSP OUTPUT LOAD LIBRARY
//*   USR1MSL = READ/WRITE MSL
//*   USR2MSL = READ-ONLY MSL
//*
//*********************************************************************
//*  INITIALIZE GENERATE/COMPILE/LINK-EDIT LISTING DATA SETS          *
//*********************************************************************
//*
//INITLIST EXEC PGM=BC1PDSIN,MAXRC=0
//C1INIT01 DD DSN=&&EZECOUT,DISP=(,PASS),
//            UNIT=tdisk,SPACE=(TRK,(5,5)),
//            DCB=(RECFM=FBA,LRECL=121,BLKSIZE=6171)
//C1INIT02 DD DSN=&&EZEPRINT,DISP=(,PASS),
//            UNIT=tdisk,SPACE=(TRK,(5,5)),
//            DCB=(RECFM=VBA,LRECL=654,BLKSIZE=6000)
//C1INIT02 DD DSN=&&TRNPRINT,DISP=(,PASS),
//            UNIT=tdisk,SPACE=(TRK,(15,5)),
//            DCB=(RECFM=FBA,LRECL=121,BLKSIZE=6171)
//C1INIT04 DD DSN=&&COBPRINT,DISP=(,PASS),
//            UNIT=tdisk,SPACE=(TRK,(15,5)),
//            DCB=(RECFM=FBA,LRECL=133,BLKSIZE=6251)
//C1INIT05 DD DSN=&&CB2PRINT,DISP=(,PASS),
//            UNIT=tdisk,SPACE=(TRK,(15,5)),
//            DCB=(RECFM=FBA,LRECL=133,BLKSIZE=6251)
//C1INIT06 DD DSN=&&LNKPRINT,DISP=(,PASS),
//            UNIT=tdisk,SPACE=(TRK,(5,5)),
//            DCB=(RECFM=FBA,LRECL=121,BLKSIZE=3630)
//*
//COBGEN EXEC PGM=BC1PDCGB,COND=(0,NE),MAXRC=4
```

```
//*----------------------
//*   LOADLIB AND MESSAGE FILES
//*----------------------
//STEPLIB DD DISP=SHR,DSN=&CRS..SELALMD
//        DD DISP=SHR,DSN=&CSP..SEZELMD
//*       DD DISP=SHR,DSN=EDC.SEDCLINK
//*       DD DISP=SHR,DSN=PLI.SIBMLINK
//        DD DISP=SHR,DSN=&COBLIB.
//        DD DISP=SHR,DSN=&CONLIB.
//*
//EZECOMM DD DISP=SHR,DSN=&CSPVSAM..&NLS..EZECOMM
//*
//*-------------------------------------------------------
//*  THE FOLLOWING MESSAGE FILES SHOULD BE PRE-ALLOCATED
//*-------------------------------------------------------
//*EZEDMSG DD DISP=SHR,DSN=&CSPVSAM..FZEMSG
//*EZEEMSG DD DISP=SHR,DSN=&CSPVSAM..&NLS..EZEMSG
//*EZETES  DD DISP=SHR,DSN=&CSPVSAM..&NLS..FZETUTOR
//*EZEMAP  DD DISP=SHR,DSN=&CSPVSAM..&NLS..FZEMAPDS
//*----------------------
//*  WORK FILES
//*----------------------
//EZEWORK DD DISP=OLD,DSN=&CSPUSER..EZEWORK
//EZEWRK1 DD UNIT=tdisk,SPACE=(CYL,(2,1)),
//           DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//*----------------------
//*  COMMAND AND LOG FILES
//*----------------------
//EZECOUT  DD DSN=&&EZECOUT,DISP=(OLD,PASS)
//*----------------------
//*  PRINT FILES
//*----------------------
//EZEPRINT DD DSN=&&EZEPRINT,DISP=(OLD,PASS)
//EZECPRT  DD SYSOUT=&SOUT.
//SYSPRINT DD SYSOUT=&SOUT.
//*----------------------
//*  COBOL GENERATION CONTROL FILES
//*----------------------
//EZEOPT  DD DISP=SHR,DSN=&CSPCUST..PDS.EZEOPT
//*EZEOPT  DD DISP=SHR,DSN=&CSPVSAM..VSAM.EZEOPT
//EZEJTMP DD DISP=SHR,DSN=&CSPCUST..EZESAMP
//EZEWORD DD DISP=SHR,DSN=&CSPCUST..EZESAMP(EZEWORDS)
//EZESNAP DD SYSOUT=&SOUT.
//*----------------------
//*  INTERNAL READER FILE FOR PREPARATION JCL SUBMISSION
//*----------------------
//EZEJCL  DD DSN=&&EZEJCL,DISP=(,PASS),
//           UNIT=tdisk,SPACE=(TRK,(1,1)),
//           DCB=(RECFM=FB,LRECL=80,BLKSIZE=6160)
```

```
//*EZEJCL  DD SYSOUT=(,INTRDR)
//*----------------------
//*----------------------
//*  ENVIRONMENT DEPENDENT CONTROL AND OUTPUT FILES
//*
//*  THE FOLLOWING FILES MAY OR MAY NOT BE REQUIRED, DEPENDING
//*  ON FUNCTIONS USED AT SPECIFIC INSTALLATIONS. THE DD CARDS
//*  ARE SHOWN AS COMMENTS IN THE PROCEDURE. THE GENERATOR USES
//*  DYNAMIC ALLOCATION TO ALLOCATE EXISTING DATA SETS WITH THE
//*  NAMES AS SHOWN IN THE DD CARDS, WHERE &CGHLQ. IS THE DATA SET
//*  QUALIFIER FROM THE USERID COBOL GENERATION OPTION AND &ENV.
//*  IS THE TARGET EXECUTION ENVIRONMENT.
//*
//*  TO USE DATA SETS WITH OTHER NAMES, MODIFY THE PROCEDURE OR THE
//*  JCL TO ALLOCATE THE DATA SETS WITH THE APPROPRIATE DD NAMES
//*  AS SHOWN BELOW. THE GENERATOR CHECKS TO SEE IF A DATA SET
//*  HAS ALREADY BEEN ALLOCATED TO THE DD NAME BEFORE ATTEMPTING TO
//*  DYNAMICALLY ALLOCATE A DATA SET WITH THE STANDARD NAME.
//*
//*----------------------
//*  COBOL SOURCE FILE
//*----------------------
//EZESRC  DD DISP=(NEW,PASS),DSN=&&EZESRC,
//           UNIT=tdisk,SPACE=(CYL,(5,5,10),RLSE),
//           DCB=(RECFM=FB,LRECL=80,BLKSIZE=6160)
//*
//EZEDUMMY DD DUMMY,DISP=SHR,DSN=&CGHLQ..&ENV..EZEDUMMY
//*----------------------
//*----------------------
//*  USER DEFINED LINK EDIT CONTROL STATEMENTS
//*----------------------
//EZELINK DD DUMMY
//*----------------------
//*  USER DEFINED BIND CONTROL STATEMENTS
//*----------------------
//EZEBIND DD DUMMY
//*-------------------------
//*  IMS COBOL COPY LIBRARY
//*-------------------------
//*EZECOPY DD DISP=SHR,DSM=&CGHLQ..&ENV..EZECOPY
//*----------------------
//*  OBJECT LIBRARIES FOR GENERATED MAP GROUPS
//*----------------------
//EZEFOBJ DD DISP=(NEW,PASS),DSN=&&EZEFOBJ,
//           UNIT=tdisk,SPACE=(TRK,(5,5,1),RLSE),
//           DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120)
//*EZEPOBJ DD DISP=SHR,DSN=&CGHLQ..&ENV..EZEPOBJ
//*----------------------
//*  TSO CLIST LIBRARY
```

