

AllFusion™ Endeavor® Change Manager

Extended Processors Guide
4.0



Computer Associates™

This documentation and related computer software program (hereinafter referred to as the "Documentation") is for the end user's informational purposes only and is subject to change or withdrawal by Computer Associates International, Inc. ("CA") at any time.

This documentation may not be copied, transferred, reproduced, disclosed or duplicated, in whole or in part, without the prior written consent of CA. This documentation is proprietary information of CA and protected by the copyright laws of the United States and international treaties.

Notwithstanding the foregoing, licensed users may print a reasonable number of copies of this documentation for their own internal use, provided that all CA copyright notices and legends are affixed to each reproduced copy. Only authorized employees, consultants, or agents of the user who are bound by the confidentiality provisions of the license for the software are permitted to have access to such copies.

This right to print copies is limited to the period during which the license for the product remains in full force and effect. Should the license terminate for any reason, it shall be the user's responsibility to return to CA the reproduced copies or to certify to CA that same have been destroyed.

To the extent permitted by applicable law, CA provides this documentation "as is" without warranty of any kind, including without limitation, any implied warranties of merchantability, fitness for a particular purpose or noninfringement. In no event will CA be liable to the end user or any third party for any loss or damage, direct or indirect, from the use of this documentation, including without limitation, lost profits, business interruption, goodwill, or lost data, even if CA is expressly advised of such loss or damage.

The use of any product referenced in this documentation and this documentation is governed by the end user's applicable license agreement.

The manufacturer of this documentation is Computer Associates International, Inc.

Provided with "Restricted Rights" as set forth in 48 C.F.R. Section 12.212, 48 C.F.R. Sections 52.227-19(c)(1) and (2) or DFARS Section 252.227-7013(c)(1)(ii) or applicable successor provisions.

First Edition, December 2002

© 2002 Computer Associates International, Inc.
All rights reserved.

All trademarks, trade names, service marks, and logos referenced herein belong to their respective companies.

Contents

Chapter 1. Introduction	1-1
1.1 Processor Overview	1-2
1.2 Processor Group Overview	1-3
1.3 Processors and Element Types	1-4
1.4 Processors Invoked by Endeavor Actions	1-5
1.5 Documentation Overview	1-6
1.6 Name Masking	1-7
1.6.1 Usage	1-7
1.7 Syntax Conventions	1-9
1.7.1 Sample Syntax Diagram	1-12
1.7.2 Syntax Diagram Explanation	1-12
1.7.3 General Coding Information	1-14
1.7.3.1 Valid Characters	1-14
1.7.3.2 Incompatible Commands and Clauses	1-15
1.7.3.3 Ending A Statement	1-15
1.7.3.4 SCL Parsing Information	1-15
Chapter 2. Writing Processors	2-1
2.1 Overview	2-2
2.2 Suggested Processor Naming Conventions	2-3
2.3 Processor Features	2-4
2.3.1 Reserved Words and Labels	2-4
2.4 Processor Keywords	2-5
2.4.1 FOOTPRNT	2-5
2.4.2 MAXRC	2-6
MAXRC Scenario	2-7
MAXRC Example	2-7
2.4.3 EXECIF	2-7
2.4.4 BACKOUT	2-9
2.4.5 MONITOR	2-9
2.5 Symbolic Parameters	2-10
2.5.1 Using the Ampersand (&)	2-10
2.5.2 General Guidelines	2-11
2.5.3 User Symbolics	2-11
2.5.4 Site Symbolics	2-12
2.5.5 Using Site Symbolics in Processors	2-12
2.5.6 Defining Site Symbolics	2-13
2.5.7 Endeavor Symbolics	2-14
2.5.7.1 Substringing	2-19

2.5.8 In-Stream Data	2-21
2.6 Controlling Processor Flow	2-22
2.6.1 IF-THEN-ELSE statement	2-22
2.6.1.1 RC	2-28
2.6.1.2 ABENDCC	2-28
2.6.1.3 ABEND	2-28
2.6.1.4 ¬ABEND	2-29
2.6.1.5 RUN	2-29
2.6.1.6 ¬RUN	2-29
2.6.2 The ENDEVOR IF-THEN-ELSE Trace Facility	2-29
Sample output	2-29
2.7 Authorizing a Non-Endevor Program	2-32
Chapter 3. Processor Utilities	3-1
3.1 Overview	3-2
3.1.1 Utilities Available	3-2
3.2 BC1PDSIN Utility	3-4
3.2.1 Sample JCL	3-4
3.3 BC1PTMP0 Utility	3-5
3.3.1 Sample JCL	3-5
3.3.1.1 Parameters	3-6
3.3.2 Return Codes	3-6
3.3.2.1 Other Return Codes	3-6
3.4 BC1PXFPI Utility	3-7
3.5 BSTCOPY Utility	3-8
3.5.1 Supported Copy Functions	3-8
3.5.1.1 BSTCOPY Syntax - Literal Interpretation	3-8
3.5.1.2 BSTCOPY Syntax - Alternate Interpretation	3-9
3.5.2 Unsupported Functions	3-9
3.5.3 BSTCOPY and OVERLAY Modules	3-10
3.5.4 SYSPRINT DCB Information	3-11
3.6 C1BM3000 Utility	3-12
3.6.1 Do Not Use...	3-12
3.6.2 Sample JCL	3-12
3.7 C1PRMGEN Utility	3-14
3.7.1 Sample JCL	3-14
3.7.1.1 Parameters	3-15
3.8 CONAPI Utility	3-16
3.8.1 Sample JCL	3-16
3.9 CONDELE Utility	3-17
3.9.1 Sample JCL	3-17
3.9.1.1 Parameters	3-17
3.10 CONLIST Utility	3-18
3.10.1 Banner Pages	3-19
3.10.2 STORE Option	3-20
3.10.3 PRINT Option	3-21
3.10.4 PRTMBR (Print Member) Option	3-21
3.10.5 COPY Option	3-22
3.10.6 DELETE Option	3-22
3.10.7 Guidelines When Creating Listings	3-23
3.11 CONRELE Utility	3-26

3.11.1	CONRELE Utility Commands	3-26
3.11.2	RELATE ELEMENT Command Syntax	3-26
3.11.2.1	RELate ELEment Syntax	3-27
	Parameters	3-27
3.11.3	RELATE MEMBER Command syntax	3-28
3.11.3.1	RELate MEMber Syntax	3-28
	Parameters	3-28
3.11.4	RELATE OBJECT Command Syntax	3-28
3.11.4.1	RELate OBJect Syntax	3-28
	Parameters	3-29
3.11.5	RELATE COMMENT Command Syntax	3-29
3.11.5.1	RELate COMment Syntax	3-29
	Parameters	3-29
3.11.6	SET ERROR RETURN CODE Command Syntax	3-29
3.11.6.1	SET ERRor RETurn CODE Syntax	3-29
	Parameters	3-29
3.11.7	Example of CONRELE Syntax	3-29
3.12	CONSCAN Utility	3-32
3.12.1	CONSCAN Parameter Data Set	3-32
3.12.2	PARMSCAN Parameter Statements	3-32
3.12.3	Excluding Source Data	3-33
	Exclusion Group Syntax	3-33
	Exclusion Group Examples	3-34
3.12.4	Selecting Source Data	3-34
	Selection Group Syntax	3-35
3.12.4.1	Generated CONRELE Control Statements	3-38
	Selection Group Examples	3-38
3.12.5	Scan Rule Processing	3-40
3.12.5.1	Sample CONSCAN Utility Processor	3-41
3.12.6	Error Messages	3-42
3.13	CONWRITE Utility	3-43
3.13.1	Writing Component List Data to an External Location	3-43
3.13.1.1	Component List Data	3-43
3.13.1.2	Output Format	3-44
3.13.2	Writing Elements to an External Location	3-44
3.13.3	Standard Form of CONWRITE	3-44
3.13.4	Extended Form of CONWRITE	3-45
3.13.5	Command Syntax for the CONWRITE Utility	3-45
3.13.5.1	CONWRITE Syntax	3-45
	Parameters	3-46
3.13.6	Using CONWRITE to Expand INCLUDEs	3-47
3.13.6.1	The PARM=EXPINCL() Clause	3-48
3.13.7	Writing Exit Programs to Use CONWRITE Input	3-49
Chapter 4. Classifying and Managing Processors		4-1
4.1	Overview	4-2
4.2	Classifying Processors	4-3
4.3	Managing Processors	4-5
4.3.1	Procedure: Implementing Processors	4-5
4.3.2	Procedure: Maintaining Processors	4-6

4.3.3	Where Endeavor Looks for Processors	4-7
Chapter 5. Processor Groups		
5.1	Processor Group Overview	5-2
5.1.1	Three Types	5-2
5.1.2	Suggested Naming Conventions for Processor Groups	5-2
5.1.2.1	Example	5-3
5.1.3	User-Defined Symbolics	5-4
5.2	Working with Processor Group Information	5-5
5.2.1	From the Environment Options Menu	5-5
5.2.2	From the Type Definition Panel	5-6
5.3	Working with Processor Group Symbolics	5-7
5.3.1.1	Example	5-8
5.3.2	Displaying Processors	5-10
5.4	Processor Group Selection List	5-11
5.5	Processor Group Definition Panel	5-12
5.5.1	Identification Fields	5-13
5.5.2	Output Management Information Fields	5-14
5.5.2.1	Move/Transfer Processor Selection	5-14
5.5.2.2	Option Fields	5-15
5.5.2.3	Processor Identification Fields	5-15
5.5.2.4	Foreground Execution Fields	5-16
5.6	Processor Group Symbolics Panel	5-17
5.6.1	Identification Fields	5-17
5.6.2	Symbolic Identification Fields	5-17
5.7	Processor Display Panel	5-19
Appendix A. Sample Processors		
A.1	Sample Processor Overview	A-2
A.2	Converting PROCs to Processors	A-3
A.3	Generate Processors	A-5
A.3.1	GCIIDBL	A-6
A.3.2	GCIINBL	A-11
A.3.3	GLNKNBL	A-14
A.3.4	GASMNBL	A-16
A.3.5	LOADONLY	A-19
A.4	Delete Processors	A-21
A.4.1	DLODDNL	A-22
A.4.2	DLODNNL	A-23
A.5	Move Processors	A-24
A.5.1	MLODDNL	A-25
A.5.2	MLODNNL	A-28
A.6	Other Processors	A-30
Appendix B. Unsupported Parameters		
B.1	General Restrictions	B-2
B.2	EXEC Statement Parameters	B-3
B.3	DD Statement Parameters	B-4
B.4	DCB Subparameters	B-5
B.5	DDNAME Subparameters	B-6

Index X-1

Chapter 1. Introduction

1.1 Processor Overview

Endevor allows you to create and maintain processors. Coded using standard JCL, processors instruct Endevor 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 Endevor. It supplies functionality beyond writing or deleting a member in an output library.

1.2 Processor Group Overview

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

- 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. (For more information, see 2.5, “Symbolic Parameters” on page 2-10.)

1.3 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, Endeavor automatically assigns the default processor group to the element. You can override this assignment on the Add/Update Elements panel. (For more information about defining types, see the *Administration Guide*. For more information about adding elements, see the *User Guide*.)

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

Note: Since Endeavor 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.

1.4 Processors Invoked by Endeavor Actions

Endeavor actions invoke processors as indicated in the following table.

This action	Invokes this processor	And provides these options
Add	Generate	Processor group Bypass generate processor
Archive	Delete	Bypass element delete
Delete	Delete	Bypass element delete
Generate	Generate (and delete if processor group changes)	Processor group
Move	Move (default) or generate, and delete	Bypass element delete
Restore	Generate	Processor group Bypass generate processor
Transfer	Generate (default) or move, and delete	Processor group Bypass generate processor Bypass delete processor Bypass element delete
Update	Generate	Processor group Bypass generate processor

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

For information about processor groups, see Chapter 5, “Processor Groups” in this manual. For information about how Endeavor determines which processor group to use for an action, see the section “Action Processing,” in the *User Guide*.

1.5 Documentation Overview

This manual is part of a comprehensive documentation set that fully describes the features and functions of Endeavor and explains how to perform everyday tasks. For a complete list of Endeavor manuals, see the PDF Table of Contents file in the PDF directory, or the Bookmanager Bookshelf file in the Books directory.

The following section describes product conventions.

1.6 Name Masking

A name mask allows you to specify all names, or all names beginning with a particular string, to be considered when performing an action.

Name masks are valid on:

- Element names
- System, subsystem, and type names within FROM clauses
- Report syntax
- ISPF panels
- API requests

Name masks are not valid on:

- Environment names
- Element names in the following situations:
 - When entering a LEVel in a statement
 - When using the MEMber clause with a particular action
 - When building a package

1.6.1 Usage

There are three ways to mask names: by using the wildcard character (*), by using the placeholder character (%), and by using both together.

The wildcard (*) can be used in one of two ways to specify external file names:

- When coded as the only character of a search string, Endeavor returns all members of the search field. For example, if you coded the statement `ADD ELEMENT *`, all elements would be added.
- When coded as the last character of a search string, Endeavor returns all members of the search field beginning with the characters in the search string preceding the wildcard. For example, the statement `ADD ELEMENT UPD*` would add all elements beginning with "UPD", such as `UPDATED` or `UPDATE`.

Note: You cannot use more than one wildcard in a string. The statement `ADD ELEMENT U*PD*` would result in an error.

The placeholder (%) can also be used in one of two ways:

- When coded as the last character in a string, Endeavor returns all members of the search field, beginning with the characters in the search string preceding the placeholder, but which have no more characters than were coded in the search string. If you coded the statement `ADD ELEMENT UPD%`, only those elements

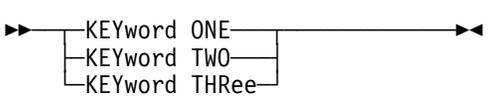
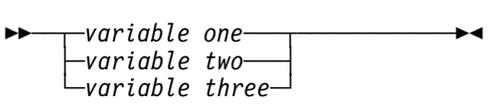
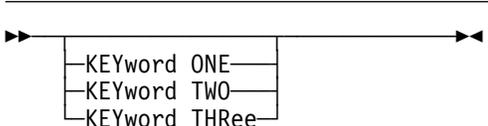
with four-character-long names beginning with "UPD" (UPD1 or UPDA, for example) would be added.

- It is also possible to use the placeholder multiple times in a single search string. The statement `ADD ELEMENT U%PD%` would return all elements with five-character-long names that have U as the first character, and PD third and fourth.

The wildcard and the placeholder can be used together, provided that the wildcard appears only at the end of the search string and is used only once. An example of a statement using both the wildcard and the placeholder is `ADD ELEMENT U%D*`. This statement would add elements with names of any length that have U as the first character and D as the third.

