

AllFusion™ Endeavor® Change Manager

Implementation 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. Basic Concepts	1-1
1.1 Overview	1-2
1.1.1 What is Endeavor	1-2
1.1.2 What Can You Do with Endeavor?	1-2
1.2 Implementing for Source, Output, or Configuration Management	1-4
1.3 The Software Life Cycle	1-5
1.3.1 The Endeavor Life Cycle	1-5
1.3.2 Life Cycle Requirements	1-5
1.3.3 The Endeavor Sample Application	1-6
1.3.4 Basic Operations	1-6
1.3.5 Integration Stage Operations	1-7
1.4 Endeavor Logical Structure	1-8
1.4.1 Using the Inventory Structure	1-8
1.4.2 Environment	1-8
1.4.3 Stage	1-9
1.4.4 System	1-9
1.4.5 Subsystem	1-9
1.4.6 Type	1-10
1.4.7 Element	1-11
1.4.8 Element Classification	1-11
1.4.9 More About Elements	1-12
1.5 Endeavor Libraries	1-13
1.5.1 Where to Allocate Your Libraries	1-14
1.6 Working with Elements	1-16
1.6.1 Endeavor Actions and Availability	1-16
1.6.2 Actions by Job Function	1-17
1.6.3 Creating Executable Forms of Elements	1-18
1.6.4 Correlating Source with Executables	1-18
1.6.5 Packages	1-18
1.7 Security	1-19
1.7.1 Data Set Security	1-19
1.7.2 Functional Security	1-19
Chapter 2. Implementation and the Organization	2-1
2.1 Overview	2-2
2.2 Organizational Considerations	2-3
2.2.1 Accessing Receptiveness to Change	2-3
2.2.2 Developing a Strategy	2-3
2.3 Selecting an Implementation Team	2-5

2.3.1	Required Technical Knowledge	2-5
2.3.2	Required Organizational Involvement	2-5
2.3.3	Public Relations Requirements	2-6
2.4	Laying the Groundwork	2-7
2.4.1	Prepare a Presentation	2-7
2.4.2	Tailor the Presentation	2-7
2.5	Make the Presentation	2-8
2.5.1	Introducing Endeavor to an Organization	2-8
2.5.2	After the Presentation	2-9
2.6	Using a Pilot Application	2-10
2.6.1	Characteristics of the Pilot Application	2-10
2.7	Building Momentum	2-11
2.8	Summary	2-12
Chapter 3.	Implementing Endeavor	3-1
3.1	Overview	3-2
3.1.1	Endeavor Implementations	3-2
3.1.2	Assumptions	3-2
3.1.3	The Software Life Cycle	3-2
3.2	Implementation Alternatives	3-3
3.2.1	A Phased Implementation	3-3
3.2.2	A Full Implementation	3-3
3.3	Implementing for Source Management	3-4
3.3.1	What's Involved?	3-4
3.4	Step 1: Define Your Software Life Cycle	3-5
3.4.1	Decide Stages for Endeavor Control	3-5
3.4.2	Define Environments	3-5
3.4.3	Define the Map	3-6
3.4.4	"Normal" Development Procedure	3-6
3.4.5	Quick Fix or Emergency Procedure	3-7
3.4.6	Mapping for Parallel Development	3-7
3.4.7	Define Master Control Files	3-7
3.4.8	Environment Implementation Checklist	3-8
3.5	Step 2: Analyze the Inventory	3-9
3.6	Step 3: Define the Logical Structure	3-10
3.6.1	Implement the Logical Structure	3-10
3.6.2	Define Systems	3-11
3.6.3	Define Subsystems	3-11
3.6.4	Define Types	3-11
3.6.5	Type Naming Conventions	3-11
3.6.6	Verifying the Definitions	3-12
3.7	Step 4: Define the Physical Structure	3-13
3.7.1	Naming Conventions	3-13
3.7.2	Using Symbolics to Define Endeavor Libraries	3-13
3.7.3	Occurrence	3-14
3.7.4	Sizing	3-14
3.7.5	Format	3-14
3.7.6	Element Storage Formats	3-15
3.7.7	Where Libraries Are Defined	3-16
3.7.8	More About Base and Delta Libraries	3-16
3.7.9	Backup and Recovery	3-16