```
//*----------------------
//EZECLST DD DISP=SHR,DSN=NULLFILE
//*----------------------
//*  PARTS LIST LIBRARY
//*----------------------
//EZEPCTL DD DISP=SHR,DSN=NULLFILE
//*----------------------
//*  MFS SOURCE LIBRARY
//*----------------------
//*EZEMFS  DD DISP=SHR,DSN=&CGHLQ..&ENV..EZEMFS
//*----------------------
//*  JCL LIBRARIES
//*----------------------
//*EZEJCLX DD DISP=SHR,DSN=&CGHLQ..&ENV..EZEJCLX
//*EZEJCLP DD DISP=SHR,DSN=&CGHLQ..&ENV..EZEJCLP
//USR1MSL  DD DISP=SHR,DSN=&USR1MSL.
//USR2MSL  DD DISP=SHR,DSN=&USR2MSL.
//NDVRIPT  DD *
 M=USR1MSL ROMSL=USR2MSL CMDIN=EZECIN CMDOUT=EZECOUT
//EZECIN  DD *
 GENERATE MEMBER(&C1ELEMENT). SYSTEM(&ENV).
      TABLES(&TABLE). MAPS(&MAP). BATCH(NO) ;
//EZEPARM  DD UNIT=tdisk,SPACE=(TRK,1),
//            DCB=(RECFM=FB,LRECL=80,BLKSIZE=80)
//*
//*********************************************************************
//*      EXTRACT THE LINKAGE-EDITOR CONTROL STATEMENTS FROM           **
//*      THE SEQUENTIAL FILE CREATED BY THE GENERATE STEP             **
//*********************************************************************
//*
//EXTRLNK EXEC PGM=LEXTRCTR,COND=(5,LT),MAXRC=0
//STEPLIB DD DISP=SHR,DSN=&CONLIB.
//INPUT   DD DSN=&&EZEJCL,DISP=(OLD,PASS)
//OUTPUT  DD DSN=&&CONTROL,DISP=(,PASS),
//           UNIT=tdisk,SPACE=(TRK,(1,1)),
//           DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120)
//*
//*********************************************************************
//** ELATCL - CICS TRANSLATE, COBOL COMPILE AND LINK-EDIT             **
//*********************************************************************
//T    EXEC PGM=DFHECP&SUFF.,MAXRC=4,
//       PARM='COBOL2,QUOTE,NOSEQ&SP&DBCS'.
//STEPLIB  DD DISP=SHR,DSN=&DFHLOAD.
//SYSPRINT DD DSN=&&TRNPRINT,DISP=(OLD,PASS)
//SYSPUNCH DD DSN=&&SYSCIN,DISP=(,PASS),
//            UNIT=tdisk,SPACE=(800,(&WSPC.,&WSPC)).
//SYSIN    DD DSN=&&EZESRC(&C1ELEMENT),DISP=(OLD,PASS)
//*
//*********************************************************************
```

```
//*      COMPILE THE COBOL PROGRAM                                      **
//*********************************************************************
//*
//C    EXEC PGM=IGYCRCTL,COND=(4,LT),MAXRC=4,
//     PARM=(NOSEQ,QUOTE,OFFSET,LIB,RES,RENT,NODYNAM,DBCS,OPT,
//     'TRUNC(BIN)','NUMPROC(NOPFD)',NOCMPR2,'DATA(&DATA)').
//STEPLIB DD DISP=SHR,DSN=&COBCOMP.
//SYSIN   DD DISP=(OLD,DELETE),DSN=&&SYSCIN
//SYSLIB  DD DISP=SHR,DSN=&CRS..SELACOPY,MONITOR=COMPONENTS
//SYSLIN  DD DISP=(NEW,PASS),DSN=&&LOADSET(&C1ELEMENT),UNIT=tdisk,
//           SPACE=(800,(&WSPC.,&WSPC.,10)),FOOTPRNT=CREATE,
//           DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120)
//SYSPRINT DD DSN=&&COBPRINT,DISP=(OLD,PASS)
//SYSUT1   DD SPACE=(800,(&WSPC.,&WSPC).,,,ROUND),UNIT=tdisk
//SYSUT2   DD SPACE=(800,(&WSPC.,&WSPC).,,,ROUND),UNIT=tdisk
//SYSUT3   DD SPACE=(800,(&WSPC.,&WSPC).,,,ROUND),UNIT=tdisk
//SYSUT4   DD SPACE=(800,(&WSPC.,&WSPC).,,,ROUND),UNIT=tdisk
//SYSUT5   DD SPACE=(800,(&WSPC.,&WSPC).,,,ROUND),UNIT=tdisk
//SYSUT6   DD SPACE=(800,(&WSPC.,&WSPC).,,,ROUND),UNIT=tdisk
//SYSUT7   DD SPACE=(800,(&WSPC.,&WSPC).,,,ROUND),UNIT=tdisk
//*
//*********************************************************************
//*      COMPILE ANY ADDITIONAL MAPS AND TABLES                         **
//*********************************************************************
//*
//C2    EXEC PGM=BC1PCCSP,COND=(4,LT),MAXRC=4,
//          PARM=(NOSEQ,QUOTE,OFFSET,LIB,RES,RENT,NODYNAM,DBCS,OPT,
//     'TRUNC(BIN)','NUMPROC(NOPFD)',NOCMPR2,'DATA(&DATA)').
//COBCOMP  DD DISP=SHR,DSN=&COBCOMP.
//SYSLIB   DD DISP=SHR,DSN=&CRS..SELACOPY,MONITOR=COMPONENTS
//SYSIN    DD UNIT=tdisk,SPACE=(6400,(&WSPC.,&WSPC)).,
//            DCB=(RECFM=FB,LRECL=80,BLKSIZE=6400)
//SYSLIN   DD UNIT=tdisk,SPACE=(3120,(&WSPC.,&WSPC)).,
//            DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120)
//SYSPRINT DD UNIT=tdisk,SPACE=(3630,(&WSPC.,&WSPC)).,
//            DCB=(RECFM=FBA,LRECL=133,BLKSIZE=6251)
//GENCNTL  DD DISP=(OLD,PASS),DSN=&&CONTROL
//EZESRC   DD DISP=(OLD,PASS),DSN=&&EZESRC
//OBJLIB   DD DISP=(OLD,PASS),DSN=&&LOADSET
//PRINTLIB DD DISP=(OLD,PASS),DSN=&&CB2PRINT
//GENINFO  DD *
ELEMENT=&C1ELEMENT.       KEEP THESE CARDS IN ORDER,
 COMPILER=IGYCRCTL       LEFT JUST WITH NO BLANKS
//SYSUT1  DD SPACE=(800,(&WSPC.,&WSPC).,,,ROUND),UNIT=tdisk
//SYSUT2  DD SPACE=(800,(&WSPC.,&WSPC).,,,ROUND),UNIT=tdisk
//SYSUT3  DD SPACE=(800,(&WSPC.,&WSPC).,,,ROUND),UNIT=tdisk
//SYSUT4  DD SPACE=(800,(&WSPC.,&WSPC).,,,ROUND),UNIT=tdisk
//SYSUT5  DD SPACE=(800,(&WSPC.,&WSPC).,,,ROUND),UNIT=tdisk
//SYSUT6  DD SPACE=(800,(&WSPC.,&WSPC).,,,ROUND),UNIT=tdisk
```