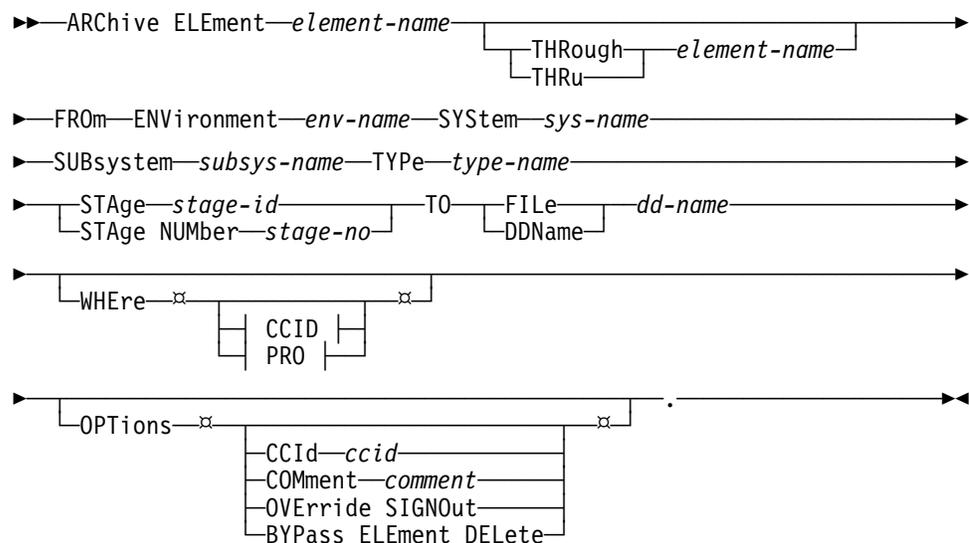
1.7 Syntax Conventions

Endevor uses the IBM standard for representing syntax. The following table explains the syntax conventions:

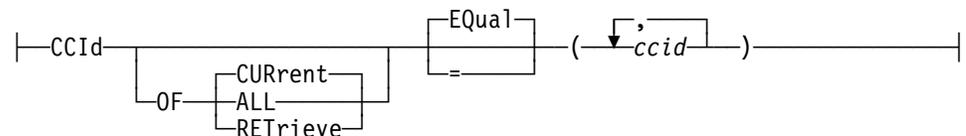
Syntax	Explanation
	Represents the beginning of a syntax statement.
	Represents the end of a syntax statement.
	Represents the continuation of a syntax statement to the following line.
	Represents the continuation of a syntax statement from the preceding line.
	Represents a required keyword. Only the uppercase letters are necessary.
	Represents a required user-defined variable.
	Represents an optional keyword. Optional keywords appear below the syntax line. If coded, only the uppercase letters are necessary.
	Represents an optional user-defined variable. Optional variables appear below the syntax line.
	Represents a choice of required, mutually exclusive keywords. You must choose one and only one keyword.
	Represents a choice of required, mutually exclusive, user-defined variables. You must choose one and only one variable.
	Represents a choice of optional, mutually exclusive keywords. Optional keywords appear below the syntax line.

Syntax	Explanation
	<p>Represents a choice of optional, mutually exclusive, user-defined variables. Optional variables appear below the syntax line.</p>
	<p>Represents a choice of optional keywords. The stars (x) indicate that the keywords are not mutually exclusive. Only code the keyword once.</p>
	<p>Represents a choice of optional user-defined variables. The stars (x) indicate that the variables are not mutually exclusive. Only code the variable once.</p>
	<p>Represents a choice of required, mutually exclusive keywords, one of which is the default. In this example, KEYword ONE is the default keyword because it appears above the syntax line.</p>
	<p>Represents a choice of required, mutually exclusive, user-defined variables, one of which is the default. In this example, <i>variable one</i> is the default variable because it appears above the syntax line.</p>
	<p>Represents a choice of optional, mutually exclusive keywords, one of which is the default. In this example, KEYword ONE is the default keyword because it appears above the syntax line.</p>
	<p>Represents a choice of optional, mutually exclusive, user-defined variables, one of which is the default. In this example, <i>variable one</i> is the default variable because it appears above the syntax line.</p>
	<p>Represents a required variable that can be repeated. Separate each occurrence with a comma and enclose any and all variables in a single set of parenthesis.</p>

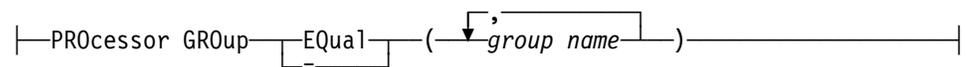
1.7.1 Sample Syntax Diagram



CCID:



PRO:



1.7.2 Syntax Diagram Explanation

Syntax	Explanation
ARChive ELEment <i>element-name</i>	The keyword ARChive ELEment appears on the main line, indicating that it is required. The variable <i>element-name</i> , also on the main line, must be coded.
THROUGH / THRu <i>element-name</i>	The keywords THROUGH and THRu appear below the main line, indicating that they are optional. They are also mutually exclusive.
FRom ENVironment ... TYPe <i>type-name</i>	Each keyword and variable in this segment appear on the main line, indicating that they are required.
STAge <i>stage-id</i> / STAge NUMBER <i>stage-no</i>	The keywords STAge and STAge NUMBER appear on and below the main line, indicating that they are required, mutually exclusive keywords.

Syntax	Explanation
TO ... <i>dd-name</i>	The keyword TO appears on the main line, indicating that it is required. The keywords FILE and DDName appear on and below the main line, indicating that they are required, mutually exclusive keywords. The variable <i>dd-name</i> also appears on the main line, indicating that it is required.
WHERe clause	This clause appears below the main line, indicating that it is optional. The keyword WHERe appears on the main line of the clause, indicating that it is required. CCID and PRO are syntax fragments that appear below the main line, indicating that they are optional. The stars (⌘) indicate that they are not mutually exclusive. For details on the CCID and PRO fragments, see the bottom of this table.
OPTion clause	This clause appears below the main line, indicating that it is optional. The keyword OPTion appears on the main line of the clause, indicating that it is required. The keywords CCId, COMment, OVErride SIGNOut, and BYPass ELEment DELEte all appear below the main line, indicating that they are optional. The stars (⌘) indicate that they are not mutually exclusive.
CCID fragment	<p>The keyword CCId appears on the main line, indicating that it is required. The OF clause appears below the main line, indicating that it is optional. If you code this clause, you must code the keyword OF, as it appears on the main line of the clause. CURrent, ALL, and RETrieve appear above, on, and below the main line of the clause, indicating that they are required, mutually exclusive keywords. CURrent appears above the main line, indicating that it is the default. If you code the keyword OF, you must choose one and only one of the keywords.</p> <p>The keywords EQual and = appear above and below the main line, indicating that they are optional, mutually exclusive keywords. EQual appears above the main line, indicating that it is the default. You can include only one. The variable <i>ccid</i> appears on the main line, indicating that it is required. The arrow indicates that you can repeat this variable, separating each instance with a comma. Enclose any and all variables in a single set of parenthesis.</p>

Syntax	Explanation
PRO fragment	The keyword PROcessor GROUp appears on the main line, indicating that it is required. The keywords EQual and = appear on and below the main line, indicating that they are required, mutually exclusive keywords. You must include one. The variable <i>group name</i> appears on the main line, indicating that it is required. The arrow indicates that you can repeat this variable, separating each instance with a comma. Enclose any and all variables in a single set of parenthesis.

1.7.3 General Coding Information

In coding syntax, you must adhere to certain rules and guidelines regarding valid characters, incompatible commands and clauses, and ending statements. In addition, knowing how the SCL parser processes syntax will help you resolve errors and undesired results. The following sections outline these rules and guidelines.

1.7.3.1 Valid Characters

The following characters are allowed when coding syntax:

- Uppercase letters
- Lowercase letters
- Numbers
- National characters
- Hyphen (-)
- Underscore (_)

The following characters are allowed when coding syntax, but must be enclosed in either single (') or double (") quotation marks:

- Space
- Tab
- New line
- Carriage return
- Comma (,)
- Period (.)
- Equal sign (=)
- Greater than sign (>)
- Less than sign (<)

- Parenthesis ()
- Single quotation marks
- Double quotation marks

A string containing single quotation marks must be enclosed in double quotation marks. A string containing double quotation marks must be enclosed in single quotation marks.

To remove information from an existing field in the database, enclose a blank space in single or double quotation marks. For example, the following statement removes the default CCID for user TCS:

```
DEFINE USER TCS  
DEFAULT CCID " ".
```

The characters "*" and "%" are reserved for name masking. See section 1.6, "Name Masking" on page 1-7 for more information.

1.7.3.2 Incompatible Commands and Clauses

The following commands and clauses are mutually exclusive:

- THROUGH and MEMBER clauses within any action except LIST
- Endeavor location information (environment, system, subsystem, type, and stage) and data set names (DSName)
- File names (DDName) and data set names (DSName)
- The stage id (STAGE / STAGE ID) and the stage number (STAGE NUMBER)
- The SET TO Endeavor location information and the SET TO MEMBER clause

1.7.3.3 Ending A Statement

You must enter a period at the end of each statement. If no period is found, you receive an error message and the job terminates.

1.7.3.4 SCL Parsing Information

- The SCL parser does not look for information in columns 73-80 of the input. Therefore, be sure that all relevant information is coded in columns 1-72.
- The SCL parser does not catch duplicate clauses coded for an SCL request. If you code the same clause twice, SCL uses the Boolean "AND" to combine the clauses. If the result is invalid, you receive an error message.
- If you enter an asterisk (*) in column 1, the remainder of the line is considered a comment by the SCL parser and is ignored during processing.
- Any value found to the right of the period terminating the SCL statement is considered a comment by the SCL parser and is ignored during processing.

Chapter 2. Writing Processors

2.1 Overview

Processors are coded using standard JCL syntax and are converted to an Endeavor 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.)

Endeavor provides several keywords, symbolic parameters, and utilities for use in coding processors. While Endeavor 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 an Endeavor processor step are deallocated 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 deallocated at processor step termination.

For an explanation and illustration of the various types and uses of processors, see Appendix A, "Sample Processors."

2.2 Suggested Processor Naming Conventions

Processor names can have up to eight characters. The abbreviations below do not represent a complete list, and are offered as guidelines only.

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
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

2.3 Processor Features

For the most part, Endeavor processors are written using standard OS JCL syntax. Endeavor supports *most* JCL parameters, for a complete list of unsupported JCL parameters, see Appendix B, “Unsupported Parameters.”

Endeavor provides additional features and capabilities within processors:

- Keywords to tailor your processors
- Endeavor, User- and Site-defined symbolics
- Component monitoring (for Endeavor ACM clients)
- Endeavor utilities (described in Chapter 3, “Processor Utilities”)
- Support for in-stream data

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

2.3.1 Reserved Words and Labels

When writing a processor avoid using these reserved names:

CC

DDA

DDB

DDB

JCL

PGM

SYM

SIN

PRITE

FLD

Otherwise, your processor will abend with this message:

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

2.4 Processor Keywords

There are five keywords that are specific to Endeavor processor statements:

- **FOOTPRNT** — Causes Endeavor 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.

2.4.1 FOOTPRNT

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

- **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 an Endeavor action, such as RETRIEVE, to a specific PDS (or Endeavor LIB) 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-Endeavor 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, as shown below.

```
//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, Endeavor 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 Endeavor 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 an Endeavor error. To override this default, the VFY_FT_FAILURES feature must be activated in the Endeavor Options table (ENCOPTBL). With this option, the footprint verification return code becomes a step return code instead of an Endeavor return code. Refer to the *Installation Guide*, Appendix F, "Endeavor Optional Feature Table" for more information.

```
//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 Endeavor to bypass footprint creation.

2.4.2 MAXRC

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=nn

nn

The highest acceptable OS return code for the step

If the return code exceeds the value specified:

- The Endeavor return code is set to 12.
- A processor-failed flag is set for the element within Endeavor.
- All remaining actions are executed.
- The processor's remaining steps 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, Endeavor 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; see the *SCL Reference Guide* for more information.

MAXRC Scenario: There is a processor containing three steps:

1. Compile
2. Link-edit
3. CONLIST

You would code the:

- 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 Endeavor 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.

2.4.3 EXECIF

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, an Endeavor, user, or site symbolic.

operator

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

- 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, an Endeavor symbolic, or a user symbolic.

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. For more information, see the chapter “Load Module Support,” in the *Utilities Guide*.

2.4.4 BACKOUT

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

When you use packages, Endeavor, 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)
```

2.4.5 MONITOR

Component monitoring is a feature of Endeavor's Automated Configuration Manager (ACM). You can use the keyword:

- **MONITOR** — monitors selected library data sets for component relationships.

For more information, see the *Automated Configuration Option Guide*.

2.5 Symbolic Parameters

Symbolic parameters provide a powerful means of writing processors for your environment. The processors treat the symbolic parameters 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. And you can write one processor to perform many functions.

There are three types of symbolics within Endeavor:

- User Symbolics
- Site Symbolics
- Endeavor Symbolics

In addition, Endeavor provides capabilities not available with standard JCL. For example:

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

These are described in the following sections.

2.5.1 Using the Ampersand (&)

Symbolic parameters can be used to tailor processors. Symbolics are recognized by an ampersand (&) as the first character of the value. Symbolics can be used for any value that appears on the right side of a keyword parameter, for example:

- A data set name, as **DSN=&dsname..LOADLIB**

To use the symbolic *&dsname* to define a library:

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

If CA is the value substituted for *&dsname*, the statement reads:

```
DSN = CA.LOADLIB
```

Note: When using a symbolic for a dataset, do not include the member name as part of the symbolic or you will get an allocation error. Use two symbolics; for example:

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

- An execution parameter, as **PARM='&execparm'**

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

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

2.5.2 General Guidelines

When using symbolics in Endeavor processors, remember that:

- The first character of a site symbolic must begin with an &.
 - When referencing a site symbolic, the ampersand (&) is appended to the beginning of the site symbolic name.
 - User-defined symbolic parameters (see 2.5.3, “User Symbolics”) must be defined in PROC statements at the beginning of the processor.
 - You can specify symbolics only on the right side of keyword parameters.
 - One symbolic 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 symbolic &TRAK.
- Note:** In this case, (1,1) is considered to be two subparameters. Thus, defining a single symbolic to be '1,1' and substituting that symbolic in the SPACE statement results in an allocation error.
- Most symbolics are not validated until execution time. Therefore, ensure symbolics specified as defaults and overrides are defined in the processor group definition.

2.5.3 User Symbolics

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

Processors treat user symbolics 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 symbolic default values initially established in the PROC statement. The user symbolic override function is part of the processor group definition procedure. For more information, see 5.5, “Processor Group Definition Panel” on page 5-12 (in the discussion of processor groups).

A sample PROC statement is provided below.

```

/**
//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.

2.5.4 Site Symbolics

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

When Endeavor is initialized, the site symbolics are placed into memory and when Endeavor is terminated, the site symbolic storage is released.

To implement site symbolics, the symbolic 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 CIDEFLTS table with the name of the site symbolics table. These actions are described in the following sections.