3.8 Step 5: Define CCIDs (Optional)	3-17
3.8.1 CCID Definition Data Set	3-17
3.9 Step 6: Enable Package Processing	3-18
3.9.1 Set Up a Naming System	3-18
3.9.2 Define and Allocate a Package Data Set	3-18
3.9.3 Enable Component Validation	3-19
3.9.4 Approver Groups	3-19
3.9.5 What to Do Next	3-19
3.10 Implementing for Output Management	3-20
3.11 Step 7: Define and Allocate Output Libraries	3-21
3.11.1 Output Library Information	3-21
3.11.2 More About Output Libraries	3-22
3.11.3 Output Library Allocation	3-22
3.12 Step 8: Define Processors	3-23
3.12.1 Writing Processors	3-23
3.12.2 Processor Groups and Types	3-23
3.12.3 Managing Processors	3-23
3.12.4 Determining Type Processing Sequence	3-23
3.12.5 Sample Processors	3-23
3.12.6 What to Do Next	3-24
3.13 Implementing for Configuration Management	3-25
3.14 Step 9: Enable ACM	3-26
3.15 Final Implementation Steps	3-27
3.16 Step 10: Enable Security	3-28
3.17 Step 11: Load the Inventory	3-29
3.17.1 Overview	3-29
3.17.2 Enable the ACM Baseline	3-29
3.18 Step 12: Provide Internal Training	3-30
3.19 Step 13: Go into Production	3-31
Appendix A. Upgrading to Endeavor 4.0	A-1
A.1 New Features	A-2
A.2 Changes for Endeavor 4.0	A-3
A.3 Recommendations for Endeavor 4.0	A-4
Glossary	X-1
Index	X-13

Chapter 1. Basic Concepts

1.1 Overview

This guide introduces basic concepts for AllFusion Endeavor Change Manager (formerly known as Endeavor for OS/390, and hereafter referred to as simply Endeavor) and explains how to perform everyday tasks that are important for system administrators to be familiar with before implementing Endeavor. This guide is part of a comprehensive documentation set that fully describes the features and functions of Endeavor. 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.

This guide contains information about Endeavor from Computer Associates and related products on the OS/390 and z/OS operating systems. The Endeavor documentation set describes procedures and, in some cases, JCL, for the OS/390 environment. This same procedure and JCL are applicable for the z/OS environment.

This chapter introduces basic Endeavor concepts. You should be familiar with these concepts before implementing Endeavor.

- Implementing for Source, Output, or Configuration Management
- The Software Life Cycle
- Endeavor Logical Structure
- Endeavor Libraries
- Working with Elements
- Security

1.1.1 What is Endeavor

Endeavor is an integrated set of management tools that is used to automate, control, and monitor your software development life cycle. Endeavor is implemented and run under z/OS and OS/390, within the TSO ISPF environment, and in batch.

1.1.2 What Can You Do with Endeavor?

Using Endeavor, you can do the following:

- Compare and track your changes against production automatically, creating an online change history. This speeds up the debugging process and enables you to always know what was changed, by whom, and why.
- Prevent conflicting changes to the same system component.
- Browse and manipulate all components relating to an application from a single screen, saving you time and ensuring that changes are complete.
- Create executables automatically.
- Ensure that the source, executable, and any other form (for example, listings) of an element correspond.

- Apply the same procedures (including automating compiles or performing impact analyses and standards checking) to any component type, thereby simplifying this process.
- Put change packages and approvals online, eliminating the need for change-related paperwork.
- View or retrieve prior levels of any element.
- Report on element definition, content, and change history.
- Enforce change control procedures.

1.2 Implementing for Source, Output, or Configuration Management

Endevor provides different levels of functionality.

Level of Functionality	Description
Inventory management	This function creates and maintains the Master Control File definitions. All implementations of Endevor use inventory management. (This level of functionality is standard with Endevor.)
Source management	This function manages element source in base and delta libraries. All implementations of Endevor use source management. (This level of functionality is standard with Endevor.)
Output management	This function manages the outputs created by Endevor processors in source output, output, listing, load, and object libraries. (This level of functionality is optional with Endevor.)
Configuration management	This function assures that executables include the current versions of all input components, and tracks changes to input components over time. (This level of functionality is optional with Endevor.)

See Chapter 3, “Implementing Endevor” on page 3-1 for step-by-step instructions on implementing each level of functionality.

1.3 The Software Life Cycle

Endevor allows you to automate and control the movement of software through your software life cycle.