```
//SYSUT7  DD SPACE=(800,(&WSPC.,&WSPC).,,,ROUND),UNIT=tdisk
//*
//*
//*********************************************************************
//*      LINK-EDIT THE COBOL PROGRAM                                  **
//*      IF THE RETURN CODE ON ALL PREVIOUS STEPS IS 4 OR LESS        **
//*********************************************************************
//*
//L    EXEC PGM=IEWL,COND=(5,LT),MAXRC=0,
//     PARM='RENT,REUS,LIST,XREF,MAP,AMODE(31),RMODE(ANY)'
//SYSLIB  DD DISP=SHR,DSN=&DFHLOAD.,MONITOR=COMPONENTS
//        DD DISP=SHR,DSN=&COBCICS.,MONITOR=COMPONENTS
//        DD DISP=SHR,DSN=&COBLIB.,MONITOR=COMPONENTS
//*       DD DISP=SHR,DSN=&RESLIB.,MONITOR=COMPONENTS
//OBJLIB  DD DISP=(OLD,DELETE),DSN=&&LOADSET,MONITOR=COMPONENTS
//        DD DISP=(OLD,DELETE),DSN=&&EZEFOBJ,MONITOR=COMPONENTS
//SELALMD DD DISP=SHR,DSN=&CRS..SELALMD,MONITOR=COMPONENTS
//SYSLMOD DD DISP=SHR,DSN=&LOADLIB.,
//       FOOTPRNT=CREATE,MONITOR=COMPONENTS
//SYSPRINT DD DSN=&&LNKPRINT,DISP=(OLD,PASS)
//SYSUT1   DD SPACE=(1024,(&WSPC.,&WSPC)).,UNIT=tdisk
//SYSLIN   DD DSN=&&CONTROL,DISP=(OLD,DELETE)
//*********************************************************************
//*  STORE OUTPUT LISTINGS IN LIST LIBRARY                          **
//*********************************************************************
//CONLIST  EXEC PGM=CONLIST,PARM=STORE,MAXRC=0,COND=EVEN
//C1LLIBO  DD DSN=&LISTLIB.,DISP=SHR,MONITOR=COMPONENTS
//C1BANNER DD UNIT=tdisk,SPACE=(TRK,1),
//            DCB=(RECFM=FBA,LRECL=121)
//LIST01   DD DSN=&&EZECOUT,DISP=(OLD,DELETE)
//LIST02   DD DSN=&&EZEPRINT,DISP=(OLD,DELETE)
//LIST03   DD DSN=&&EZEJCL,DISP=(OLD,DELETE)
//LIST04   DD DSN=&&TRNPRINT,DISP=(OLD,DELETE)
//LIST05   DD DSN=&&COBPRINT,DISP=(OLD,DELETE)
//LIST06   DD DSN=&&CB2PRINT,DISP=(OLD,DELETE)
//LIST07   DD DSN=&&LNKPRINT,DISP=(OLD,DELETE)
//*
```

The GELPTCLB CSP Processor

```
//*********************************************************************
//** GENERATE, DB2 PRE-PROCESS, CICS TRANSLATE, COMPILE, LINK-EDIT    **
//**  AND BIND CSP APPLICATIONS                                      **
//*********************************************************************
//*
//GELPTCLB PROC CGHLQ='USER',
//          BINDDSN  ='&C1USERID..NDVRCSP.BINDDSN',
//          COBCICS  ='SYS1.COB2CICS',
//          COBCOMP  ='SYS1.COB2COMP',
//          COBLIB   ='SYS1.COB2LIB',
//          CONLIB   ='iprfx.iqual.CSIQLOAD',
//          CRS      ='CRS210',
//          CSP      ='CSP410',
//          CSPVSAM  ='CSP410',
//          CSPCUST  ='CSP410',
//          CSPUSER  ='CSPUSER',
//          DATA     ='31',
//          DBRMLIB  ='NULLFILE',
//          DSNEXIT  ='DSN220.DSNEXIT',
//          DSNLOAD  ='DSN220.DSNLOAD',
//          ENV      ='MVSCICS',
//          DBCS     =,
//          DFHLOAD  ='SYS5.CICS212.LOADLIB',
//          LISTLIB  ='NULLFILE',
//          LOADLIB  ='NULLFILE',
//          MAP      ='NONE',
//          NLS      ='ENU',
//          RESLIB   ='IMSVS.RESLIB',  (UNCOMMENT BELOW IF NECESSARY)
//          SOUT     ='*',
//          SP       =,
//          SUFF     =1$,
//          TABLE    ='NONE',
//          USR1MSL  ='NULLFILE',
//          USR2MSL  ='NULLFILE',
//          WSPC     =500
//*
//*
//* CSP PARMS: (PARM VALUES SHOULD BE SPECIFIED VIA PROC GROUPS)
//*
//*    CGHLQ   = COBOL GENERATION USER DATA SET HIGH-LEVEL QUALIFIER
//*    COBCICS = COBOL CICS RUN TIME LIBRARY
//*    COBCOMP = COBOL COMPILER LIBRARY
//*    COBLIB  = COBOL RUN TIME LIBRARY
```