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

2.5.5 Using Site Symbolics in Processors

Site symbolic names are always preceded by an ampersand (&) before the name. The site symbolic **#VENDORLIB** looks like this in a processor:

&#VENDORLIB

Example: In your site, these site symbolics are used:

Symbol	Value
#ENDPRE	CA
#MIDDLE	'QA&&C1SY..S1'
#SITESYM	'QA&&#ENDPRE..S1'

This is how they 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..&&C1TYPE
```

These are the values at runtime:

```
//IN      DD DISP=SHR,DSN=CA.R40.SRCLIB
//OUT     DD DISP=SHR,DSN=CA.QAc1sy..S1.OBJLIB
//DUMMY   DD DISP=SHR,DSN=CA.QAc1sy..S1.c1type
```

2.5.6 Defining Site Symbolics

Before defining the site symbolics, remember:

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

```
&#VENDORLB
```

Steps for defining a site symbolics table:

1. Define the symbolic and its data value — This is the syntax for the site symbolics table:

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

symbolname

The symbol name must begin with the # character and is 1 to 11 characters in length. The # indicates that the symbol is defined in the site-defined symbolics table.

symbolvalue

The data value associated with the site symbolic 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 symbolic, Endeavor treats it as a null variable.

2. Assemble and link-edit the symbols table — The BC1JSYMT JCL member located in *iprfx.igual*. creates the symbolics table.
3. Update C1DEFLT5 to associate the site symbolics to the Endeavor environment.

By using the Site Information panel, you can determine if site symbolics are defined in your Endeavor Environment.

The Site Information from C1DEFLT5 panel indicates if site symbolics are installed. The name of the site symbolics file is displayed in the SYMBOL Tbl field when symbolics are used.

```

----- Site Information from C1DEFLT5 -----
Command ==>

Customer Name..... SUPPORT 4.0 BETA/2
----- Function Controls -----
Site ID..... 0      Access Table..... BC1TNEQU      - Options -
Release..... B4000C  SMF Record Number. 000      ACM..... Y
Environments..... 2      Library System... PV      DB2..... N
Userid Start..... 1      Library Program...      QuickEdit Y
Userid Length..... 7      VIO Unit..... SYSDA      ELINK.... N
Batch ID..... 0      Work Unit..... SYSDA      ESI..... Y
SPFEDIT QNAME..... SPFEDIT  Work Volser.....      INFO..... N
SYSIEWL QNAME..... SYSIEWLP  Lines per Page.... 60      LIBENV... Y
Authorized Tables. REQUIRED  MODHLI.....      NETMAN... N
Gen in place/SO... N      Signout on fetch.. Y      PDM..... Y
CA-LSERV JRNL SBS.      ELINK XLTE TBL....      PROC..... Y
PITR Journal Grp..      Mixed Format..... CCID COMMENT DESCRIPTION
SYMBOLICS Table.  ESYMBOL

(Press Enter for Next Panel)

```

To display your site symbolics and the associated values, from the AllFusion Endeavor Primary Options Menu select the option for Display.

Select **S** to display the site symbolics and values defined in your Endeavor environment.

```

DISPLAY ----- SYMBOL TABLE: ESYMBOLS ----- Row 1 to 6 of 6
COMMAND ==>                                     SCROLL ==> CSR

SYMBOL      VALUE
-----
#ENDPRE     CA
#MIDDLE     'QA&&C1SY..S1'
#PERMBASE1  /u/endeavor/sampletest/admin/base
#SITESYM    'QA&&#ENDPRE..S1'
***** Bottom of data *****

```

2.5.7 Endeavor Symbolics

Endeavor symbolic names are reserved, and represent values specific to Endeavor, such as environment and element name. Endeavor determines the value to be substituted at execution time. For example, if you code the Endeavor symbolic **&C1STGID1** in a processor and add an element, Endeavor substitutes the appropriate Stage 1 ID when the processor executes. For compatibility with previous releases, Endeavor symbolics can begin with one or two ampersands: **&** or **&&**. When Endeavor encounters an Endeavor symbolic immediately followed by a period, it takes one of the following actions:

- If the symbolic begins with a single ampersand (**&**), Endeavor does not include the period in the resolved statement.
- If the symbolic begins with a double ampersand (**&&**), Endeavor retains the period in the resolved statement.

For example, assume the symbolic `&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
```

Endevor 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 an Endevor symbolic, you must specify two consecutive periods if your symbolic uses a single ampersand, and only one period if your symbolic uses double ampersands.

Endevor symbolic parameters are listed and explained below.

Note: The symbolics in italics in the table below should be used only for the source location of MOVE or TRANSFER actions made using the move processor. For all actions that execute a generate or a delete processor, Endevor assigns these symbolics the same values as those assigned to the corresponding symbolic beginning with `&C1`.

Symbolic (alias)	Length	Replaced in processor by the
<code>&C1ACTION</code>	8	Action currently being executed. Note: If action is ADD with update option turned on and element exists on store, the <code>C1ACTION</code> resolves to UPDATE.
<code>&C1ADDMMYY</code>	7	Date in the format DDMMYY. For example: 15JUN01
<code>&C1ADDMMYY</code>	6	Date in the format DDMMYY. For example: 150601
<code>&C1ADDMMYYYY</code>	8	Date in the format DDMMYYYY. For example: 15062001
<code>&C1ADD</code>	2	Date in the format DD. For example: 15
<code>&C1AMM</code>	2	Date in the format MM. For example: 06
<code>&C1AMMM</code>	3	Date in the format MMM. For example: JUN
<code>&C1AYY</code>	2	Date in the format YY. For example: 01
<code>&C1AYYYY</code>	4	Date in the format YYYY. For example: 2001
<code>&C1AHHMMSS</code>	6	Time in the format HHMMSS. For example: 125930

Symbolic (alias)	Length	Replaced in processor by the
&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	Base/image library for the type specified in the action.
&C1CCID	12	Last-specified CCID for the element.
&C1COMMENT (&C1COM)	40	Comment associated with the action being executed.
&C1ELEMENT	8	1-8 character name of the element being processed. Names longer than eight characters are truncated.
&C1ELMCHG	1	1 character used to indicate whether the source manager detected changes to this element; Y-Yes, N-No.
&C1ELMNT10	10	1-10 character name of the element being processed. Generally used to assign a name to an AllFusion™ CA-Panvalet member.
&C1ELMNT255 1	255	1-255 character name for element names greater than 10 characters and mixed/lower case names less than 10 characters.
&C1ELTYPE (&C1TY)	8	Type associated with the element being processed.
&C1ENVMNT (&C1EN)	8	Name of the current environment.
&C1FOOTPRT	64	Footprint of the element being processed.
&C1LEV	2	Level number for the element being processed.
&C1PKGID	16	Name of the package being executed. Blank if no package is being executed.
&C1PRGRP	8	Name of the current processor group. This symbolic 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.

Symbolic (alias)	Length	Replaced in processor by the
&C1PRTYPE	8	Function of the current processor (generate, move, or delete).
&C1SITE	1	Current site ID.
&C1STAGE (&C1ST)	8	Name of the current stage.
&C1STAGE1 (&C1ST1)	8	Stage 1 name. This symbolic must be used for the target stage name for a MOVE action.
&C1STAGE2 (&C1ST2)	8	Stage 2 name. This symbolic must be used for the target stage name for a MOVE action.
&C1STGID (&C1SI)	1	ID of the current stage.
&C1STGID1 (&C1SI1)	1	Stage 1 ID. This symbolic must be used for the target stage ID for a MOVE action.
&C1STGID2 (&C1SI2)	1	Stage 2 ID. This symbolic must be used for the target stage ID for a MOVE action.
&C1STGNUM (&C1S#)	1	Number of the current stage.
&C1SUBSYS (&C1SU)	8	Name of the current subsystem.
&C1SYSTEM (&C1SY)	8	Name of the current system.
<i>&CISELEMENT</i>	8	Element name at the source of MOVE or TRANSFER actions that execute a move processor.
<i>&CISELMNT10</i>	10	AllFusion™ CA-Panvalet element name at the source of MOVE or TRANSFER actions that execute a move processor.
<i>&CISELMNT255</i> 1	255	1-255 character name for elements at the source location of actions such as move and transfer.
<i>&CISELTYPE</i>	8	Element type at the source of MOVE or TRANSFER actions that execute a move processor.
<i>&CISEVMNT</i>	8	Environment name at the source of MOVE or TRANSFER actions that execute a move processor.
<i>&CISLEV</i>	2	Element level at the source of MOVE or TRANSFER actions that execute a move processor.
<i>&CISPRGRP</i>	8	Processor group name at the source of MOVE or TRANSFER actions that execute a move processor.

Symbolic (alias)	Length	Replaced in processor by the
<i>&CISSTAGE</i>	8	Stage name at the source of MOVE or TRANSFER actions that execute a move processor.
<i>&CISSTAGE1</i>	8	Stage 1 name at the source of MOVE or TRANSFER actions that execute a move processor.
<i>&CISSTAGE2</i>	8	Stage 2 name at the source of MOVE or TRANSFER actions that execute a move processor.
<i>&CISSTGID</i>	1	Stage ID at the source of MOVE or TRANSFER actions that execute a move processor.
<i>&CISSTGID1</i>	1	Stage 1 ID at the source of MOVE or TRANSFER actions that execute a move processor.
<i>&CISSTGID2</i>	1	Stage 2 ID at the source of MOVE or TRANSFER actions that execute a move processor.
<i>&CISSTGNUM</i>	1	Stage number at the source of MOVE or TRANSFER actions that execute a move processor.
<i>&CISSUBSYS</i>	8	Subsystem name at the source of MOVE or TRANSFER actions that execute a move processor.
<i>&CISSYSTEM</i>	8	System name at the source of MOVE or TRANSFER actions that execute a move processor.
<i>&CISVER</i>	2	Element version at the source of MOVE or TRANSFER actions that execute a move processor.
<i>&CIUSERID</i>	8	User ID associated with the current action.
<i>&CIUSRDSN</i>	44	Data set name of the source for ADD or UPDATE requests.
<i>&CIUSRFILE</i> 1	255	Source HFS file names for the ADD or UPDATE requests.
<i>&CIUSRMBR</i>	10	Source member name for the ADD or UPDATE requests.
<i>&CIUSRPTH</i> 1	768	HFS path name of the source for the ADD or UPDATE requests.

Symbolic (alias)	Length	Replaced in processor by the
&C1VER	2	Version number of the element being processed.
&C1XLANG	8	External language for the element type.

Note: **1** - To utilize these symbolics within a processor, you must use the Endeavor processor symbolic substringing feature. For example, &C1USRPTH(720,48).

2.5.7.1 Substringing

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

- `&SYMBOLIC(start,length,pad)`

In this expression:

start

The position within the symbolic where substringing is to begin. Default is 1.

length

The number of characters in the substring. Default is the length of the symbolic.

pad

A single character used to add trailing blanks. Default is blank.

If the *start* length is greater than the symbolic length (as defined in the table provided in 2.5.7, “Endeavor Symbolics” on page 2-14), Endeavor does not perform the substring substitution. If the *length* of the substring exceeds the length of the symbolic, Endeavor 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`

where **(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 **APNDVRT01**. 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, Endeavor 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, **NDVRO** 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 *&dsname* 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 symbolics), Endeavor begins substitution with the innermost parentheses. The following example illustrates the principles discussed in this section.

- `DSN = &LIB(&CIELEMENT(&S,&L,&P))`

where:

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

Endeavor resolves this statement by substituting the values for:

- the starting position (*&S*), length (*&L*), and padding character (*&P*):
`DSN = &LIB(&CIELEMENT(1,8,$))`
- *&CIELEMENT*:
`DSN = &LIB(ABC(1,8,$))`
- *&LIB*:

DSN = CA.LIB(ABC(1,8,\$))

When Endeavor finishes resolving the statement, the result is:

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

2.5.8 In-Stream Data

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

Note: 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, with each line truncated or padded to 72 characters.

2.6 Controlling Processor Flow

2.6.1 IF-THEN-ELSE statement

The Endeavor 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 Endeavor IF-THEN-ELSE statement is similar to the COND and EXECIF keywords function. However, the IF-THEN-ELSE statement can provide 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.

Endeavor allows selection of steps and DD statements using not only condition codes from prior steps but also values of Endeavor symbolics and processor symbolics. 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. For example:

```

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

```

Where:

keyword

Valid keywords are:

- Endeavor symbolics (see 2.5.7, “Endeavor Symbolics” on page 2-14)
- IBM defined expressions
 - RC (see 2.6.1.1, “RC” on page 2-28)
 - ABENDCC (see 2.6.1.2, “ABENDCC” on page 2-28)
 - ABEND (see 2.6.1.3, “ABEND” on page 2-28)
 - ¬ABEND (see 2.6.1.4, “¬ABEND” on page 2-29)
 - RUN (see 2.6.1.5, “RUN” on page 2-29)
 - ¬RUN (see 2.6.1.6, “¬RUN” on page 2-29)

operator

Valid operators are:

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

Note: See 2.4.3, “EXECIF” on page 2-7 for additional information regarding the operators.

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.

The first example is valid while the second example is invalid.

```

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

```

Valid Example

```

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

```

Invalid Example

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 (RC.STEP1=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.

```

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

```

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

```

An IF statement extending over multiple statements, with an 'X' in column 72.

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.

```

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

```

'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

```

'THEN' clause on a separate line from the 'IF' statement.

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 of a nested IF-THEN-ELSE statement.

DD statements related to an EXEC must be included within the IF-THEN-ELSE block for that step. Endeavor 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 Endeavor symbolic variables and literals. The six IBM expressions are:

- RC
- ABENDCC

- ABEND
- -ABEND
- RUN
- -RUN

2.6.1.1 RC

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

Example	Results
IF(RC=8)	Tests for a return code equal to 8
IF(stepname.RC>=10)	Tests for a return code greater than or equal to 10

2.6.1.2 ABENDCC

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

Example	Results
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)	Indicates the relational expression tests the abend code for a specific step.

2.6.1.3 ABEND

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

Example	Results
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.

2.6.1.4 `¬ABEND`

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

Example	Results
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.

2.6.1.5 `RUN`

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

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

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

2.6.1.6 `¬RUN`

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

```
IF(¬stepname.RUN)
```

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

The Endeavor symbolic variables are any variables currently supported by processors. A list of symbolic variables is given with an explanation of the EXECIF statement. See 2.4.3, “EXECIF” on page 2-7.

2.6.2 The ENDEVOR IF-THEN-ELSE Trace Facility

The Endeavor 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 Endeavor JCL to activate the trace.