Software life cycles are site-specific. A representative life cycle might consist of the five phases shown below. Note, however, that Endevor can be implemented to adapt to any requirements of the software life cycle.

- *Development*, where programs originate and are developed
- *Test*, where programs are unit tested
- *QA*, where applications are system tested
- *Integration*, where fixes are applied to production code (This stage may also be known as *emergency*)
- *Production*, where production applications are stored

1.3.1 The Endevor Life Cycle

Four of the phases above can map to stages in the Endevor life cycle:

Phase	Endevor Stage Name	Endevor Stage ID
Test	Unittest	UT
QA	Quality Assurance	QA
Integration	Integration	INT
Production	Production	PRD

Note: For the remainder of this guide, the Endevor life cycle stage IDs will be used.

1.3.2 Life Cycle Requirements

Many sites use a four-stage life cycle for normal development activities. Therefore, a typical non-Endevor life cycle might contain the following stages:

DEV	UT	QA	PRD
-----	----	----	-----

In addition, many sites have a stage for handling quick fix (or emergency) or problem resolution situations without disrupting existing development and QA activities. This is the *integration* stage.

At other sites, program development takes place outside of Endevor. In this situation, a site may place the last three stages of the normal development life cycle—UT, QA,

and PRD—under the control of Endeavor. The site then adds the quick fix stage—INT—to the life cycle, creating the four-stage life cycle shown below:

UT	QA	INT	PRD
----	----	-----	-----

These are the four stages of the Endeavor life cycle that is used in the Endeavor sample application.

1.3.3 The Endeavor Sample Application

You can (and, if you are a new site, should) use the sample application as a guide to implement Endeavor. The sample application is included on the installation tape. See the *Installation Guide* for information about working with the sample application.

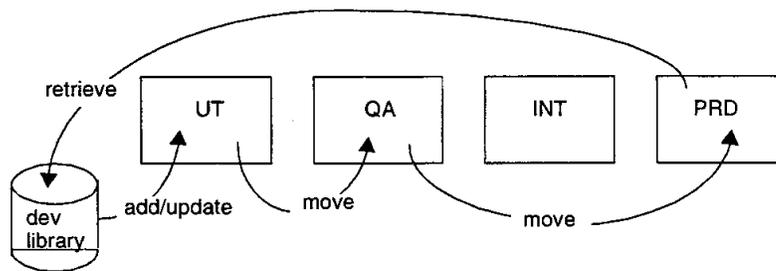
Implementing Endeavor uses the sample application in its step-by-step explanation of the implementation process.

1.3.4 Basic Operations

Normal change procedures include the following:

- Retrieving elements from production to a development library
- Making changes to elements
- Adding or updating elements into the test stage
- Moving elements to QA
- Moving elements back into production

The following diagram shows normal change procedures in a software life cycle.



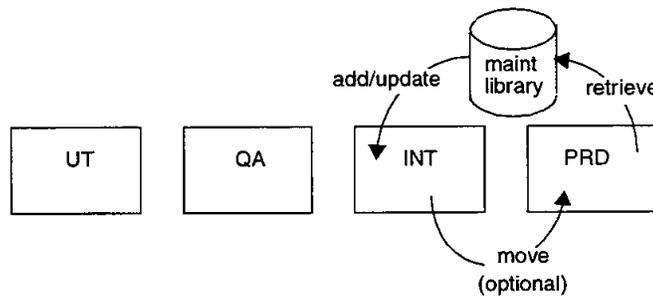
1.3.5 Integration Stage Operations

Integration stage procedures include the following:

- Retrieving elements from production
- Making changes to elements
- Adding or updating elements into the emergency stage
- Optionally, moving elements to production

Note: If you are dealing with an emergency fix, you may want the emergency libraries concatenated in front of your regular production environments, in your processors.

The following diagram illustrates emergency change procedures in a software life cycle.



1.4 Endeavor Logical Structure

Endeavor helps manage the software life cycle by providing a consistent and flexible logical structure for classifying software inventory. There are six components to this inventory structure: environments, stages, systems, subsystems, types, and elements. Environments, stages, systems, subsystems, and types are defined by the Endeavor administrator. Users act on elements. These terms are defined on the following pages.