```
//*    CRS     = CSP/370RS HIGH LEVEL QUALIFIER
//*    CSP     = CSP/370AD HIGH LEVEL QUALIFIER
//*    CSPVSAM = HIGH LEVEL QUALIFIER FOR VSAM DATA SETS
//*    CSPCUST = HIGH LEVEL QUALIFIER FOR CUSTOMIZED LIBRARIES
//*    CSPUSER = CSP/370AD USER WORK DATA SET HIGH LEVEL QUALIFIER
//*    DATA    = COMPILE OPTION FOR PLACING WORKING STORAGE
//*              ABOVE 16M LINE
//*   DBCS     = DOUBLE BYTE CHARACTER SUPPORT
//*   DFHLOAD  = CICS TRANSLATOR LIBRARY
//*   DSNEXIT  = DB2 DSNEXIT LIBRARY
//*   DSNLOAD  = DB2 DSNLOAD LIBRARY
//*   ENV      = COBOL GENERATION USER DATA SET ENVIRONMENT QUALIFIER
//*              (SHOULD BE EQUAL TO GENERATION TARGET ENVIRONMENT)
//*   MAP      = NONE/ALL - FOR APPLICATION, GEN MAPS WITH APPL
//*   NLS      = NLS CODE (NATIONAL LANGUAGE DATA SET QUALIFIER)
//*   RESLIB   = IMS RESLIB LIBRARY
//*   SOUT     = SYSOUT ASSIGNMENT
//*   SP       = CICS/ESA SYSTEM PROGRAMMING TRANSLATOR OPTION
//*   SUFF     = CICS TRANSLATOR PROGRAM NAME SUFFIX
//*   TABLE    = NONE/ALL - FOR APPLICATION, GEN TABLES WITH APPL
//*   WSPC     = PRIMARY AND SECONDARY SPACE ALLOCATION
//*
//* Endevor PARMS: (PARM VALUES SHOULD BE SPECIFIED VIA PROC GROUPS)
//*   BINDDSN  = 'PERMANENTLY-ALLOCATED' TEMPORARY BIND CNTL CARD LIB
//*   CONLIB   = Endevor RUN TIME LIBRARY
//*   DBRMLIB  = DB2 DBRMLIB LIBRARY
//*   LISTLIB  = Endevor/CSP LISTINGS LIBRARY
//*   LOADLIB  = Endevor/CSP OUTPUT LOAD LIBRARY
//*   USR1MSL  = READ/WRITE MSL
//*   USR2MSL  = READ-ONLY MSL
//*
//*
//*********************************************************************
//*  INITIALIZE GENERATE/COMPILE/LINK-EDIT LISTING DATA SETS          *
//*********************************************************************
//*
//INITLIST EXEC PGM=BC1PDSIN,MAXRC=0
//C1INIT01 DD DSN=&&EZECOUT,DISP=(,PASS),
//            UNIT=tdisk,SPACE=(TRK,(5,5)),
//            DCB=(RECFM=FBA,LRECL=121,BLKSIZE=6171)
//C1INIT02 DD DSN=&&EZEPRINT,DISP=(,PASS),
//            UNIT=tdisk,SPACE=(TRK,(5,5)),
//            DCB=(RECFM=VBA,LRECL=654,BLKSIZE=6000)
//C1INIT03 DD DSN=&&PRPPRINT,DISP=(,PASS),
//            UNIT=tdisk,SPACE=(TRK,(15,5)),
//            DCB=(RECFM=FBA,LRECL=133,BLKSIZE=6251)
//C1INIT04 DD DSN=&&TRNPRINT,DISP=(,PASS),
//            UNIT=tdisk,SPACE=(TRK,(15,5)),
```

```
//            DCB=(RECFM=FBA,LRECL=121,BLKSIZE=6171)
//C1INIT05 DD DSN=&&COBPRINT,DISP=(,PASS),
//            UNIT=tdisk,SPACE=(TRK,(15,5)),
//            DCB=(RECFM=FBA,LRECL=133,BLKSIZE=6251)
//C1INIT06 DD DSN=&&CB2PRINT,DISP=(,PASS),
//            UNIT=tdisk,SPACE=(TRK,(15,5)),
//            DCB=(RECFM=FBA,LRECL=133,BLKSIZE=6251)
//C1INIT07 DD DSN=&&LNKPRINT,DISP=(,PASS),
//            UNIT=tdisk,SPACE=(TRK,(5,5)),
//            DCB=(RECFM=FBA,LRECL=121,BLKSIZE=3630)
//C1INIT08 DD DSN=&&BNDPRINT,DISP=(,PASS),
//            UNIT=tdisk,SPACE=(TRK,(5,2)),
//            DCB=(RECFM=FBA,LRECL=121,BLKSIZE=6171)
//*********************************************************************
//* INVOKE IDCAMS TO ENSURE THE BIND CNTL CARD O/P DATA SET IS DELETED*
//*********************************************************************
//IDCAMS EXEC PGM=IDCAMS
//SYSPRINT DD DUMMY
//SYSIN    DD *
 DELETE '&BINDDSN'.
 SET MAXCC = 0
//COBGEN EXEC PGM=BC1PDCGB,COND=(0,NE),MAXRC=4
//*
//*-----------------------
//*  LOADLIB AND MESSAGE FILES
//*-----------------------
//STEPLIB DD DISP=SHR,DSN=&CRS..SELALMD
//        DD DISP=SHR,DSN=&CSP..SEZELMD
//*       DD DISP=SHR,DSN=EDC.SEDCLINK
//*       DD DISP=SHR,DSN=PLI.SIBMLINK
//        DD DISP=SHR,DSN=&COBLIB.
//        DD DISP=SHR,DSN=&CONLIB.
//*
//EZECOMM DD DISP=SHR,DSN=&CSPVSAM..&NLS..EZECOMM
//*
//*-------------------------------------------------------
//*  THE FOLLOWING MESSAGE FILES SHOULD BE PRE-ALLOCATED
//*-------------------------------------------------------
//*EZEDMSG DD DISP=SHR,DSN=&CSPVSAM..FZEMSG
//*EZEEMSG DD DISP=SHR,DSN=&CSPVSAM..&NLS..EZEMSG
//*EZETES  DD DISP=SHR,DSN=&CSPVSAM..&NLS..FZETUTOR
//*EZEMAP  DD DISP=SHR,DSN=&CSPVSAM..&NLS..FZEMAPDS
//*-----------------------
//*  WORK FILES
//*-----------------------
//EZEWORK DD DISP=OLD,DSN=&CSPUSER..EZEWORK
//EZEWRK1 DD UNIT=tdisk,SPACE=(CYL,(2,1)),
//           DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//*-----------------------
```