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

Sample output: 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 SUBSTITUTED: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)
```

2.7 Authorizing a Non-Endevor Program

Programs that require authorization outside of Endevor may lose their authorization when used within Endevor. In order to maintain authorization in Endevor, those non-Endevor programs must be added to a specific Endevor table. Endevor currently hardcodes the following programs in this table: IEBCOPY, IEHMOVE, and IKJEFT01.

If you are using another program that requires authorization, you must add it to the Endevor table using a ZAP. Please refer to your optional PTF index to locate this ZAP and follow the instructions provided. If you have any questions on this procedure, call Endevor Technical Support.

Chapter 3. Processor Utilities

3.1 Overview

3.1.1 Utilities Available

The following utilities are distributed with Endeavor. You can include these utilities anywhere in your processor logic, to perform the functions described.

This Utility	Is used to:
BC1PDSIN	Initialize any number of allocated data sets that begin with ddname C1INIT. This utility is generally used to allocate list data sets.
BC1PTMP0	Execute 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	Install “transportable” footprints in object modules generated under OS/MVS or OS/390, DOS/VSE, or VM/CMS. For details, see the “Footprints” chapter in the <i>Administration Guide</i> .
BSTCOPY	Copy members from one partitioned data set to another. BSTCOPY allows for the use of MONITOR=COMPONENTS and backout.
C1BM3000	Execute Endeavor from within a processor.
C1PRMGEN	Create 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 Endeavor symbolic parameters contained in the statements, allowing you to vary the input control specifications based on the values of Endeavor symbolic parameters.
CONAPI	Used to invoke a program which issues ENDEVOR API calls through a processor.
CONDELE	Remove a member from a user library or HFS directory, after verifying the footprint for the member.

This Utility	Is used to:
CONLIST	Manage 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).
CONRELE	Include entities related to an element in a component list when generating component list reports and when using the LIST action.
CONSCAN	Create ACM relationships between an Endeavor 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	Combine all levels of an element, optionally expanding INCLUDE statements, write the merged source to a user-specified data set or HFS directory. Can be used as input to a specified program. Writes a component list to an external data set for further processing.
ENBX1000	See the “Expand Includes” section of the <i>Utilities Guide</i> for more information.

CAUTION:

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

3.2 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.

Notes:

To initialize a PDS within BC1PDSIN, specify:

1. The member name in the dataset name
2. DSORG=PS
3. The number of directory blocks to allocate in the SPACE parameter

3.2.1 Sample JCL

```
//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=PS)
//C1INITxx      DD  DSN=&&LNKLST,DISP=(,PASS,DELETE),
//              UNIT=SYSDA,SPACE=(TRK,(1,2),RLSE),
//              DCB=(RECFM=FBA,LRECL=121,BLKSIZE=3630,DSORG=PS)
```

3.3 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 Endeavor ISPF dialog.

Notes:

1. BC1PTMP0 issues TSO command processor STACK services using the BARRIER option. Only releases of TSO that support the BARRIER option can use this utility.
2. 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.

- If no TSO environment is present, BC1PTMP0 ends with a return code of 5. See 3.3.2, “Return Codes” on page 3-6, for more details.
- 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 an Endeavor processor.

Both the Endeavor Information/Management Interface and the Endeavor for DB2 product require the use of BC1PTMP0.

Warning: ISPF services cannot be invoked in a TMP. Endeavor 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.

3.3.1 Sample JCL

The following JCL is specified in the processor:

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

3.3.1.1 Parameters

Parameter	Description
PARM=	The name of the data set containing the commands to be executed by BC1PTMP0.
uprfx.uqual.CLISTLIB	The name of the list library containing the commands that you want to execute.
clist	The name of the CLIST that you want to execute.
iprfx.iqual.CONLIB	The Endeavor installation load library containing the program BC1PTMP0. (For DB2 processors, you may need to add your DB2 load library.)

3.3.2 Return Codes

BC1PTMP0 can return any of the following return codes:

Code	Meaning
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 Endeavor.
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.

3.3.2.1 Other Return Codes

Other return codes may be returned as a result of one of the commands processed by BC1PTMP0. For example, a program check that occurs in a program attached by BC1PTMP0 can result in a return code of S0C4.

3.4 BC1PXFPI Utility

The BC1PXFPI utility installs “transportable” footprints in object modules generated under OS/390, DOS/VSE, or VM/CMS. Transportable footprints provide footprint audit trails for software that executes in a non-OS/390 environments (DOS/VSE and VM/CMS).

For more information, see the “Transportable Footprints” chapter in the *Footprints Guide*. (This facility is for non-OS/390 environments only.)

3.5 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 Endeavor Automated Configuration Manager (ACM), or if package backout has been enabled for the output library. This is because Endeavor cannot detect the method used by IEBCOPY to update output library members.

3.5.1 Supported Copy Functions

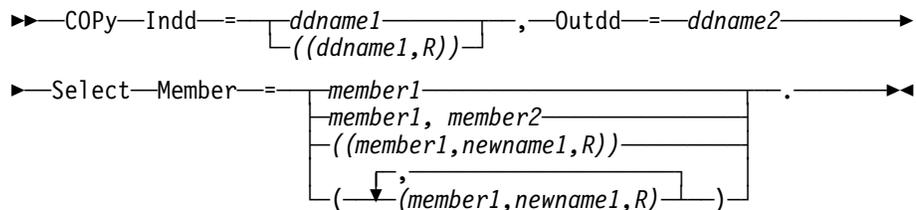
BSTCOPY can be used to copy between PDS load libraries and PDSE load libraries, from PDSE to PDSE, and from PDSE back to PDS. It cannot copy between a PDSE load library and any other Endeavor access method (AllFusion™ CA-Librarian, AllFusion™ CA-Panvalet, Endeavor/Lib).

BSTCOPY can copy aliases, but the alias must be explicitly requested. To copy an alias, the original loadlib member must be copied first.

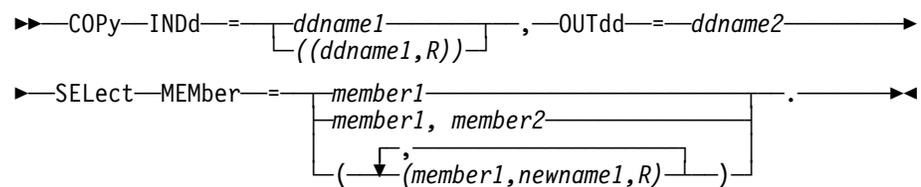
Copying from a PDSE load library to a PDS may result in errors, depending on whether the PDSE contains any program object members which cannot be converted back to load modules. This includes program objects which are greater than 16 megabytes in size, or which included mixed-case, extended names, or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

3.5.1.1 BSTCOPY Syntax - Literal Interpretation



3.5.1.2 BSTCOPY Syntax - Alternate Interpretation



CAUTION:

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

3.5.2 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))
/*
```

3.5.3 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 processor below 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,0=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,0=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 Endeavor/ACM 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.

3.5.4 SYSPRINT DCB Information

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

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

3.6 C1BM3000 Utility

The C1BM3000 utility allows you to execute Endeavor 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.
- Endeavor to write the Action Summary report to a separate file, you must pass the MSGOUT2 parameter.

Warning: Backout information is not created for elements processed by the C1BM3000 utility.

3.6.1 Do Not Use...

Do not use the following:

- BSTIPT01 for SCLIN, the name is reserved by Endeavor for input SCL.
- C1MSG1 for MSGOUT1, the name is reserved by Endeavor for batch execution report messages.
- C1MSG2 for MSGOUT2, the name is reserved by Endeavor for the Action Summary report.

CAUTION:

You cannot act against an element that is currently being acted upon. Except for the LIST and PRINT actions, the SCL executing in a processor cannot execute an action for the same element. Nor can this SCL perform any source changes to the element being processed. This creates a “deadly embrace,” loop by waiting for an element that is currently in use.

3.6.2 Sample JCL

Sample JCL for the C1BM3000 utility is illustrated below. 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.AUTHLIB,  
// 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.

3.7 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 in-stream data, you can include Endeavor symbolics in the parameter passed to C1PRMGEN. If you do this, the utility expands the symbolics as it creates the output statements, allowing you to vary the input control statements passed to subsequent job steps based on the values of Endeavor symbolic parameters. This is illustrated next, where C1PRMGEN creates two control statements for use by the IBM IEBCOPY utility.

3.7.1 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,0=0| S M=((&&C1ELEMENT,,R)) '  
//PARMOUT DD DSN=&&CPYPARM,DISP=(,PASS,DELETE)  
//          DCB=(RECFM=FB,LRECL=80,BLKSIZE=80)
```

3.7.1.1 Parameters

Parameter	Description
PARM=	<p>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 (I) in the PARM statement, as shown above.</p> <p>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:</p> <ul style="list-style-type: none"> ■ 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: <pre> Col Col 1 72 PARM='abcd.....'ZX // z.....a' </pre> <ul style="list-style-type: none"> ■ 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	<p>The data set to which the expanded statement is written.</p> <p>In the previous example, the output written to PARMOUT, assuming that the current element is NDVR, is as follows:</p> <pre> Col 1 C I=I,0=0 S M=((NDVR,R)) </pre>

3.8 CONAPI Utility

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

A program which issues ENDEVOR 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.

For more information on the API interface, refer to the *API Guide*.

3.8.1 Sample JCL

```
//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
```

3.9 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

HFS directory

This utility is generally used in delete processors.

Before it removes the member, CONDELE checks the member for a footprint that matches the element being processed. If the footprint is missing or invalid, CONDELE deletes the member, but passes back a Endeavor informational message stating that a footprint compromise has occurred.

3.9.1 Sample JCL

Sample JCL for this utility is shown below:

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

3.9.1.1 Parameters

Parameter	Description
PARM=	<p>Indicates an alternate member name (<i>mbr-name</i>) 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.</p> <p>CAUTION: 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.</p>
C1LIB	<p>Specifies the library or HFS directory containing the member targeted for deletion. The member name is the name of the element being processed (generally moved, deleted, or archived).</p>

3.10 CONLIST Utility

The CONLIST utility is a multi-purpose utility used to manage output listings. Five options are available with this utility: STORE, PRINT (the default), PRTMBR, COPY, and DELETE. Use the PARM= statement on the EXEC card to specify the option you want.

Note: All of these options, except for the MBR parameter, are mutually exclusive; that is, you can use *only one* option in the PARM= statement.

PARM	Description
STORE	<p>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 Endeavor LIB data set which is stored as if RECFM=VBA. You can optionally generate a banner page (as described below) and store it at the front of the member.</p> <p>If more than one file is input, CONLIST concatenates the files before storing them.</p> <p>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.</p>
PRINT	<p>Default. Prints a temporary list data set, optionally generating a banner page before the listing.</p> <p>If more than one file is input, CONLIST concatenates the files before printing them.</p>
PRTMBR	<p>Decompresses then prints a member from a listing library.</p>
COPY	<p>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.</p>
DELETE	<p>Deletes a member from the output library.</p>

PARM	Description
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.

3.10.1 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.

```

*****
*****
*
* IIIIIIIIII PPPPPPPPPP PPPPPPPPPP SSSSSSSSSS 2222222222 11 00000000 6666666666 *
* IIIIIIIIII PPPPPPPPPPPP PPPPPPPPPPPP SSSSSSSSSSSS 2222222222 111 0000000000 666666666666 *
* II PP PP PP SS SS 22 22 1111 00 00 66 66 *
* II PP PP PP SS 22 11 00 00 66 *
* II PP PP PP SSS 22 11 00 00 66 *
* II PPPPPPPPPPPP PPPPPPPPPPPP SSSSSSSSSS 22 11 00 00 6666666666 *
* II PPPPPPPPPPPP PPPPPPPPPPPP SSSSSSSSSS 22 11 00 00 6666666666 *
* II PP PP SSS 22 11 00 00 66 66 *
* II PP PP SS 22 11 00 00 66 66 *
* II PP PP SS 22 11 00 00 66 66 *
* IIIIIIIIII PP PP SSSSSSSSSSSS 2222222222 1111111111 0000000000 666666666666 *
* IIIIIIIIII PP PP SSSSSSSSSS 2222222222 1111111111 00000000 6666666666 *
*****
*****
**
** ***** UPDATE ***** **
**
** USER ID.....JSMITH1 **
** DATE.....15JUN00 15:36 **
** Endeavor RC.....0000 **
**
** ENVIRONMENT....CA **
** STAGE.....QA **
** SYSTEM.....FINANCE **
** SUBSYSTEM.....ACCTPAY **
** ELEMENT.....IPPS2106 **
** VV.LL.....01.01 **
** TYPE.....COBPROG **
** PROC GRP.....COBOLSTD **
** PROCESSOR.....GENCOB01 **
** INITCOB.....RC=0000 **
** INITLNK.....RC=0000 **
** WRITE.....RC=0000 **
** COMPILE.....RC=0000 **
** LKED.....RC=0000 **
**
** *****

```

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

- The element name across the top (IPPS2106 in the example above).
- A summary of the processor being run (user ID, date, Endeavor return code, stage, etc.).

- An itemization of each processor step, up to but not including the CONLIST step, with the Endeavor 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.

In the example shown next, the COMPILE step ended with an 0004 return code, which exceeded the specified MAXRC:

```

**      WRITE.....RC=0000      **
**      **COMPILE.....RC=0004 > MAXRC  **
**      LKED.....NOT EXEC (CC)  **

```

3.10.2 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)

```

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

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

3.10.3 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)
```

Parameter	Description
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.

3.10.4 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)
```

Parameter	Description
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)'
```

3.10.5 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)
```

Parameter	Description
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 LIST nn 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 LIST nn data set, assign the ddnames sequentially (LIST01, LIST02, and so forth). LIST nn data sets are concatenated (in order by the nn suffix) before they are appended to the member.

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

3.10.6 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
```

Parameter	Description
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)'
```

3.10.7 Guidelines When Creating Listings

There are several ways you can approach the creation and handling of listings within Endeavor processors.