1.4.1 Using the Inventory Structure

The Endeavor inventory structure allows a user to:

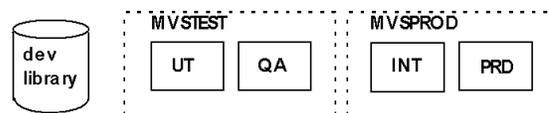
- Work with program modules without having to know where they are physically located or how they are compiled.
- List all the program components that make up an application, regardless of type.
- Determine the location(s) of an element simply by entering the element name on a display screen.
- Act on a cross-section of your program inventory. For example, Endeavor allows you to list all COBOL code in your shop, or promote an entire new release of the payroll application with a single command.

These are only some of the ways you can use an inventory structure to your advantage.

1.4.2 Environment

Environments are mapped to functional areas within an organization. For example, there might be separate development and production environments at your site. There is no limit to the number of environments that may be defined.

In the life cycle, assume the UT and QA stages in the life cycle are part of the development function. Production applications and their maintenance are part of a function called production. The administrator defines environment MVSTEST to include Stage UT and Stage QA, and environment MVSPROD to include Stage INT and Stage PRD. Development activities take place in development libraries, outside of Endeavor.



1.4.3 Stage

The term *stage* refers to physical locations in the software life cycle. There must be exactly two stages within each environment.

Stages have a name, representing their place in the life cycle (for example, UNITTEST), and an ID (a single alphanumeric character). Stages are referred to in this manual as Stage 1 (the first stage in an environment) and Stage 2 (the second stage in an environment).

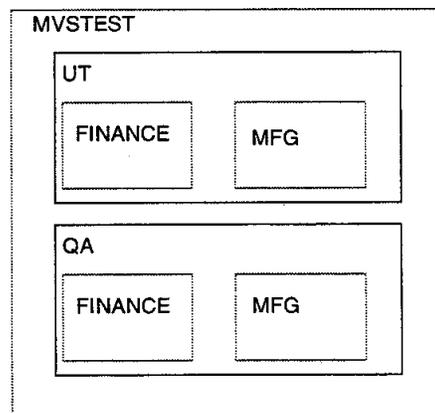
Stages can be linked together to establish unique promotion routes for program inventory within and between environments. These routes make up the map for a site.

1.4.4 System

Systems generally represent applications at a site. For example, a site may have both financial and manufacturing applications.

Each system must be defined to each environment in which it will be used. This definition makes the system available in both stages of an environment.

There are two systems in this example: FINANCE and MFG (manufacturing).



1.4.5 Subsystem

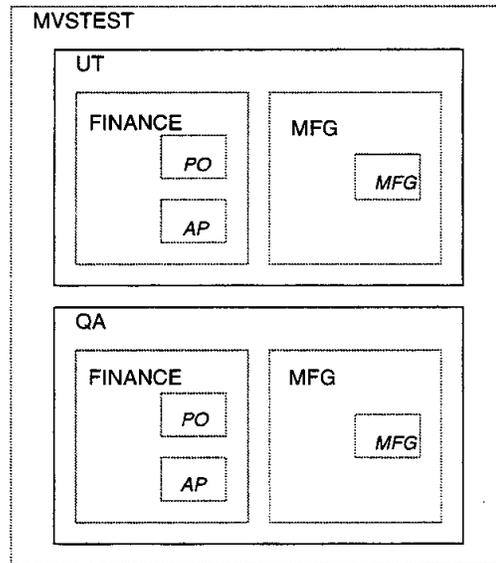
A *subsystem* is a logical application within a system. For example, the financial system might include a purchase order application (PO) and an accounts payable (AP) application. Keep in mind the following:

- There must be at least one subsystem per system.
- A subsystem must be defined to each system in which it will be used.

For example, if you plan to have subsystem PO within system FINANCE, and you define system FINANCE to several environments, then you must define subsystem PO to system FINANCE in each of those environments.

- A subsystem can have the same name as the system to which you define it.

In this example, system FINANCE has two subsystems, PO and AP. System MFG has one subsystem, MFG.