```
//*  COMMAND AND LOG FILES
//*----------------------
//EZECOUT DD DSN=&&EZECOUT,DISP=(OLD,PASS)
//*----------------------
//*  PRINT FILES
//*----------------------
//EZEPRINT DD DSN=&&EZEPRINT,DISP=(OLD,PASS)
//EZECPRT  DD SYSOUT=&SOUT.
//SYSPRINT DD SYSOUT=&SOUT.
//*----------------------
//*  COBOL GENERATION CONTROL FILES
//*----------------------
//EZEOPT   DD DISP=SHR,DSN=&CSPCUST..PDS.EZEOPT
//*EZEOPT  DD DISP=SHR,DSN=&CSPVSAM..VSAM.EZEOPT
//EZEJTMP  DD DISP=SHR,DSN=&CSPCUST..EZESAMP
//EZEWORD  DD DISP=SHR,DSN=&CSPCUST..EZESAMP(EZEWORDS)
//EZESNAP  DD SYSOUT=&SOUT.
//*----------------------
//*  INTERNAL READER FILE FOR PREPARATION JCL SUBMISSION
//*----------------------
//EZEJCL   DD DSN=&&EZEJCL,DISP=(,PASS),
//            UNIT=tdisk,SPACE=(TRK,(1,1)),
//            DCB=(RECFM=FB,LRECL=80,BLKSIZE=6160)
//*EZEJCL  DD SYSOUT=(,INTRDR)
//*----------------------
//*----------------------
//*  ENVIRONMENT DEPENDENT CONTROL AND OUTPUT FILES
//*
//*  THE FOLLOWING FILES MAY OR MAY NOT BE REQUIRED, DEPENDING
//*  ON FUNCTIONS USED AT SPECIFIC INSTALLATIONS. THE DD CARDS
//*  ARE SHOWN AS COMMENTS IN THE PROCEDURE. THE GENERATOR USES
//*  DYNAMIC ALLOCATION TO ALLOCATE EXISTING DATA SETS WITH THE
//*  NAMES AS SHOWN IN THE DD CARDS, WHERE &CGHLQ. IS THE DATA SET
//*  QUALIFIER FROM THE USERID COBOL GENERATION OPTION AND &ENV.
//*  IS THE TARGET EXECUTION ENVIRONMENT.
//*
//*  TO USE DATA SETS WITH OTHER NAMES, MODIFY THE PROCEDURE OR THE
//*  JCL TO ALLOCATE THE DATA SETS WITH THE APPROPRIATE DD NAMES
//*  AS SHOWN BELOW. THE GENERATOR CHECKS TO SEE IF A DATA SET
//*  HAS ALREADY BEEN ALLOCATED TO THE DD NAME BEFORE ATTEMPTING TO
//*  DYNAMICALLY ALLOCATE A DATA SET WITH THE STANDARD NAME.
//*
//*----------------------
//*  COBOL SOURCE FILE
//*----------------------
//EZESRC  DD DISP=(NEW,PASS),DSN=&&EZESRC,
//           UNIT=tdisk,SPACE=(CYL,(5,5,10),RLSE),
//           DCB=(RECFM=FB,LRECL=80,BLKSIZE=6160)
//*
```

```
//EZEDUMMY DD DUMMY,DISP=SHR,DSN=&CGHLQ..&ENV..EZEDUMMY
//*----------------------
//*----------------------
//*  USER DEFINED LINK EDIT CONTROL STATEMENTS
//*----------------------
//EZELINK DD DUMMY
//*----------------------
//*  USER-DEFINED BIND CONTROL STATEMENTS
//*----------------------
//EZEBIND DD DSN=&BINDDSN.,DISP=(,CATLG,DELETE),
//           UNIT=tdisk,SPACE=(TRK,(1,1,1)),
//           DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120)
//*-------------------------
//*  IMS COBOL COPY LIBRARY
//*-------------------------
//*EZECOPY DD DISP=SHR,DSM=&CGHLQ..&ENV..EZECOPY
//*----------------------
//*  OBJECT LIBRARIES FOR GENERATED MAP GROUPS
//*----------------------
//EZEFOBJ DD DISP=(NEW,PASS),DSN=&&EZEFOBJ,
//           UNIT=tdisk,SPACE=(TRK,(5,5,1),RLSE),
//           DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120)
//*EZEPOBJ DD DISP=SHR,DSN=&CGHLQ..&ENV..EZEPOBJ
//*----------------------
//*  TSO CLIST LIBRARY
//*----------------------
//EZECLST DD DISP=SHR,DSN=NULLFILE
//*----------------------
//*  PARTS LIST LIBRARY
//*----------------------
//EZEPCTL DD DISP=SHR,DSN=NULLFILE
//*----------------------
//*  MFS SOURCE LIBRARY
//*----------------------
//*EZEMFS  DD DISP=SHR,DSN=&CGHLQ..&ENV..EZEMFS
//*----------------------
//*  JCL LIBRARIES
//*----------------------
//*EZEJCLX DD DISP=SHR,DSN=&CGHLQ..&ENV..EZEJCLX
//*EZEJCLP DD DISP=SHR,DSN=&CGHLQ..&ENV..EZEJCLP
//USR1MSL  DD DISP=SHR,DSN=&USR1MSL.
//USR2MSL  DD DISP=SHR,DSN=&USR2MSL.
//NDVRIPT  DD *
 M=USR1MSL ROMSL=USR2MSL CMDIN=EZECIN CMDOUT=EZECOUT
//EZECIN  DD *
 GENERATE MEMBER(&C1ELEMENT). SYSTEM(&ENV).
      TABLES(&TABLE). MAPS(&MAP). BATCH(NO) ;
//EZEPARM  DD UNIT=tdisk,SPACE=(TRK,1),
//            DCB=(RECFM=FB,LRECL=80,BLKSIZE=80)
```

```
//*
//*********************************************************************
//*      EXTRACT THE LINKAGE-EDITOR CONTROL STATEMENTS FROM           **
//*      THE SEQUENTIAL FILE CREATED BY THE GENERATE STEP             **
//*********************************************************************
//*
//EXTRLNK EXEC PGM=LEXTRCTR,COND=(5,LT),MAXRC=0
//STEPLIB DD DISP=SHR,DSN=&CONLIB.
//INPUT   DD DSN=&&EZEJCL,DISP=(OLD,PASS)
//OUTPUT  DD DSN=&&CONTROL,DISP=(,PASS),
//           UNIT=tdisk,SPACE=(TRK,(1,1)),
//           DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120)
//*
//*********************************************************************
//** ELAPCLB - COBOL PRECOMPILE/COMPILE AND LINK-EDIT                 **
//*********************************************************************
//*
//*****************************************************************
//*      PRECOMPILE THE COBOL PROGRAM                             **
//*****************************************************************
//*
//P     EXEC PGM=DSNHPC,MAXRC=4,
//      PARM=('HOST(COB2),APOSTSQL,QUOTE')
//STEPLIB DD DISP=SHR,DSN=&DSNEXIT.
//        DD DISP=SHR,DSN=&DSNLOAD.
//DBRMLIB DD DISP=OLD,DSN=&DBRMLIB.(&C1ELEMENT).
//SYSCIN  DD DISP=(NEW,PASS),DSN=&&DSNHOUT,UNIT=tdisk,
//           SPACE=(800,(&WSPC.,&WSPC)).
//SYSLIB  DD DISP=SHR,DSN=&CRS..SELACOPY,MONITOR=COMPONENTS
//SYSPRINT DD DSN=&&PRPPRINT,DISP=(OLD,PASS)
//SYSUT1  DD SPACE=(800,(&WSPC.,&WSPC).,,,ROUND),UNIT=tdisk
//SYSUT2  DD SPACE=(800,(&WSPC.,&WSPC).,,,ROUND),UNIT=tdisk
//SYSIN   DD DISP=(OLD,PASS),DSN=&&EZESRC(&C1ELEMENT)
//*
//*********************************************************************
//** ELATCL - CICS TRANSLATE, COBOL COMPILE AND LINK-EDIT             **
//*********************************************************************
//T      EXEC PGM=DFHECP&SUFF.,MAXRC=4,
//       PARM='COBOL2,QUOTE,NOSEQ&SP&DBCS'.
//STEPLIB  DD DISP=SHR,DSN=&DFHLOAD.
//SYSPRINT DD DSN=&&TRNPRINT,DISP=(OLD,PASS)
//SYSPUNCH DD DSN=&&SYSCIN,DISP=(,PASS),
//            UNIT=tdisk,SPACE=(800,(&WSPC.,&WSPC)).
//SYSIN    DD DSN=&&DSNHOUT,DISP=(OLD,DELETE)
//*
//*
//*********************************************************************
//*      COMPILE THE COBOL PROGRAM                                    **
//*********************************************************************
```