Recommended Approach: The recommended approach involves:

1. Writing the listings to a data set other than SYSOUT during the processor step
2. Running the Endeavor 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 3.2, “BC1PDSIN Utility” on page 3-4.
2. In the JCL for the processor utilities, write to the appropriate listing data set--*not* to SYSOUT.
3. Use the Endeavor 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 which 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.
- Endeavor 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.
- Endeavor CONLIST utility creates a Endeavor 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 (as described in the *User Guide*). 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 below.

```

//*
//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=*
//*

```

- Endeavor 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)
*

```

3.11 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 Endeavor 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 Endeavor.

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. Use the following sample processor JCL to execute CONRELE:

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

3.11.1 CONRELE Utility Commands

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

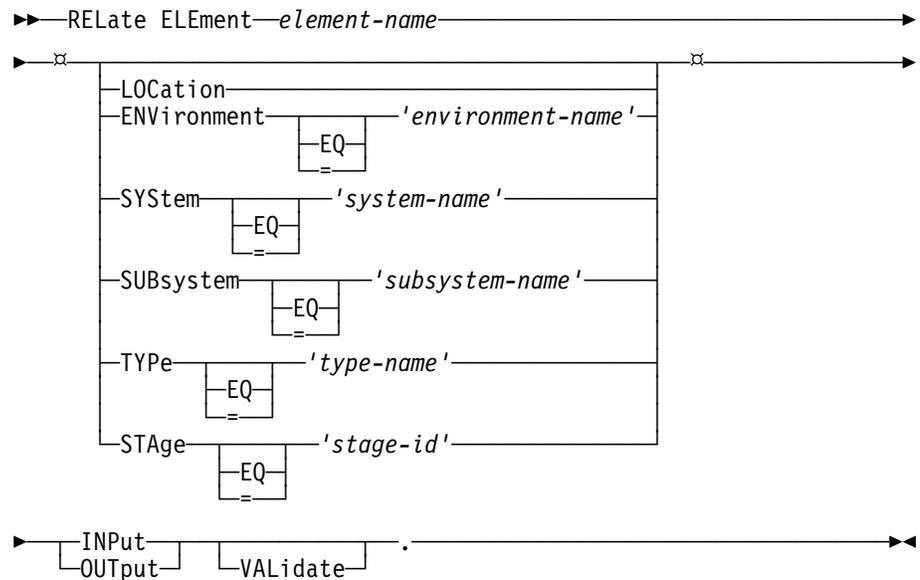
- RELATE ELEMENT--Relates an element to another element.
- RELATE MEMBER--Relates a data set member to an element.
- RELATE OBJECT--Relates an object such as a pathname or a filename for an object on another platform to an element.
- RELATE COMMENT--Adds comments to a component list.
- SET ERROR RETURN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

The syntax for these commands follows.

3.11.2 RELATE ELEMENT Command Syntax

The RELATE ELEMENT command syntax is shown below. For details about building SCL commands, see the *SCL Reference Guide*.

3.11.2.1 RELate ELEment Syntax



Parameters

element-name

The name of the element. The maximum element-name length is 10.

LOCATION

Location of the *member-name*. This is optional. If you do not include the location in your syntax the location defaults to the target location of the current Endeavor element.

INPut

Input component

OUTput

Output component

VALIDATE

Verifies the elements' presence at the specified location in Endeavor. 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.

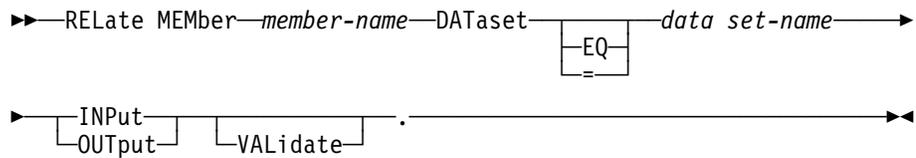
- When the VALIDATE option is specified, the location names (environment, system, etc.) cannot be wildcarded. If they are omitted, the location names associated with the target element are used.
- When the VALIDATE option is NOT specified, CONRELE stores the location information "as specified". Omitted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

3.11.3 RELATE MEMBER Command syntax

The RELATE MEMBER command syntax is shown below.

3.11.3.1 RELate MEMber Syntax



Parameters

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

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

If you specify the VALIDATE option, Endevor 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.

3.11.4 RELATE OBJECT Command Syntax

The RELATE OBJECT command syntax is shown below.

3.11.4.1 RELate OBJect Syntax



Parameters**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.

3.11.5 RELATE COMMENT Command Syntax

The RELATE COMMENT command syntax is shown below.

3.11.5.1 RELate COMment Syntax

►►—RELate COMment—*comment-data*—.

Parameters**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.

3.11.6 SET ERROR RETURN CODE Command Syntax

The SET ERROR RETURN CODE command syntax is shown below.

3.11.6.1 SET ERRor RETurn CODE Syntax

►►—SET ERRor RETurn CODE—EQ—*error-return-code*—.

Parameters**error-return-code**

Specifies the error return code when the Validate option fails. The default error return code is 8.

3.11.7 Example of CONRELE Syntax

This is an example of the CONRELE syntax.

```
SET ERROR RETURN CODE 0000.
RELATE ELEMENT BGSQ600
LOCATION
ENVIRONMENT = TEST
SYSTEM      = FINANCE
SUBSYSTEM   = AP
TYPE        = BGLOAD2
```

```
        STAGE      = 1
    INPUT.
RELATE ELEMENT BGSQ723
    LOCATION
        ENVIRONMENT = TEST
        SYSTEM      = FINANCE
        SUBSYSTEM   = AP
        TYPE        = BGLoad3
        STAGE      = 1
    INPUT.
RELATE ELEMENT BGSQ601
    LOCATION
        ENVIRONMENT = TEST
        SYSTEM      = FINANCE
        SUBSYSTEM   = AP
        TYPE        = BGLoad2
        STAGE      = 1
    OUTPUT.
RELATE ELEMENT BGSQ64
    LOCATION
        ENVIRONMENT = TEST
        SYSTEM      = FINANCE
        SUBSYSTEM   = AP
        TYPE        = DB2COB3
        STAGE      = 1
    OUTPUT.
RELATE ELEMENT BGSQ65
    LOCATION
        ENVIRONMENT = TEST
        SYSTEM      = FINANCE
        SUBSYSTEM   = AP
        TYPE        = DB2COB
        STAGE      = 1
    OUTPUT
    VALIDATE.
RELATE MEMBER BC1PSQ1
    dataset = 'JSMITH.SRCLIB'
    INPUT.
RELATE MEMBER BGSQ60
    dataset = 'JSMITH.DBRMLIB'
    INPUT
    VALIDATE.
RELATE MEMBER BC1PSQ3
    dataset = 'JSMITH.SRCLIB'
    OUTPUT.

RELATE MEMBER BGSQ70
    dataset = 'JSMITH.DBRMLIB'
    INPUT
    VALIDATE.
RELATE OBJECT 'D;\ENDEAVOR\TEMP.DOC'.
RELATE OBJECT 'D;\ENDEAVOR\TEMP.DOC2'.
RELATE OBJECT 'D;\ENDEAVOR\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'.
```

3.12 CONSCAN Utility

The CONSCAN utility allows you to identify additional Automated Configuration Management (ACM) relationships between Endeavor 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. Refer to 3.11, “CONRELE Utility” on page 3-26 for more information regarding CONRELE utility.

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 Endeavor processor, followed by a CONRELE step. If used as a stand-alone utility, none of the Endeavor processor symbolic parameters (&C1ELEM, etc.) are available and the processor IF, THEN, ELSE logic cannot be utilized.

3.12.1 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 Endeavor TYPE that utilizes the CONSCAN utility.

3.12.2 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.

3.12.3 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.

'string',

Specified 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.

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.

Exclusion Group Examples:

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
```

3.12.4 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.

See “Example 3” on page 3-39 for an example of the REJECT parameter.

STRING=

Indicates to start scanning this record for a string.

'string',

Specified 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.

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

- If TYPE is FORW or BACK, PARM defines the number of characters after or before the last character found by the FIND statement(s).
- 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.

n

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.

3.12.4.1 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. Listed below is a sample CONRELE control statement output.

```

RELATE COMMENT
    'WS-ABEND-PGM'
.
RELATE ELEMENT DTESUB
    LOCATION
    ENVIRONMENT = ' '
    SYSTEM = ' '
    SUBSYSTEM = ' '
    TYPE = ' '
    STAGE = ' '
    INPUT
.
RELATE MEMBER SCANCBL
    data set='uprfx.uqual.SOURCE'
    INPUT
.
RELATE OBJECT
    'iprfx.iqua1.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)	SCANTYPE MEMBER	RELATE MEMBER ABC
//DD DSN=SYS1.PROC2II	FIND1 STRING='DSN=',	DSN='SYS1.PROCS'
	POS=ANY	INPUT.
	START TYPE=DFLT	RELATE OBJECT
	END1 TYPE=SPAC	DSN='SYS1.PROC2II'
	END2 TYPE=CHAR,	INPUT.
	PARM=','	

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 procs. 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 SCANTYPE ELEMENT FIND1 STRING='EXEC', POS=ANY START TYPE=DFLT END1 TYPE=SPAC	RELATE ELEMENT NDVR LOCATION ENVIRONMENT = ' ' SYSTEM = ' ' SUBSYSTEM = ' ' TYPE = ' ' STAGE = ' ' INPUT.

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'.

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

3.12.5 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.

3.12.5.1 Sample CONSCAN Utility Processor

The following figure shows sample processor CONSCAN JCL. This sample can be found in the iprfx.igual.JCL library.

```

/*-----*
/*
/* COPYRIGHT (C) COMPUTER ASSOCIATES 2000
/*
/* 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=4096K
/******
/* INPUT FILES
/******
//SRCIN DD DISP=(OLD,DELETE),DSN=&&ELMOUT
//PARMSCAN DD DISP=SHR,DSN=uprfx.igual.SOURCE(&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
/*

```

3.12.6 Error Messages

Following is a sample of messages issued by the CONSCAN utility, additional messages are documented in the *Error Codes and Messages Guide*.

C2FM0010 C2FSCANC C2FSCANC SCANCE COMMENT CARD IN ERROR

Explanation: A syntax error was detected in one of the COMMENT control cards. Correct the syntax error and rerun the job.

C2FM0011 C2FSCANR C2FSCANR SCANCR CARD RULE IN ERROR

Explanation: A syntax error was detected in one of the SCANTYPE control cards. Correct the syntax error and rerun the job.

C2FM000F CONSCAN C2FREADP PARMSCAN PARAMETER IN ERROR

Explanation: Warning, no records were selected.

3.13 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 HDFS directory.
 - Passes individual element data to a user exit program.

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

3.13.1 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.

3.13.1.1 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.

3.13.1.2 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). Refer to 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.

3.13.2 Writing Elements to an External Location

When CONWRITE writes to an external data set or HFS 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.

3.13.3 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.

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

The JCL below 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)
```

3.13.4 Extended Form of CONWRITE

You can use the extended form of CONWRITE to extract elements or component list records for any Endeavor 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 JCL below is an example of the extended form of CONWRITE

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

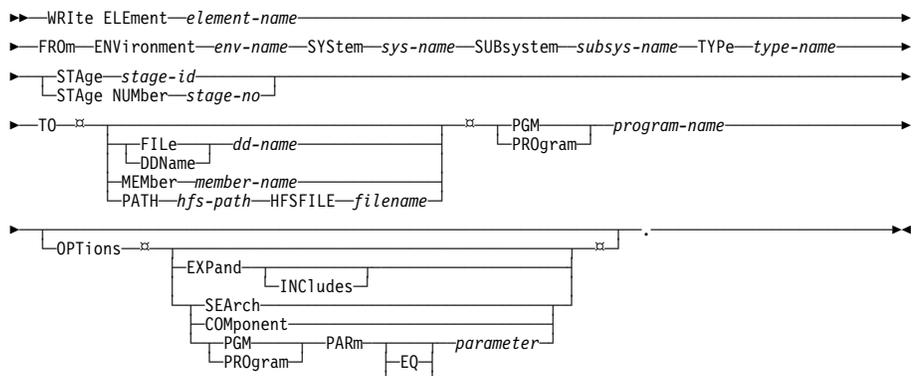
You create copies of more than one element or component list record using the CONWIN DD statement. You can also use this statement to pass the individual element or component list records to a user exit program.

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

```
//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 &clelement
FROM ENV &c1envmnt SYSTEM &c1system SUBSYSTEM &c1subsys
TYPE &c1w1type STAGE &c1stgid
TO DDN COMPOUT
OPTION COMPONENT.
/*
```

3.13.5 Command Syntax for the CONWRITE Utility

3.13.5.1 CONWRITE Syntax



Parameters**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 example below 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)

```

3.13.6 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.

3.13.6.1 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, Endeavor 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.

3.13.7 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.

Return Code	Meaning
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.

Chapter 4. Classifying and Managing Processors

4.1 Overview

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 chapter discusses the classification scheme.

4.2 Classifying Processors

You must classify processors as type PROCESS. The Type Definition panel is shown below:

```

CREATE ----- TYPE DEFINITION -----
COMMAND ==>
CURRENT ENV: DEMO          STAGE ID: D      SYSTEM: ADMIN    TYPE: PROCESS

DESCRIPTION                ==> ENDEVOR PROCESSOR DEFINITIONS

----- ELEMENT OPTIONS -----
FWD/REV/IMG DELTA: R (F/R/I) COMPRESS BASE/ENCRYPT NAME: Y (Y/N)
DFLT PROC GRP:      PROCESS REGRESSION PCT ==> 50   REGR SEV ==> W (I/W/C/E)
SOURCE LENGTH:     80      COMPARE FROM:      1     COMPARE TO:   72
AUTO CONSOL:       N      LANGUAGE:        CNTLPROC PV/LB LANG:   DATA
REMOVE/CONSOL AT LVL: 96   HFS RECFM:          (COMP/CR/CRLF/F/LF/NL/V)
LVLS TO REMOVE/CONSOL: 50  WS HOME OPSYS:      WS FILE EXT:

----- COMPONENT LIST OPTIONS -----
FWD/REV DELTA:     F (F/R) AUTO CONSOL: Y (Y/N) CONSOL AT LVL: 99
LVLS TO CONSOL:   49

----- LIBRARIES -----
BASE/IMAGE LIBRARY ==> ENDEVOR.DEMO.STG1.BASE
DELTA LIBRARY:     ==> ENDEVOR.DEMO.STG1.DELTA
INCLUDE LIBRARY:   ==>
SOURCE O/P LIBRARY:
EXPAND INCLUDES:   N (Y/N)

```

For elements of type PROCESS, Endeavor 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 Endeavor.

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.

The Processor Group Definition panel for group PROCESS is shown below:

```
CREATE ----- PROCESSOR GROUP DEFINITION -----  
COMMAND ==>  
  
CURRENT ENV: DEMO      STAGE ID: D      SYSTEM: ADMIN      TYPE: PROCESS  
  
PROCESSOR GROUP ==> PROCESS  
  
DESCRIPTION      ==> AUTOMATIC CONVERSION OF TYPE PROCESSOR INFORMATION  
  
----- OUTPUT MANAGEMENT INFORMATION -----  
  
                                FOREGROUND EXECUTION  
GENERATE PROCESSOR: GPPROCSS      ==> Y (Y/N)  
DELETE PROCESSOR  : DPPROCSS      ==> Y (Y/N)  
MOVE PROCESSOR   : GPPROCSS      ==> Y (Y/N)
```

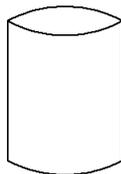
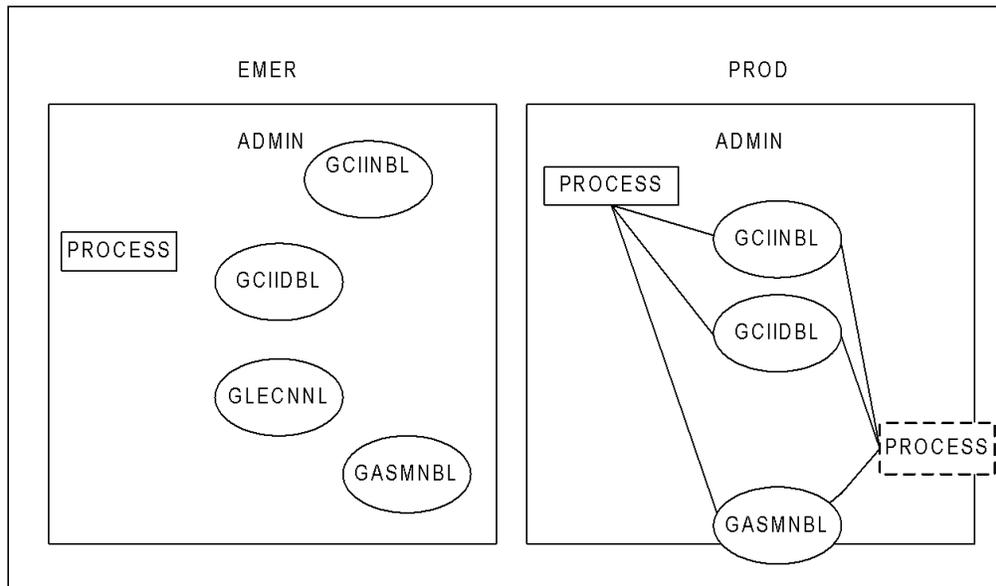
After defining the type 'process', you can perform any Endeavor action against the type 'process'. For more information about action processing, see the *User Guide*.

4.3 Managing Processors

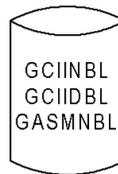
4.3.1 Procedure: Implementing Processors

Follow this procedure to implement processors:

1. Define a system, for example ADMIN, in your production environment, specifying ENDEVOR.EMER.PRCLOAD as the Stage 1 processor load library and ENDEVOR.PROD.PRCLOAD 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 diagram below shows the resulting configuration:



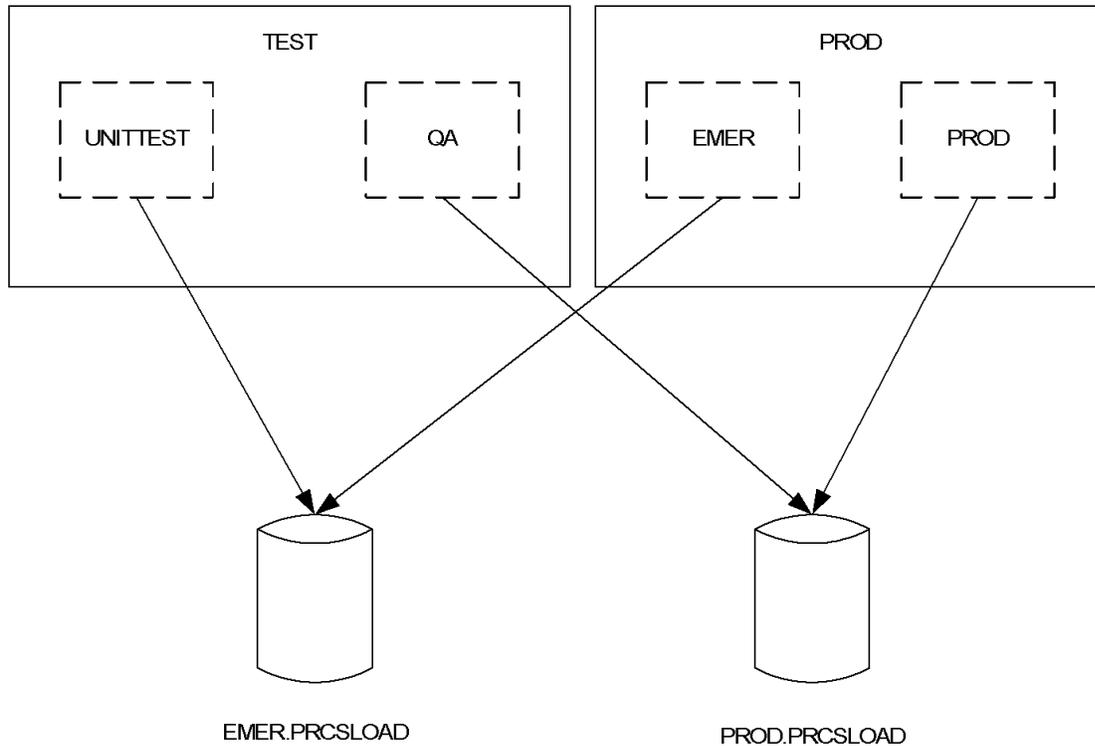
EMER.PRCLOAD



PROD.PRCLOAD

3. Have all systems in all environments reference ENDEVOR.EMER.PRCLOAD as their Stage 1 processor load library, and

ENDEVOR.PROD.PRCLOAD 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.

4.3.2 Procedure: Maintaining Processors

To maintain processors managed according to the previous example:

1. Copy your current administration system to create a test system.
2. Modify the PRCLOAD 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.

4.3.3 Where Endeavor Looks for Processors

After determining the processor group for an action, Endeavor looks for the requisite processor first in Stage 1, then in Stage 2.

Chapter 5. Processor Groups

5.1 Processor Group Overview

5.1.1 Three Types

As described in 1.1, “Processor Overview” on page 1-2, there are three main processor types, namely delete, generate, and move. Instead of associating one of each type of processor with each element type, Endeavor 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 Endeavor 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 (for more information on symbolics, see 2.5, “Symbolic Parameters” on page 2-10.)

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 Endeavor allows you to create a single COBOL type with two processor groups, one to handle each variation of COBOL code. Furthermore, symbolics allow you to use the same processor for both types, changing only the symbolic overrides in the two processor groups.

You can set up processor groups in Stage 1 and Stage 2. (If processor groups exist for Stage 1, they must also exist for Stage 2.)

5.1.2 Suggested Naming Conventions for Processor Groups

Processor group names can have up to eight characters. The abbreviations below do not represent a complete list, and are offered as guidelines only

Position	Description
1-3	Language Type. <ul style="list-style-type: none"> ▪ ASM = Assembler ▪ COB = COBOL ▪ CII = COBOL 2 ▪ EAS = Easytrieve ▪ FOR = Fortran ▪ PLI = PL-1 ▪ RPG = RPG ▪ TRA = Transform ▪ UTL = Utility
4	Database environment. <ul style="list-style-type: none"> ▪ D = DB2/DL1 ▪ S = IDMS ▪ I = IMS ▪ N = None
5	Operating environment. <ul style="list-style-type: none"> ▪ B = Batch ▪ C = CICS ▪ S = IDMS-DC ▪ I = IMS-DC ▪ N = None
6	Output type. <ul style="list-style-type: none"> ▪ 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.

5.1.2.1 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.

5.1.3 User-Defined Symbolics

Endeavor supports user-defined symbolics in processors. The symbolics 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 symbolics for this processor group. (For details about symbolics, see 2.5, “Symbolic Parameters” on page 2-10.)

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 symbolics at execution time (see 5.3, “Working with Processor Group Symbolics” on page 5-7).

5.2 Working with 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.

5.2.1 From the Environment Options Menu

To create, update, or display processor group information from the Environment Options Menu:

1. Select option **6** and press ENTER. Endeavor displays the Processor Group Request panel.

```

----- PROCESSOR GROUP REQUEST -----
OPTION ==>
  blank - Display processor group definition
  # - Delete processor group definition
  C - Create processor group definition
  U - Update processor group definition
ENVIRONMENT ==> TEST
SYSTEM      ==> FINANCE
TYPE        ==> PROCESS
STAGE       ==> 1      1 - UNITTEST  2 - QA
GROUP       ==>

```

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 Endeavor 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 a **#** 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. For details about this panel, see 5.5, “Processor Group Definition Panel” on page 5-12.

5.2.2 From the 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. Endeavor 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. Endeavor displays the message **GROUP CREATED** in the upper right corner. For details about this panel, see 5.5, “Processor Group Definition Panel” on page 5-12.
4. To return to the Type Definition panel, press END.

5.3 Working with Processor Group Symbolics

To change or display the symbolics 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. Endeavor displays the processing option you have selected in the upper left corner of this panel.

```

UPDATE:----- PROCESSOR GROUP SYMBOLICS ----- Row 1 of 20
COMMAND ==>                                     SCROLL ==> PAGE

CURRENT ENV: TEST      STAGE ID: 1  SYSTEM: FINANCE  TYPE: ASMPGM

PROCESSOR GROUP: ASMIRUAL      PROCESSOR: GASM
DESCRIPTION: ASSEMBLER - REUSEABLE, AUTHORIZED
LOAD LIBRARY: ENDEVOR.PROD.LOADLIB
DEFAULT VALUES ARE INDICATED BY -
OVERRIDE VALUES ARE INDICATED BY 0

SYMBOLIC -/O VALUE
AUTH      0 1
DENV      - D40
IENV      - I40
LET       - NOLET
LINK      - YES
LOADLIB   0 AUTHLIB
MACLIB    - SYS1.MACLIB
MVSLVL    0 B40
PENV      - P40
QENV      - Q40
RENT      0 NORENT
REUS      - REUS

```

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 symbolic 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 symbolic by typing **-- (dash)** in the **-/O** field and pressing ENTER.

Once you have established a default value and an override value for a symbolic you can toggle back and forth between them by typing **-- (dash)** or **O** in the **D/O** field and pressing ENTER.

Note: If the dash is specified the symbolic's original value is restored when you end your Endeavor session.

5.3.1.1 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 symbolics. You then use the same processors as the base for a second processor group (COBNBL01) that, by changing the symbolics, 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 Endeavor.
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**.

```

DISPLAY ----- TYPE DEFINITION -----
COMMAND ==>

CURRENT ENV: TEST      STAGE ID: Q      SYSTEM: FINANCE   TYPE: COBPROG
NEXT   ENV: PROD      STAGE ID: P      SYSTEM: FINANCE   TYPE: COBPROG

DESCRIPTION:          COBOL PROGRAMS
UPDATED:              15JUN01 01:04 BY JSMITH

----- ELEMENT OPTIONS -----
FWD/REV/IMG DELTA: R (F/R/I) COMPRESS BASE/ENCRYPT NAME: Y (Y/N)
DFLT PROC GRP:      COBNBL  REGRESSION PCT: 50  REGR SEV: W (I/W/C/E)
SOURCE LENGTH:      80      COMPARE FROM: 7  COMPARE TO: 72
AUTO CONSOL:        N      LANGUAGE: COBOL  PV/LB LANG: COBOL
REMOVE/CONSOL AT LVL: 96  HFS RECFM: (COMP/CR/CRLF/F/LF/NL/V)
LVLS TO REMOVE/CONSOL: 50  WS HOME OPSYS: WS FILE EXT:

----- COMPONENT LIST OPTIONS -----
FWD/REV DELTA: F (F/R) AUTO CONSOL: N (Y/N) CONSOL AT LVL: 99
LVLS TO CONSOL: 49

----- LIBRARIES -----

BASE/IMAGE LIBRARY: ENDEVOR.NDVR.BASE
DELTA LIBRARY: ENDEVOR.NDVR.DELTA
INCLUDE LIBRARY:
SOURCE O/P LIBRARY:
EXPAND INCLUDES: Y (Y/N)

```

When you press ENTER, Endeavor displays a Processor Group Definition panel.

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. A completed Processor Group Definition panel is shown below.

```

UPDATE ----- PROCESSOR GROUP DEFINITION -----
COMMAND ==>

CURRENT ENV: TEST      STAGE ID: Q      SYSTEM: FINANCE      TYPE: COBPROG
NEXT ENV:  PROD      STAGE ID: P      SYSTEM: FINANCE      TYPE: COBPROG

PROCESSOR GROUP ==> COBNBL      PROCESSOR O/P TYPE ==>
DESCRIPTION      ==> COBOL COMPILE AND LINK, LISTING IS STORED
NEXT PRCS GROUP ==> COBNBL

UPDATED:          15JUN01 01:04 BY JSMITH

----- OUTPUT MANAGEMENT INFORMATION -----

PROCESSOR TO USE FOR MOVE ACTION: M (M/G)
PROCESSOR TO USE FOR TRANSFER ACTION: G (M/G)

S - Browse Symbolics          L - List Processor
U - Update Symbolics

GENERATE PROCESSOR ==> GCIINBL      FOREGROUND EXECUTION ==> Y (Y/N)
DELETE PROCESSOR  ==> DLODNNN      ==> Y (Y/N)
MOVE PROCESSOR    ==> MLODNNL      ==> Y (Y/N)

```

4. To view a list of the symbolics for any of these processors, type **S** next to a processor and press ENTER. The Processor Group Symbolics panel for the GCIINBL processor is shown below.

```

DISPLAY ----- PROCESSOR GROUP Symbolics ----- --ROW 1 OF 14
COMMAND ==>                                     SCROLL ==> PAGE
CURRENT ENV: TEST      STAGE ID: QA      SYSTEM: FINANCE      TYPE: COBOL
PROCESSOR GROUP: COBNBL
DESCRIPTION: COBOL COMPILE AND LINK, LISTING IS STORED
PROCESSOR: GCIINBL
LOAD LIBRARY: ENDEVOR.NDVR.PRCLOAD
DEFAULT Values are indicated by "-"
OVERRIDE Values are indicated by "0"
SYMBOLIC -/O VALUE
COBLIB --- SYS1.VSCLLIB
COBSTPLB --- SYS1.VSCOLIB
CSYSLIB1 --- ENDEVOR.STG1.NDVR.COPYLIB
CSYSLIB2 --- ENDEVOR.STG2.NDVR.COPYLIB
EXPINC --- N
LISTLIB --- ENDEVOR.STG1.NDVR.LISTING
LOADLIB --- ENDEVOR.STG1.NDVR.LOADLIB
LSYSLIB1 --- ENDEVOR.STG1.NDVR.LOADLIB
LSYSLIB2 --- ENDEVOR.STG2.NDVR.LOADLIB
MEMBER --- &C1ELEMENT
MONITOR --- COMPONENTS

```

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. The completed Processor Group Definition panel is shown below.

```

UPDATE ----- PROCESSOR GROUP DEFINITION -----
COMMAND ==>

CURRENT ENV: TEST      STAGE ID: Q      SYSTEM: FINANCE      TYPE: COBPROG
NEXT  ENV:  PROD      STAGE ID: P      SYSTEM: FINANCE      TYPE: COBPROG

PROCESSOR GROUP ==> COBNBL01      PROCESSOR O/P TYPE ==>
DESCRIPTION      ==> COBOL COMPILE AND LINK, LISTING IS STORED
NEXT PRCS GROUP ==> COBNBL01

UPDATED:                15JUN01 01:04 BY JSMITH

----- OUTPUT MANAGEMENT INFORMATION -----

PROCESSOR TO USE FOR MOVE ACTION:      M (M/G)
PROCESSOR TO USE FOR TRANSFER ACTION:   G (M/G)

      S - Browse Symbolics                L - List Processor
      U - Update Symbolics

                                FOREGROUND EXECUTION
GENERATE PROCESSOR ==> GCIINBL      ==> Y (Y/N)
DELETE PROCESSOR  ==> DLODNNN      ==> Y (Y/N)
MOVE PROCESSOR    ==> MLODNNL      ==> Y (Y/N)

```

Note: We have used the same generate, move, and delete processors for both processor groups.

1. 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 symbolic LISTLIB, then type new information in the VALUE field for this symbolic.