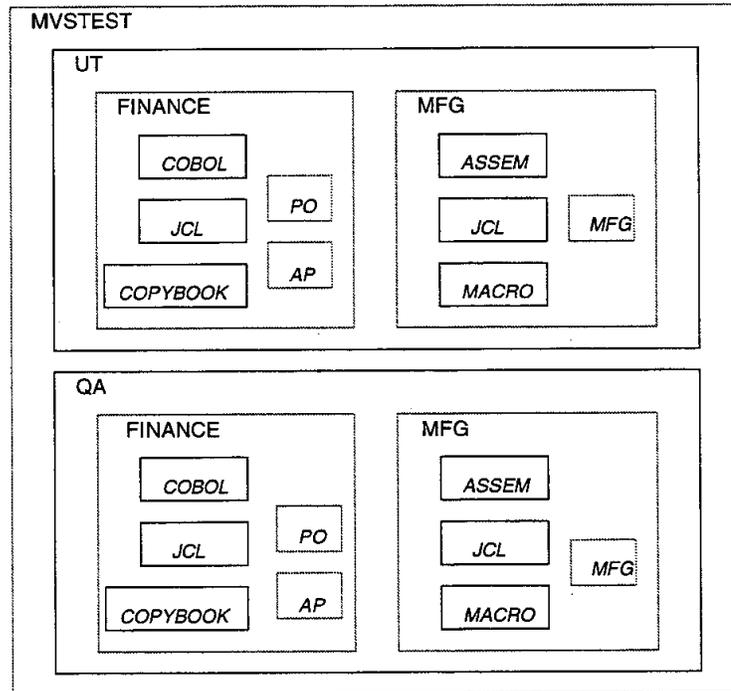


1.4.6 Type

A *type* is a category of source code. For example, you might create the following types: COBOL (for COBOL code), COPYBOOK (for copybooks), and JCL (for JCL streams). You must define a type to each system in each stage in which you want to use it. All subsystems defined to a system can use the types defined to that system.

Endeavor uses JCL streams called *processors* to automate the creation of executables such as program load modules. You must associate one or more *processor groups* with each type. Each processor group identifies the processors needed for a particular type of source. *User symbolics* make it possible to share processors across groups.

In the example below, system FINANCE has types COBOL (COBOL code), JCL (JCL streams), and COPYBOOK (copybooks). System MFG has types ASSEM (Assembler code), JCL, and MACRO (Macros).



1.4.7 Element

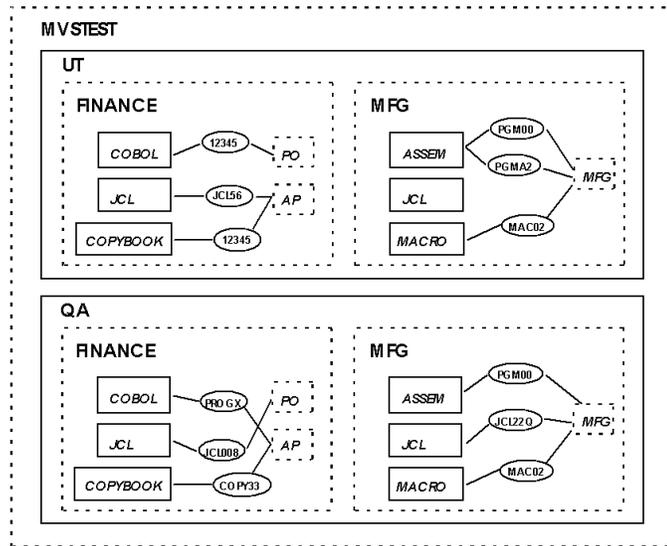
Elements are members in a partitioned data set, an AllFusion CA-Panvalet for OS/390 and z/OS data set (hereafter referred to as simply "Panvalet"), an AllFusion CA-Librarian for OS/390 and z/OS data set (hereafter referred to as simply "Librarian"), sequential data sets, or an HFS file that are placed under control of Endeavor. The element name is frequently the member name. Each element is classified by system, subsystem, and type. Its environment and stage determine its location in the software life cycle.

1.4.8 Element Classification

Endeavor classifies elements according to the inventory structure you set up. Each element is described uniquely in terms of the following:

- Location in the software life cycle. This is determined by the environment and stage where the element resides.
- Inventory classification. This is determined by the system, subsystem, and type with which the element is associated.

The diagram below illustrates element classification:



1.4.9 More About Elements

The diagram illustrates additional characteristics of elements:

- Two elements can share the same name across types. In Stage UT, system FINANCE, there are two elements named 12345: one is a COBOL program, one is a copybook. The elements are different types, however.
- The same element can exist at different stages in the life cycle. PGM00, an assembler program, exists in both Stage UT and Stage QA in system Manufacturing (MFG).
- The same element and type can be in two different subsystems of the same system.