```
//*
//C     EXEC PGM=IGYCRCTL,COND=(4,LT),MAXRC=4,
//      PARM=(NOSEQ,QUOTE,OFFSET,LIB,RES,RENT,NODYNAM,DBCS,OPT,
//     'TRUNC(BIN)','NUMPROC(NOPFD)',NOCMPR2,'DATA(&DATA)').
//STEPLIB DD DISP=SHR,DSN=&COBCOMP.
//SYSIN   DD DISP=(OLD,DELETE),DSN=&&SYSCIN
//SYSLIB  DD DISP=SHR,DSN=&CRS..SELACOPY,MONITOR=COMPONENTS
//SYSLIN  DD DISP=(NEW,PASS),DSN=&&LOADSET(&C1ELEMENT),UNIT=tdisk,
//           SPACE=(800,(&WSPC.,&WSPC.,10)),FOOTPRNT=CREATE,
//           DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120)
//SYSPRINT DD DSN=&&COBPRINT,DISP=(OLD,PASS)
//SYSUT1   DD SPACE=(800,(&WSPC.,&WSPC).,,,ROUND),UNIT=tdisk
//SYSUT2   DD SPACE=(800,(&WSPC.,&WSPC).,,,ROUND),UNIT=tdisk
//SYSUT3   DD SPACE=(800,(&WSPC.,&WSPC).,,,ROUND),UNIT=tdisk
//SYSUT4   DD SPACE=(800,(&WSPC.,&WSPC).,,,ROUND),UNIT=tdisk
//SYSUT5   DD SPACE=(800,(&WSPC.,&WSPC).,,,ROUND),UNIT=tdisk
//SYSUT6   DD SPACE=(800,(&WSPC.,&WSPC).,,,ROUND),UNIT=tdisk
//SYSUT7   DD SPACE=(800,(&WSPC.,&WSPC).,,,ROUND),UNIT=tdisk
//
* ********************************************************************
//*      COMPILE ANY ADDITIONAL MAPS AND TABLES                       **
//********************************************************************
//*
//C2     EXEC PGM=BC1PCCSP,COND=(4,LT),MAXRC=4,
//       PARM=(NOSEQ,QUOTE,OFFSET,LIB,RES,RENT,NODYNAM,DBCS,OPT,
//      'TRUNC(BIN)','NUMPROC(NOPFD)',NOCMPR2,'DATA(&DATA)').
//COBCOMP  DD DISP=SHR,DSN=&COBCOMP.
//SYSLIB   DD DISP=SHR,DSN=&CRS..SELACOPY,MONITOR=COMPONENTS
//SYSIN    DD UNIT=tdisk,SPACE=(6400,(&WSPC.,&WSPC)).,
//            DCB=(RECFM=FB,LRECL=80,BLKSIZE=6400)
//SYSLIN   DD UNIT=tdisk,SPACE=(3120,(&WSPC.,&WSPC)).,
//            DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120)
//SYSPRINT  DD UNIT=tdisk,SPACE=(3630,(&WSPC.,&WSPC)).,
//             DCB=(RECFM=FBA,LRECL=133,BLKSIZE=6251)
//GENCNTL  DD DISP=(OLD,PASS),DSN=&&CONTROL
//EZESRC   DD DISP=(OLD,PASS),DSN=&&EZESRC
//OBJLIB   DD DISP=(OLD,PASS),DSN=&&LOADSET
//PRINTLIB DD DISP=(OLD,PASS),DSN=&&CB2PRINT
//GENINFO DD *
ELEMENT=&C1ELEMENT.       KEEP THESE CARDS IN ORDER,
 COMPILER=IGYCRCTL       LEFT JUST WITH NO BLANKS
//SYSUT1  DD SPACE=(800,(&WSPC.,&WSPC).,,,ROUND),UNIT=tdisk
//SYSUT2  DD SPACE=(800,(&WSPC.,&WSPC).,,,ROUND),UNIT=tdisk
//SYSUT3  DD SPACE=(800,(&WSPC.,&WSPC).,,,ROUND),UNIT=tdisk
//SYSUT4  DD SPACE=(800,(&WSPC.,&WSPC).,,,ROUND),UNIT=tdisk
//SYSUT5  DD SPACE=(800,(&WSPC.,&WSPC).,,,ROUND),UNIT=tdisk
//SYSUT6  DD SPACE=(800,(&WSPC.,&WSPC).,,,ROUND),UNIT=tdisk
//SYSUT7  DD SPACE=(800,(&WSPC.,&WSPC).,,,ROUND),UNIT=tdisk
//*
```

```
//*
//*********************************************************************
//*      LINK-EDIT THE COBOL PROGRAM                                  **
//*      IF THE RETURN CODE ON ALL PREVIOUS STEPS IS 4 OR LESS        **
//*********************************************************************
//*
//L      EXEC PGM=IEWL,COND=(5,LT),MAXRC=0,
//       PARM='RENT,REUS,LIST,XREF,MAP,AMODE(31),RMODE(ANY)'
//SYSLIB  DD DISP=SHR,DSN=&DFHLOAD.,MONITOR=COMPONENTS
//        DD DISP=SHR,DSN=&COBCICS.,MONITOR=COMPONENTS
//        DD DISP=SHR,DSN=&COBLIB.,MONITOR=COMPONENTS
//        DD DISP=SHR,DSN=&DSNLOAD.,MONITOR=COMPONENTS
//*       DD DISP=SHR,DSN=&RESLIB.,MONITOR=COMPONENTS
//OBJLIB  DD DISP=(OLD,DELETE),DSN=&&LOADSET,MONITOR=COMPONENTS
//     DD DISP=(OLD,DELETE),DSN=&&EZEFOBJ,MONITOR=COMPONENTS
//SELALMD DD DISP=SHR,DSN=&CRS..SELALMD,MONITOR=COMPONENTS
//SYSLMOD DD DISP=SHR,DSN=&LOADLIB.,
//       FOOTPRNT=CREATE,MONITOR=COMPONENTS
//SYSPRINT DD DSN=&&LNKPRINT,DISP=(OLD,PASS)
//SYSUT1   DD SPACE=(1024,(&WSPC.,&WSPC)).,UNIT=tdisk
//SYSLIN   DD DSN=&&CONTROL,DISP=(OLD,DELETE)
//*
//*********************************************************************
//* BIND APPLICATION PLAN IN FOREGROUND                               **
//* NOTE: ATTEMPTING TO RUN THIS STEP IN BG WILL RESULT IN RC=5       **
//*********************************************************************
//BINDFG  EXEC PGM=BC1PTMP0,COND=(8,LT),MAXRC=5,
//        PARM='&BINDDSN(&C1ELEMENT)'.
//STEPLIB  DD DISP=SHR,DSN=&DSNLOAD.
//DBRMLIB  DD DISP=SHR,DSN=&DBRMLIB.,MONITOR=COMPONENTS
//         DD DISP=SHR,DSN=&CRS..SELADBRM,MONITOR=COMPONENTS
//SYSTSPRT DD DISP=(OLD,PASS),DSN=&&BNDPRINT
//*********************************************************************
//* BIND APPLICATION PLAN IN BACKGROUND                               **
//*********************************************************************
//BINDBG  EXEC PGM=IKJEFT01,COND=((8,LT),(5,NE,BINDFG)),MAXRC=0
//STEPLIB DD DISP=SHR,DSN=&DSNLOAD.
//SYSTSIN DD DISP=SHR,DSN=&BINDDSN.(&C1ELEMENT).
//DBRMLIB DD DISP=SHR,DSN=&DBRMLIB.,MONITOR=COMPONENTS
//        DD DISP=SHR,DSN=&CRS..SELADBRM,MONITOR=COMPONENTS
//SYSTSPRT DD DISP=(OLD,PASS),DSN=&&BNDPRINT
//*********************************************************************
//* FOOTPRINT DB2 PLAN                                                **
//*********************************************************************
//FOOTDB2 EXEC PGM=BC1PCAF,COND=(8,LT),MAXRC=0,
//       PARM='DB2T,BC1PSQL1,BC1PDBFP'
//STEPLIB DD DISP=SHR,DSN=&DSNLOAD.
//DBRMLIB DD DISP=SHR,DSN=&DBRMLIB.
//        DD DISP=SHR,DSN=&CRS..SELADBRM
```