```

UPDATE ----- PROCESSOR GROUP SYMBOLICS ----- ROW 1 OF 14
COMMAND ==>                                     SCROLL ==> PAGE
CURRENT ENV: TEST      STAGE ID: Q      SYSTEM: FINANCE      TYPE: COBOL
PROCESSOR GROUP: COBNBL
DESCRIPTION:          COBOL COMPILE AND LINK, LISTING IS STORED
PROCESSOR:           GCIINBL
LOAD LIBRARY:        ENDEVOR.STG1.NDVR.PRCLOAD
  To override symbolics : Enter "0" and supply new value
  To reset to Default value: Enter "---"
  To exit without saving : Enter CANCEL on the command line
SYMBOLIC -/O VALUE
COBLIB  --- SYS1.VSCLLIB
COBSTPLB --- SYS1.VSCOLIB
CSYSLIB1 --- ENDEVOR.STG1.NDVR.COPYLIB
CSYSLIB2 --- ENDEVOR.STG2.NDVR.COPYLIB
EXPINC  --- N
LISTLIB  0 NO
LOADLIB  --- ENDEVOR.STG1.NDVR.LOADLIB

```

2. Press ENTER. You have created a second processor group to print compile listings by simply changing the symbolics referenced by the GCIINBL processor.

5.3.2 Displaying 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. For details about this panel, see 5.7, "Processor Display Panel" on page 5-19.

5.4 Processor Group Selection List

Endevor 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.

```

----- PROCESSOR GROUP SELECTION LIST ----- ROW 1 OF 4
COMMAND ==>                                SCROLL ==> PAGE

CURRENT ENV: TEST      STAGE ID: Q      SYSTEM: FINANCE   TYPE: COBOL
NEXT   ENV: PROD      STAGE ID: P      SYSTEM: FINANCE   TYPE: COBOL

PROCESSOR
GROUP      PROCESSOR GROUP DESCRIPTION
COBDBL    DB2 COBOL COMPILE AND LINK EDIT LISTING IS STORED
COBNBL    COBOL COMPILE AND LINK, LISTING IS STORED
COBNBL01  COBOL COMPILE AND LINK, LISTING IS PRINTED
COBNBO    COBOL COMPILE ONLY, LISTING IS PRINTED
***** BOTTOM OF DATA *****

```

The panel fields are described below. Except for SELECTION, they are display-only.

Field	Description
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.

5.5 Processor Group Definition Panel

When you update and/or create processor groups for all types other than type PROCESS, Endeavor displays the following Processor Group Definition panel.

```

UPDATE ----- PROCESSOR GROUP DEFINITION -----
COMMAND ==>

CURRENT ENV: TEST      STAGE ID: Q      SYSTEM: FINANCE   TYPE: COBPROG
NEXT     ENV: PROD      STAGE ID: P      SYSTEM: FINANCE   TYPE: COBPROG

PROCESSOR GROUP ==> COBNBL      PROCESSOR O/P TYPE ==>
DESCRIPTION      ==> COBOL COMPILE AND LINK, LISTING IS STORED
NEXT PRCS GROUP ==> NEWGRP

UPDATED:          15JUN01 01:04 BY JSMITH

----- OUTPUT MANAGEMENT INFORMATION -----

PROCESSOR TO USE FOR MOVE ACTION:      M (M/G)
PROCESSOR TO USE FOR TRANSFER ACTION:   G (M/G)

      S - Browse Symbolics              L - List Processor
      U - Update Symbolics

GENERATE PROCESSOR ==> GCIINBL      FOREGROUND EXECUTION
DELETE PROCESSOR   ==> DLODNNN      ==> Y (Y/N)
MOVE PROCESSOR     ==> MLODNNL      ==> Y (Y/N)

```

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.

Note: In effect, there is only one processor group for type PROCESS. These “processors that process processors” (GPPROCSS and DPPROCSS) are hardcoded in Endeavor. 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 shown.

The use of the Processor Group Definition panel varies by processing option:

With This Option	Use This Panel to
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.

Once you have entered the necessary information on the panel, press ENTER to perform the requested processing.

5.5.1 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.

Field	Description
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.

Field	Description
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	<p>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.</p> <p>This feature is turned on at the system definition level. For more information regarding element registration, refer to the <i>Administration Guide</i>.</p>
Description	Description of the processor group.
Next Prcs Group	<p>Name of the processor group at the next map location. If this processor group was defined to Endeavor:</p> <ul style="list-style-type: none"> ▪ 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. <p>You can override these default values in update mode.</p>
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.

5.5.2 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.

5.5.2.1 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 Endeavor 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 Endeavor to execute the move, not the generate, processor.

Note: You can select either a move or a generate processor only when transferring from one Endeavor location to another. When transferring from an archive data set to Endeavor, or when using a move processor with a TRANSFER action, Endeavor does not allow you to rename elements at the target location.

5.5.2.2 Option Fields

The options that appear on Processor Group Definition panels may be used as indicated below. These options are not available for type Process or for processor groups named *NOPROC*.

Option	Description
S--Browse Symbolics	Appears on Processor Group Definition panels for display, delete, create, and update processing. Used to access a Processor Group Symbolics panel (described in the following section).
L--List Processor	Appears on Processor Definition panels for display, delete, create, and update processing. Used to access a Processor Display panel (described in "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 (described in the following section).

5.5.2.3 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. (Endevor converts one or more blanks in these fields to ***NOPROC***.)

5.5.2.4 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.

5.6 Processor Group Symbolics Panel

Symbolics panel allows you to change the default and override values for the processor's symbolic parameters.

```

UPDATE _____ PROCESSOR GROUP SYMBOLICS _____ ROW 1 OF 14
COMMAND ==>>>                                SCROLL ==>> PAGE
CURRENT ENV: TEST      STAGE ID: Q      SYSTEM: FINANCE      TYPE: COBOL
PROCESSOR GROUP: COBNBL
DESCRIPTION:          COBOL COMPILE AND LINK, LISTING IS STORED
PROCESSOR:           GCIINBL
LOAD LIBRARY:        ENDEVOR.STG1.NDVR.PRCLOAD
    To override symbolics   : Enter "0" and supply new value
    To reset to Default value: Enter "--"
    To exit without saving  : Enter CANCEL on the command line
SYMBOLIC -/O VALUE
COBLIB   — SYS1.VSCLLIB
COBSTPLB — SYS1.VSCOLIB
CSYSLIB1 — ENDEVOR.STG1.NDVR.COPYLIB
CSYSLIB2 — ENDEVOR.STG2.NDVR.COPYLIB
EXPINC   — N
LISTLIB  — ENDEVOR.STG1.NDVR.LISTING

```

5.6.1 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.

5.6.2 Symbolic Identification Fields

The three remaining fields on this panel provide information about the symbolics defined for this processor.

Field	Description
Symbolic	Name of the symbolic in the PROC statement of the processor.
-- (dash)/O	Indicates the status of the value of the symbolic. -- (dash)--Default value for the symbolic. This is the value assigned to the symbolic in the PROC statement contained in the processor. O--Override value for the symbolic.

Field	Description
Value	Value to be assigned to the symbolic during the next run of the processor.

5.7 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. Endeavor 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.

```

DISPLAY - LOAD LIBRARY: ENDEVOR.DEVCLS2.LOADLIB -----CHARS 'MACLIB' FOUND
COMMAND ==> SCROLL ==>
ENVIRONMENT SYSTEM SUBSYS ELEMENT TYPE STG VV.LL DATE TIME
ENDEVOR FINANCE ACCTREC GASM PROCESS 1 01.12 15JUN01 20:48
-----
:
***** TOP OF DATA *****
/*-----*
/* GASM: Endeavor processor for assemblies *
/* see type definition for ASMPGM and related processor groups *
/* every symbolic is more or less self explanatory except for *
/* MACLIB, SMACLIB1, and SMACLIB2. Processor Groups should *
/* override these symbols for other macro libraries which can *
/* be substituted (for example, INFO compiler product *
/* macro libraries. *
/*-----*
//GASM PROC RENT=RENT, whether to check rent code
//GASM PROC RENT=RENT, whether to check rent code
// REUS=REUS, load reusability (link step)
// AUTH=0, currently ignored
// LOADLIB=LOADLIB, loadlib last qualifier (link)
// VIO=VIO, unit= for temp dsns
// LINK=YES, whether to link or not
// LET=NOLET, for link step

```

This panel displays:

- PROCESSOR FOOTPRINT fields, that provide footprint information about the processor.
- A processor listing.

All fields are display-only.

Appendix A. Sample Processors

A.1 Sample Processor Overview

The processor examples in this appendix demonstrate the use of Endeavor utilities, keywords, and user-defined symbolics within the processors. These capabilities allow you to more easily convert your own JCL and PROCs into Endeavor processors. User-defined symbolics can then be overridden by the administrator during processor group definition.

A.2 Converting 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.igual.JCL library as member BC1PDSIN. It can be copied into your PROCs and modified to include or exclude data sets as appropriate. For more information, see 3.2, “BC1PDSIN Utility” on page 3-4

- 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 the AllFusion™ Endeavor Change Manager for CA-Panvalet Interface or the AllFusion™ Endeavor Change Manager for CA-Librarian interface, the CONWRITE utility will expand ++INCLUDE or -INC statements in the source. For more information, see 3.13, “CONWRITE Utility” on page 3-43.

There is a sample CONWRITE step in the iprfx.igual.JCL library as member CONWRITE. This sample CONWRITE step uses the &EXPINC and &MONITOR symbolics. You can define the following values for these symbolics 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=*nn*** to each step.

The MAXRC keyword is added to each step on the EXEC statement. It tells Endeavor of the highest acceptable OS return code (RC) for the step. If the

utility returns an OS return code higher than the value coded, Endeavor will set a failed flag on for the element and issue an Endeavor 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 3.10, “**CONLIST Utility**” on page 3-18.

There is a sample **CONLIST** step in the iprfx.igual.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** symbolic. If you add **LISTLIB=*listing library name*** to your PROC statement, Endeavor stores the listing in the *listing library name* as coded. If you add **LISTLIB=NO** to your PROC statement Endeavor prints the listing.

- f. Ensure that all user symbolics are defined on the PROC statement.

A.3 Generate Processors

This section contains sample generate processors. Samples are provided in `iprfx.igual.JCLLIB`. These processors are:

Processor Name	Description
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 Endeavor to manage. It is used as a generate process for ADD and UPDATE, and a move process for MOVE and TRANSFER. For more information on load module support, see the <i>Utilities Guide</i> .

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` symbolics 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` symbolic 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` symbolic 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 in 2.2, "Suggested Processor Naming Conventions" on page 2-3. Each sample is introduced with an expansion of its identifier. For example, `GCIINBL` is introduced as **Generate COBOL II, No database, for Batch, and create a Load module**.

A.3.1 GCIIDBL

Generate Cobol II, for DB2, for Batch, and create a Load 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='SYSD2LIB',
//          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=133,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
//SYSTEM  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))

```

```

/*****
/**   COMPILER THE ELEMENT                               **
/*****
//COMPILE EXEC PGM=IGYCRCTL,COND=(0,LT),MAXRC=4,
// PARM='&PARMC0B'
/* 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)
//C1LLIB0 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)
//**
```

A.3.2 GCIINBL

Generate Cobol II, No database, for Batch, and create a Load 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 OS/390

```

//*****
//**
//** 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=133,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)
//**

```

A.3.3 GLNKNBL

Generate **Link**-edit only No database for **Batch** and create a **Load** 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=133,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)
//C1LLIB0 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)
//*

```

A.3.4 GASMNBL

Generate **Assembler**, No database, for **Batch**, and create a **Load** 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,
// LSYLIB1='&PROJECT..&GROUP.&C1ST..LOADLIB',
// LSYLIB2='&PROJECT..&GROUP.&C1ST2..LOADLIB',
// LSYLIB3='&PROJECT..&GROUP.&STG3..LOADLIB',
// LSYLIB4='&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=133,BLKSIZE=0,DSORG=PS)
//C1INIT02 DD DSN=&&LNKLIST,DISP=(,PASS),
// UNIT=&WRKUNIT,SPACE=(TRK,(10,10),RLSE),
// DCB=(RECFM=FBA,LRECL=133,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)

```

A.3.5 LOADONLY

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 an Endeavor 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))
//OUTDD DD DSN=&LOADLIB1,DISP=SHR,FOOTPRNT=VERIFY
//INDD 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)
/*
/**

```

A.4 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.igual.JCLLIB. These processors are:

Member Name	Which Stands For...
DLODDNL	Delete Load modules, for DB2 , No (All) operating environments, No (All) output types.
DLODNNL	Delete Load modules, No (All) databases, No (All) operating environments, No (All) output types.

These processors can be used as Stage 1 or Stage 2 delete processors. To change the PROC statement symbolic values, override the data set names during processor group definition.

If you do not want to store listings, change the &LISTLIB symbolic to NO.

A.4.1 DLODDNL

Delete **LOad** modules, for **DB2**, **No** (All) operating environments, **No** (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
//*
//DELDDBRM 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

```

A.4.2 DLODNNL

Delete **LO**ad modules, **No** (All) databases, **No** (All) operating environments and Listings

```
//*****  
//*  
//* 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
```

A.5 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. For more information, see A.3, “Generate Processors” on page A-5.
- 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. These samples are provided in iprfx.igual.JCLLIB. These processors are: 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 symbolic is overridden during processor group definition to read **NO**, Endeavor 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 symbolic with any value except **components**.

A.5.1 MLODDNL

Move **Load** Modules, for **DB2, No (All)** operating environments, to **Load** 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.CONLIB',
//
// 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='&C1STAGE.', CURRENT STAGE
//
// STG2='&C1STAGE.', TO STAGE
//
// PLAN=&C1ELEMENT,
//
// 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
//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
//SYSIN    DD *
          COPY O=OUTDD,I=INDD
          SELECT MEMBER=((&C1ELEMENT, ,R))
//*****
//* MOVE THE COMPONENT LIST *
//*****
//*
//MOVECL   EXEC PGM=BC1PMVCL,MAXRC=04,COND=(4,LT)
//*
//*****
//* 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
//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)

```

```
//C1LLIB1 DD DSN=&LISTLIB1,DISP=SHR
//C1LLIB0 DD DSN=&LISTLIB2,DISP=SHR
//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)
//*
```

A.5.2 MLODNNL

Move Load Modules, No (All) databases, No (All) operating environments, with associated component lists and Listings.