1.5 Endeavor Libraries

To implement an inventory structure, certain libraries must first be defined and allocated (in the table below, the Endeavor Defaults Table is mentioned; see the *Installation Guide* for information about this, if necessary).

Library	Comments	Where/When Defined
Master Control File	There is one Master Control File (MCF) for every stage. A Master Control File stores system, subsystem, and type definitions, the names of the elements currently in that stage, and other information.	In the Endeavor Defaults Table, during implementation
Package Data Set	There is one package data set per site. Endeavor stores all packages and package-related information built at the site in this data set.	In the Endeavor Defaults Table, during implementation
CCID Validation Data Set	The CCID Validation data set defines CCIDs and the types requiring CCID use. This data set must be predefined in order to perform CCID validation.	In the Endeavor Defaults Table, during implementation
Base and Delta	Endeavor uses base and delta libraries, for each defined type, to store source code. Base libraries store a full copy of either the original code (if forward deltas are used), or the current code (if reverse deltas are used). With reverse delta format, base libraries are also referred to as "image" libraries. Delta libraries store changes made to the source. There must be at least one base and one delta library per environment. Generally, there is one set of libraries for each type in each system.	On the Type Definition panel, during implementation, or with DEFINE TYPE SCL
Output	Endeavor uses output libraries to store executable forms of elements produced by processors. Allocate these libraries by stage.	Within processors, during implementation

Library	Comments	Where/When Defined
Source Output	Endeavor uses source output libraries to store copybooks, assembler macros, or JCL procedures that are copied from elsewhere and therefore have to be available in full source form. Source output libraries are stage-specific. You can define a source output library for each type in a stage, or share one library across types.	On the Type Definition panel or with DEFINE TYPE SCL Note: Source output libraries are optional if you store elements in reverse delta (PDS) format and do not compress nor encrypt.
Processor Load and Listings	Endeavor uses <i>processor load libraries</i> to store the executable form of Endeavor processors. Allocate one processor load library for each stage of your production environment. Point to the production processor load library from all stages. Processor listing libraries are optional. Endeavor uses them to store listings when processors are compiled.	On the System Definition panel, during implementation, or with DEFINE SYSTEM SCL
Endeavor Listing	Endeavor uses listing libraries for listings that are created by compilers, link-editors, and other utilities within processors, and then stored by the CONLIST utility. A single library can be shared across systems.	Within processors, during implementation
Include	Endeavor uses Include libraries to store the full form of Panvalet (++)INCLUDE) and Librarian (-INC) INCLUDE statements.	On the Type Definition panel, during implementation, or with DEFINE TYPE SCL
Processor Output	These are libraries referred to in processors, to which processors write their output. Processor output libraries can be source libraries, executable libraries, or listing libraries.	Within processors, during implementation

1.5.1 Where to Allocate Your Libraries

The table below summarizes where each of these libraries should be allocated in the sample software life cycle.

Library Name	C1DEFLT	DEV TEST	DEV QA	PROD EMERG	PROD PROD
Package data set	x				
Master Control Files	x				
CCID validation data set	x				
Base and delta libraries, by type		x	x	x	x
Output libraries, by type		x	x	x	x
Source output libraries, by type		x	x	x	x
Processor load libraries		x	x	x	x
Processor output libraries (source, executable, list)		x	x	x	x
Include libraries, by type		x	x	x	x

1.6 Working with Elements

Endevor inventory is manipulated by executing Endevor commands called *actions*. Some actions are available in both foreground and in batch, while others are available only in batch. Batch actions are also available when building packages.

- The *User Guide* explains how to execute actions in foreground and submit batch action requests.
- The *SCL Reference Guide* contains the syntax for the Software Control Language (SCL) for Endevor. SCL allows you to code Endevor batch action requests.

1.6.1 Endevor Actions and Availability

The table below summarizes Endevor actions and their availability.