```
//BSTIPT  DD DISP=SHR,DSN=&BINDDSN.(&C1ELEMENT).
//*********************************************************************
//* INVOKE IDCAMS TO ENSURE THE BIND CNTL CARD O/P DATA SET IS DELETED*
//*********************************************************************
//IDCAMS   EXEC PGM=IDCAMS
//SYSPRINT DD DUMMY
//SYSIN    DD *
 DELETE '&BINDDSN'.
 SET MAXCC = 0
//*********************************************************************
//*  STORE OUTPUT LISTINGS IN LIST LIBRARY                          **
//*********************************************************************
//CONLIST  EXEC PGM=CONLIST,PARM=STORE,MAXRC=0,COND=EVEN
//C1LLIBO  DD DSN=&LISTLIB.,DISP=SHR,MONITOR=COMPONENTS
//C1BANNER DD UNIT=tdisk,SPACE=(TRK,1),
//            DCB=(RECFM=FBA,LRECL=121)
//LIST01   DD DSN=&&EZECOUT,DISP=(OLD,DELETE)
//LIST02   DD DSN=&&EZEPRINT,DISP=(OLD,DELETE)
//LIST03   DD DSN=&&EZEJCL,DISP=(OLD,DELETE)
//LIST04   DD DSN=&&PRPPRINT,DISP=(OLD,DELETE)
//LIST05   DD DSN=&&TRNPRINT,DISP=(OLD,DELETE)
//LIST06   DD DSN=&&COBPRINT,DISP=(OLD,DELETE)
//LIST07   DD DSN=&&CB2PRINT,DISP=(OLD,DELETE)
//LIST08   DD DSN=&&LNKPRINT,DISP=(OLD,DELETE)
//LIST09   DD DSN=&&BNDPRINT,DISP=(OLD,DELETE)
//*
```

The GCSP41LN CSP Processor

```
//************************+++ GCSP41LN +++*****************************
//* GENERATE PROCESSOR THE CSP EXTENSION FACILITY.
//* EXTENSION TYPES ARE RECDEX, PROCEX AND SGRPEX.
//* THIS PROCESSOR PERFORMS THE FOLLOWING ACTIVITIES:
//* 1) ALLOCATE LISTING DATA SETS.
//* 2) WRITE A CURRENT COPY OF THE ELEMENT TO A TEMP FILE.
//* 3) ALLOCATE EZEWORK FILE.
//* 4) INVOKE CSP/AD VIA BC1PDCGB TO IMPORT THE MEMBER FROM
//* THE TEMP FILE TO THE STAGE MSL.
//* 5) WRITE LISTINGS TO LISTINGS LIBRARY.
//*********************************************************************
//*
//GCSP41LN PROC COBLIB='SYS1.COB2LIB', COBOL RUN-TIME LIBRARY00130002
//         CONLIB='iprfx.iqual.CSIQLOAD', 00140002
//         CRS='CRS210', CSP/370RS HIGH LEVEL QUALIFIER
//         CSP='CSP410', CSP/370AD HIGH LEVEL QUALIFIER
//         CSPUSER='CSPUSER', USER WORK DATASET HIGH QUALIFIER
//         CSPVSAM='CSP410', HIGH LEVEL QUALIFIER - VSAM DS
//         LISTLIB='NULLFILE', Endevor/CSP LISTINGS LIBRARY
//         NLS='LLL', NATIONAL LANGUAGE QUALIFIER
//         WORKDISK='tdisk', UNITNAME FOR TEMP DATASETS
//         USR1MSL='NULLFILE', READ/WRITE MSL
//         USR2MSL='NULLFILE', READ-ONLY MSL
//         vvolser=''VOLSER FOR VSAM EZEWORK
//*********************************************************************
//* STEP1 - ALLOCATE AND INITIALIZE ALL THE TEMP DATA SETS            **
//*********************************************************************
//INITTDS EXEC PGM=BC1PDSIN,MAXRC=0
//C1INIT01  DD DISP=(,PASS),DSN=&&AMSLST,
//             UNIT=&WORKDISK.,SPACE=(TRK,(5,2),RLSE),
//             DCB=(RECFM=FBA,LRECL=121,BLKSIZE=1210)
//C1INIT02  DD DISP=(,PASS),DSN=&&LOGLST,
//             UNIT=&WORKDISK.,SPACE=(TRK,(5,2),RLSE),
//             DCB=(RECFM=FBA,LRECL=121,BLKSIZE=1210)
//C1INIT03  DD DISP=(,PASS),DSN=&&CMDLST,
//             UNIT=&WORKDISK.,SPACE=(TRK,(5,2),RLSE),
//             DCB=(RECFM=VBA,LRECL=650,BLKSIZE=654)
//C1INIT04  DD DISP=(,PASS),DSN=&&TEMPSRC,
//             UNIT=&WORKDISK.,SPACE=(TRK,(5,2),RLSE),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120)
//*********************************************************************
//* STEP2 - WRITE A CURRENT COPY OF THE ELEMENT TO THE ESF DATA SET   **
//*********************************************************************
//CONWRT EXEC PGM=CONWRITE,PARM='EXPINCL(N)',COND=(0,LT),MAXRC=0
//ELMOUT DD DISP=(OLD,PASS),DSN=&&TEMPSRC
```