```

//*****
//*
//* 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='&C1STAGE.',      CURRENT STAGE
//          STG2='&C1STAGE.',      TO STAGE
//          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
//SYSIN    DD *
           COPY O=OUTDD,I=INDD
           SELECT MEMBER=((&C1ELEMENT,,R))
//*****
//* MOVE THE COMPONENT LIST
//*****
//MOVECL   EXEC  PGM=BC1PMVCL,COND=(0,NE)
//*****
//* COPY & STORE THE LISTINGS IF: &LISTING=LISTING LIBRARY
//*****
//COPYLIST EXEC PGM=CONLIST,MAXRC=0,PARM=COPY,COND=EVEN,
//          EXECIF=(&LISTLIB,EQ,YES)
//C1LLIB1  DD DSN=&LISTLIB1,DISP=SHR
//C1LLIB0  DD DSN=&LISTLIB2,DISP=SHR

```

```
//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)
//*
```

A.6 Other Processors

These processes are provided as an example of how to manage other types of source. Use these processors as a guide

Member name	Processor Description
ALIASDEL	Deletes alias load modules associated with the current element
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 Advantage CA-Easytrieve
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 Advantage CA-IDEAL
IMSDBD	Sample IMS processor
JCLCHK	Sample processor for Unicenter CA-JCLCheck
JOBSCAN	Sample JOBSCAN Processor
SDFIIMAP	Generate Processor for SDF II maps

Appendix B. Unsupported Parameters

B.1 General Restrictions

As discussed previously, Endeavor processors are written using standard OS JCL syntax--with a few restrictions. Most JCL parameters are supported in Endeavor processor statements.

Selected JCL parameters are unsupported by Endeavor and, if used, are ignored by the system or cause an error. A message is returned when Endeavor encounters one of these parameters:

- If the parameter is ignored, the message reads:

```
C1X0243I STMT statement-no IGNORED VALUE parm-value
```

- If the parameter causes an error, the message reads:

```
C1X0244E STMT statement-no INVALID--VALUE  
STARTING WITH parm-value IS CURRENTLY  
UNSUPPORTED BY ENDEVOR FOR OS/390
```

The following JCL keyword parameters are not supported by Endeavor processors. This list was compiled as of MVS/SP version 2. If you use a parameter from this list, the appropriate message is returned.

The parameters are listed by statement type (EXEC= or DD=) in which they are normally used.

B.2 EXEC Statement Parameters

Endevor 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

B.3 DD Statement Parameters

Endevor 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
SPIN
SUBSYS=PANV

B.4 DCB Subparameters

Endevor processors support the DCB parameter, but not these DCB subparameters:

BUFIN	INTVL
BUFMAX	IPLTXID
BUFOFF	MODE
BUFOUT	NTM
BUFSIZE	PCI
CODE	PRTSP
CPRI	RESERVE
CYLOFL	RKP
FRID	STACK
FUNC	THRESH
GNCP	TRTCH

B.5 DDNAME Subparameters

Endevor processors support the DDNAME parameter, but not these DDNAME subparameters:

DSID	QNAME
FLASH	REFDD
MODIFY	SPLIT
MSVGP	SUBALLOC
OUTPUT	DDNAME

Index

Special Characters

& (ampersand) 2-10
&& (double ampersand) 2-14

A

Abend conditions, testing for 2-28, 2-29
ABENDCC control statement 2-28
ACM
 BSTCOPY 3-8
 FASTCOPY 3-10
 identifying additional relationships 3-32
Action Summary Report 3-12
Actions invoking processors 1-5
Audit trails 3-7
Automatic footprinting 2-5

B

BACKOUT utility 2-9
BC1PDSIN utility 3-4
BC1PTMP0 utility 3-5—3-6
BC1PXFPI utility 3-7
BSTCOPY utility 2-5, 3-8—3-11
Bypassing footprint creation 2-6

C

C1BM3000 utility 3-12—3-13
C1DEFLTS 2-12
C1PRMGEN utility 3-14—3-15
Classifying Processors 4-1
CLIST 3-5
Component list
 active 3-43
 element type 3-44
 existing 3-43
 format 3-44
 including related elements 3-26

Component list (*continued*)
 input to processes 3-43
 write data 3-43
 write data to external location 3-43
Component monitoring 2-9
Compressing temporary data sets 3-20
CONAPI utility 3-16
CONDELE utility 3-17
Conditional execution 2-7
CONLIST utility 3-18—3-25
 COPY 3-22
 DELETE 3-22
 footprints 2-5
 listing guidelines 3-23
 PRINT 3-21
 PRNTMBR 3-21
 PRTMBR 3-21
 STORE 3-20
CONRELE utility 3-26—3-31
 commands 3-26—3-29
 SET ERROR RETURN CODE 3-29
 example 3-29
 footprints 2-5
 overview 3-26
 producing control statements 3-32
 RELATE COMMENT 3-29
 RELATE ELEMENT 3-26
 RELATE MEMBER 3-28
 RELATE OBJECT 3-28
 SET ERROR RETURN CODE 3-29
CONSCAN utility 3-32—3-42
 ACM 3-32
 CONRELE control statements 3-38
 error message 3-32, 3-42
 examples 3-34, 3-38
 exclusion group 3-34
 Selection Group 3-38
 excluding source 3-33
 exclusion group 3-33

CONSCAN utility (*continued*)

- format 3-32
- input control statements 3-32
- output 3-38
- PARMSCAN data set 3-32
- PARMSCAN statements 3-32
- producing CONRELE control statements 3-32
- return code 3-32
- sample processor 3-41
- scan rule processing 3-40
- scan rules 3-32
- selecting data 3-34
- selecting data syntax 3-35—3-37
- selection criteria 3-32
- selection groups 3-34, 3-35

Consolidating temporary data sets 3-20

CONWRITE utility 3-43—3-50

- expanding INCLUDEs 3-47
- extended form 3-43
- extended form JCL 3-45
- extended form processing 3-45
- INCLUDE statements 3-43
- parameter list 3-49
- PARM=EXPLINCL 3-48
- return code 3-49
- specifying additional INCLUDE libraries 3-48
- standard form 3-43
- standard form JCL 3-44
- standard form processing 3-44
- syntax 3-45
- truncating element records 3-44
- types of component list data 3-43
- user exit 3-49
- validating record length 3-44
- WRITE ELEMENT statement 3-48

COPY option for CONLIST 3-22

Copying aliases 3-8

Copying from input libraries 3-22

Creating footprints 2-5

D

- Debugging IF-THEN-ELSE logic 2-29
- Decompresses and prints a member 3-21
- Default processor group 1-4
- Delete processors 3-17
- Deleting a member 3-22
- Disabling FASTCOPY for BSTCOPY 3-10
- Double ampersand (&&) 2-14
- DPPROCSS processor 4-3, 5-12, 5-17

E

- Element processing 3-17
- Element types
 - default processor group 1-4
 - PROCESS 4-3
- Elements 2-6
- Endevor
 - errors 2-6
 - ISPF dialog 3-5
 - return code 2-6
 - symbolics 2-10—2-21
 - && (double ampersand) 2-14
 - periods 2-14
 - reserved names 2-14
 - substringing 2-19—2-21
- Error messages
 - CONLIST 3-18
 - CONSCAN 3-32, 3-42
 - CONWRITE 3-46
- EXECIF keyword 2-7
 - Endevor symbolics 2-7
 - operators 2-7
 - site symbolics 2-7
 - symbolic substitutions 2-10
 - user symbolics 2-7
 - values 2-7
- Executing
 - actions in a processor 3-12
 - processors from an Endevor ISPF dialog 3-5
 - TSO commands in a processor 3-5
- Expanding symbolics 3-14

F

- Feature 2-9
- Footprints
 - See also* FOOTPRNT keyword
 - bypassing creation 2-6
 - CONLIST 2-5
 - CONRELE 2-5
 - CONWRITE 2-5
 - creating 2-5
 - errors 2-6
 - NONE 2-6
 - object modules 2-5
 - verification return code 2-6
 - VERIFY 2-6
- FOOTPRNT keyword
 - See also* Footprints
 - CREATE 2-5

FOOTPRNT keyword (*continued*)

- display processor 5-19
- load modules 2-5

G

GPPROCSS processor 4-3, 5-12, 5-17

I

IBM utilities

- \$IEBCOPY 3-10
- IEBCOPY 2-32
- IEHMOVE 2-32
- IKJEFT01 2-32, 3-5

IBM-defined expressions

- ABEND 2-28
- ABENDCC 2-28
- ABEND 2-29
- RUN 2-29
- RC 2-28
- RUN 2-29

IF-THEN-ELSE processor flow control statement

- control order of SCL statements 2-22
- debugging 2-22, 2-29
- IBM-defined expressions 2-27
- IF statement location 2-24
- keywords 2-23
- multiple statements 2-25
- naming 2-22
- nesting 2-26
- run time 2-22
- stacking on the same statement 2-26
- syntax 2-22
- trace facility 2-29

IKJEFT01 2-32, 3-5

In-stream data 2-21, 3-14

INCLUDE statement expansion

- CONWRITE 3-47
- PARM=EXPLINCL 3-48
- PARM=EXPLINCL JCL 3-48
- using WRITE ELEMENT statement 3-48

Including entities in a component list 3-26

Initializing sequential data sets 3-4

Installing footprints in object modules 3-7

Invoking actions and processors 1-5

J

Job failures 2-6

K

Keywords 2-5—2-9

- BACKOUT 2-9
- EXECIF 2-7
- FOOTPRNT 2-5
- list 2-5
- MAXRC 2-6
- MONITOR 2-9

L

Load

- libraries 3-8
- modules 2-8, 3-9

M

Manage output listings 3-18

Managing Processors 4-1

MAXRC keyword 2-6—2-7

- overview 2-6

MONITOR 2-9

N

Naming conventions for processors 2-3

Non-Endevor programs 2-32

O

Object modules 2-5, 3-7

Operators 2-7

Output libraries 3-17, 3-20

P

Packages

- backout 3-8
- backout information 2-9
- ID 3-12

Panels

- creating processor groups 5-5
- displaying processor groups 5-5
- Processor Display Panel 5-19
- Processor Group Definition 5-12
- Processor Group Definition Panel 4-4
- Processor Group Symbolics 5-17
- Type Definition 4-3
- predefined processors 4-3
- updating processor groups 5-5

PARMSCAN parameter data set 3-32—3-34
PDSMAN 3-10
Periods in ENDEVOR-specific symbolic 2-14
PRINT
 member 3-21
 temporary data sets 3-21
Processor group 5-17
 change 4-3
 defining element types 1-4
 Definition Panels
 creating 5-5
 types other than PROCESSOR 5-12
 updating 5-5
 delete 5-2
 description 5-17
 DPPROCSS 5-12, 5-17
 generate 5-2
 GPPROCSS 5-12, 5-17
 inventory area 5-17
 load library 5-17
 move 5-2
 naming conventions 5-2
 overview 1-3, 5-2
 PROCESS 4-3
 Processor Group Definition Panel 4-4
 processor type 5-12, 5-17
 processor within a group 5-17
 relationship to processors 1-3
 symbolics 5-17
 identification fields 5-17
 Panel 5-17
 types 5-2
 user-defined symbolics 5-4
Processors
 capabilities 2-4
 classifying and managing 4-1
 CONRELE 3-26
 control at execution 2-22
 customizing JCL at run time 2-22
 defining 1-4
 delete 3-17, 4-3
 display panel 5-19
 DPPROCSS 4-3
 executing actions 3-12
 executing TSO commands 3-5
 executing under Endeavor ISPF dialog 3-5
 features 2-4
 flags 2-6
 footprint creation 2-5
 generate 4-3
 generate load modules 2-8

Processors (*continued*)
 GPPROCSS 4-3
 highest acceptable return code 2-6
 implementing 4-5
 in-stream data 2-21
 invoking actions 1-5
 issuing API calls 3-16
 keywords 2-5
 listing 5-19
 maintain processors 4-6
 move 2-6, 4-3
 naming conventions 2-3
 overview 1-2
 predefined processors 4-3
 processor-failed flag 2-6
 relationship to processor groups 1-3
 return code 2-6
 symbolics 2-11
 tailoring using symbolics 2-10
 terminate 2-6
 writing 2-3

R

RC processor flow control statement 2-28
RELATE COMMENT command 3-29
RELATE ELEMENT command 3-26
RELATE MEMBER command 3-28
RELATE OBJECT command 3-28
Removing members from output libraries 3-17
Reports
 Action Summary Report 3-12
 SCANPRT 3-35
Return codes
 BC1PTMP0 3-6
 CONSCAN 3-32
 CONWRITE 3-44, 3-49
 Endevor 2-6
 highest acceptable 2-6
 MAXRC 2-6
REXX EXEC 3-5

S

Sample JCL
 BACKOUT 2-9
 BC1PDSIN 3-4
 BC1PTMP0 3-5
 C1BM3000 3-12
 C1PRMGEN 3-14
 CONAPI 3-16

Sample JCL (*continued*)

- CONDELE 3-17
- CONLIST 3-23
- CONRELE 3-26
- CONSCAN 3-41
- CONWRITE extended form 3-45
- CONWRITE standard form 3-44
- CONWRITE with INCLUDE expansion 3-48
- EXECIF 2-8
- footprinting load modules 2-5
- FOOTPRNT=VERIFY 2-6
- MAXRC 2-7
- Sequential data sets, initializing 3-4
- SET ERROR RETURN CODE 3-29
- Site requirements 2-11, 2-12
- Site symbolics
 - symbolics 2-12—2-14
- STOPRC keyword 2-6
- STORE option for CONLIST 3-20
- Symbolics
 - Endevor 2-7, 2-14
 - EXECIF 2-10
 - guidelines 2-11
 - in-stream data 2-10
 - numeric operands 2-10
 - overriding default values 2-11, 2-12
 - site symbolics 2-7, 2-12—2-14
 - substringing 2-19—2-21
 - errors 2-19
 - length 2-19
 - pad 2-19
 - start 2-19
 - user symbolics 2-7
 - user-defined 2-11—2-12, 2-14
 - validated 2-11
 - values for 2-15
- Syntax
 - CONRELE EXAMPLE 3-29
 - CONSCAN exclusion group 3-33—3-34
 - CONSCAN select data 3-35—3-37
 - CONWRITE 3-45
 - EXECIF 2-7
 - RELATE COMMENT 3-29
 - RELATE ELEMENT 3-26
 - RELATE MEMBER 3-28
 - RELATE OBJECT 3-28
 - SET ERROR RETURN CODE 3-29
- SYSPRINT 3-11
- System abend testing and conditions 2-28

T

- Terminate processing 2-6
- Testing conditions and system abends 2-28—2-29
- Transportable footprints 3-7
- TSO
 - BC1PTMP0 3-5
 - inactive 3-6

U

- User exit program 3-49—3-50
- User-defined
 - completion code 2-28
 - data set 3-32
 - symbolics 2-11—2-12
 - symbolics in processor groups 5-4
- Utilities
 - BC1PDSIN 3-4
 - BC1PTMP0 3-5—3-6
 - BC1PXFPI 3-7
 - BSTCOPY 2-5, 3-8—3-11
 - C1BM3000 3-12—3-13
 - C1PRMGEN 3-14—3-15
 - CONAPI 3-16
 - CONDELE 3-17
 - CONLIST 2-5, 3-18—3-25
 - CONRELE 2-5, 3-26—3-31
 - CONSCAN 3-32—3-42
 - CONWRITE 3-43—3-50
 - overview 3-2

W

- Writing processors 2-3