```
//*********************************************************************
//* STEP3 - ALLOCATE THE EZEWORK VSAM FILE                            **
//*********************************************************************
//ALLOWRK  EXEC PGM=IDCAMS,COND=(0,LT),MAXRC=0
//SYSPRINT DD DISP=(OLD,PASS),DSN=&&AMSLST
//SYSIN DD *
 DELETE '&CSPUSER..EZEWORK' PURGE
 SET MAXCC=0
 DEFINE CLUSTER (NAME('&CSPUSER..EZEWORK') -
        VOL(&vvolser). -
        CYLINDERS(02 01) -
        KEYS(14 0) -
        RECORDSIZE(272 272) -
        SHR (3 3) -
      INDEXED            ) -
 DATA  (NAME('&CSPUSER..EZEWORK.DATA')) -
 INDEX (NAME('&CSPUSER..EZEWORK.INDEX'))
//*********************************************************************
//* STEP4 - INVOKE CSP/AD VIA E/CSP UTILITY TO PERFORM CSP COMMAND    **
//*********************************************************************
//IMPORT EXEC PGM=BC1PDCGB,COND=(0,LT),MAXRC=8
//STEPLIB DD DISP=SHR,DSN=&CRS..SELALMD
//        DD DISP=SHR,DSN=&CSP..SEZELMD
//*       DD DISP=SHR,DSN=EDC.SEDCLINK
//*       DD DISP=SHR,DSN=PLI.SIBMLINK
//        DD DISP=SHR,DSN=&COBLIB.
//        DD DISP=SHR,DSN=&CONLIB.
//*********************************************************************
//* CSP MESSAGE AND TUTORIAL FILES                                    **
//*********************************************************************
//EZECOMM  DD DISP=SHR,DSN=&CSPVSAM..&NLS..EZECOMM
//*-----------------------------------------------------------
//* THE FOLLOWING MESSAGE FILES SHOULD BE PRE-ALLOCATED
//*-----------------------------------------------------------
//*EZEDMSG DD DISP=SHR,DSN=&CSPVSAM..FZEMSG
//*EZEEMSG DD DISP=SHR,DSN=&CSPVSAM..&NLS..EZEMSG
//*EZETES  DD DISP=SHR,DSN=&CSPVSAM..&NLS..FZETUTOR
//*EZEMAP  DD DISP=SHR,DSN=&CSPVSAM..&NLS..FZEMAPDS
```

```
//*********************************************************************
//* STAGE MEMBER SPECIFICATION LIBRARY (R/W MSL)                      **
//*********************************************************************
//USR1MSL DD DISP=SHR,DSN=&USR1MSL.
//USR2MSL DD DISP=SHR,DSN=&USR2MSL.
//*********************************************************************
//* CSP TEMPORARY WORK FILE
//*********************************************************************
//EZEWORK DD DISP=SHR,DSN=&CSPUSER..EZEWORK
//EZEWRK1 DD UNIT=&WORKDISK.,SPACE=(CYL,(2,1)),
//           DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//*********************************************************************
//* CSP ESF SERIAL FILE                                               **
//*********************************************************************
//TEMPSRC DD DISP=(OLD,DELETE),DSN=&&TEMPSRC
//*********************************************************************
//* CSP INVOCATION PARAMETER                                          **
//*********************************************************************
//NDVRIPT DD *
M=USR1MSL ROMSL=USR2MSL CMDIN=EZECIN CMDOUT=EZECOUT
/*
//*********************************************************************
//* CSP LOG AND LISTINGS FILES                                        **
//*********************************************************************
//EZECOUT  DD DISP=(OLD,PASS),DSN=&&LOGLST
//EZEPRINT DD DISP=(OLD,PASS),DSN=&&CMDLST
//SYSPRINT DD SYSOUT=*
//*********************************************************************
//* CSP COMMAND INPUT                                                **
//*********************************************************************
//EZECIN DD *
IMPORT SERIAL(TEMPSRC) REPLACE(Y);
 /*
//*********************************************************************
//* STEP5 - STORE THE LISTINGS FROM THE PREVIOUS STEPS                **
//*********************************************************************
//STRLST EXEC PGM=CONLIST,COND=EVEN,MAXRC=0,PARM=STORE
//C1LLIBO  DD DSN=&LISTLIB.,DISP=SHR
//C1BANNER DD UNIT=&WORKDISK.,SPACE=(TRK,(1,1)),
//            DCB=(RECFM=FBA,LRECL=121,BLKSIZE=6171)
//LIST01   DD DISP=(OLD,DELETE),DSN=&&AMSLST
//LIST02   DD DISP=(OLD,DELETE),DSN=&&LOGLST
//LIST03   DD DISP=(OLD,DELETE),DSN=&&CMDLST
//*
```

Appendix B: Unsupported Parameters

This section contains the following topics:

General Restrictions

As discussed previously, CA Endevor SCM processors are written using standard OS JCL syntax--with a few restrictions. Most JCL parameters are supported in CA Endevor SCM processor statements.

Selected JCL parameters, and the JCL SET statement, are unsupported by CA Endevor SCM and, if used, are ignored by the system or cause as an error. A message is returned when CA Endevor SCM encounters one of these parameters:

- If the parameter is ignored, the message reads:

  `C1X0205W STMT statement-no INVALID--KEYWORD keyword IS NOT SUPPORTED`

- If the parameter causes an error, the message reads:

  `C1X0202E STMT statement-no INVALID--STATEMENT`

The following JCL keyword parameters and the JCL SET statement and INCLUDE statement are not supported by CA Endevor SCM processors.

Except for the SET statement, which can appear anywhere in JCL, the invalid parameters are shown in the following lists with the statement type (EXEC= or DD=) in which they are normally used.

SET Statement

The SET command cannot be used in a processor.

INCLUDE Statement

The JCL INCLUDE statement cannot be used in a processor. However, CA Panvalet and CA Librarian include member parameters can be used.

EXEC Statement Parameters

CA Endevor SCM processors support the EXEC statement, but they do not support backward reference (REFERBACK), or these EXEC statement parameters:

```
ACCT
ADDRSPC
DYNAMNBR
PERFORM
PROC
RD
REGION
TIME
```

DD Statement Parameters

CA Endevor SCM processors support the DD statement, but they do not support backward reference (REFERBACK), or these DD statement parameters:

```
ACCODE
AMP
BURST
CHARS
CHKPT
CNTL
DYNAM
SUBSYS=PANV
```

DCB Subparameters

CA Endevor SCM processors support the DCB parameter. The following DCB subparameters that are not supported.

- BUFIN
- BUFMAX
- BUFOFF

- BUFOUT
- BUFSIZE
- CODE
- CPRI
- CYLOFL
- FRID
- FUNC
- GNCP
- INTVL
- IPLTXID
- MODE
- NTM
- PCI
- PRTSP
- RESERVE
- RKP
- STACK
- THRESH
- TRTCH

DDNAME Subparameters

CA Endevor SCM processors support the DDNAME parameter. The following DDNAME subparameters are not supported.

- DSID
- FLASH
- MODIFY
- MSVGP
- OUTPUT
- QNAME
- REFDD
- SPLIT

- SUBALLOC
- DDNAME

Index

D

E

F

G

I

J

K

L

M

N

O

P

R

S

T

U

W