This Action	Is Available in Foreground	Is Available in Batch	Description of Action
Add	x	x	Puts a member from an external data set under Endevor control.
Archive		x	Writes the current version of an element to a sequential data set.
Copy		x	Copies an element from an archive data set to a data set external to Endevor.
Delete	x	x	Erases base and delta forms of an element and removes related information from a Master Control File.
Display (function)	x		Displays information about an element.
Generate	x	x	Creates executables and source outputs, if defined.
List		x	Creates a list of elements or PDS members that meet specific selection criteria. One effective use of this function is to perform impact analysis.
Move	x	x	Moves elements between stages, within or across environments.
Print	x	x	Prints element or member information.
Restore		x	Restores elements to Endevor from an archive data set.
Retrieve	x	x	Copies elements from Endevor to an external data set.
Signin	x	x	Removes or changes the user signout associated with an element.
Transfer		x	Moves elements between two Endevor locations or between an Endevor and an archive location.

This Action	Is Available in Foreground	Is Available in Batch	Description of Action
Update	x	x	Updates an element from an external data set.

1.6.2 Actions by Job Function

A typical site might include the following job functions:

- Development
- QA or Test
- Turnover
- Audit
- Management
- Endeavor administration

The table below summarizes, for each job function, the actions that someone might perform:

Action	Dev	QA/Test	Turnover	Audit	Mgmt	Admin
Add/ Update	x	x				x
Archive						x
Copy						x
Delete						x
Display	x	x	x	x	x	x
Generate	x					x
List	x					x
Move	x	x	x			x
Print	x	x	x	x	x	x
Restore						x
Retrieve	x	x				x
Signin	x	x			x	x
Transfer			x			x

1.6.3 Creating Executable Forms of Elements

Endevor uses OS JCL streams called *processors* to create executable forms of source code, including source modules, object modules, load modules, and listings. *Processor groups* name the Generate, Move, and Delete processors to be associated with an element, along with any symbols in the processors that can be overridden. For example, one processor group might contain processors to compile and link-edit source written in VS COBOL, and another group might contain processors for COBOL II source.

There are three kinds of processors:

- *Generate processors* execute automatically when an element is added or updated in Stage 1, or generated in either stage. Optionally, Generate processors execute when an element is moved, restored, or transferred to Endevor from an archive data set.

Typically, the Generate processor creates an executable form of the element, together with any associated outputs (such as listings).

- *Delete processors* execute when an element is deleted, transferred, moved, or archived. Generally, the Delete processor deletes any output that was created by the corresponding Generate processor.
- *Move processors* execute when Endevor moves elements from one stage in the life cycle to another. Move processors generally copy all the output previously created for the element, or recreate those outputs in the target stage. Optionally, a Move processor can be executed when an element is transferred.

See the *Extended Processors Guide* for complete information about processors.

1.6.4 Correlating Source with Executables

Endevor can place an encrypted audit stamp, called a *footprint*, in the output source, object, or load modules that are created by processors. The footprint provides an integrity check between the source form of an element and its executable form.

1.6.5 Packages

Endevor packages allow you to formalize your use of actions. Using packages, you can do the following:

- Create sets of actions (against elements) that can be tracked, maintained, and reused as a unit.
- Establish approval procedures for packages.
- Ship packages to remote locations.

See the *Packages Guide* for complete information about packages.

1.7 Security

To provide a comprehensive security program for your Endeavor system, you must address security issues in two essential areas: data set security and functional security.

1.7.1 Data Set Security

Endeavor does not provide data set security. Data set security is performed by a site security package, such as RACF, eTrust CA-ACF2 Security for OS/390 and z/OS, or eTrust CA-Top Secret Security for OS/390 and z/OS.

It is recommended that you implement data set security on the Endeavor data sets (for example, Master Control Files, package files, base and delta libraries, source output libraries, and so on). This type of security prevents users from inadvertently altering the Endeavor configuration. For additional information, see the discussion of alternate ID support in the *Installation Guide*.

1.7.2 Functional Security

Functional security involves protecting Endeavor inventory functions from unauthorized use. These functions include access to menu options, the ability to perform certain actions against certain inventory areas, and other secured Endeavor options.

Functional security is provided by Endeavor. You must choose one of the following:

- *Endeavor Native Security Tables*, which control environment access, primary and foreground menu options, and action authorization. These tables require an LLA refresh.
- *AllFusion Endeavor Change Manager Interface for External Security*, (formerly *External Security Interface - ESI*) which controls environment access, primary and foreground menu options, and action authorization through the interface and security rules under your site security package. These tables are dynamic. In addition, the Interface for External Security allows you to customize your functional security capabilities.

See the *Security Guide* for information on implementing security for Endeavor.

Chapter 2. Implementation and the Organization

2.1 Overview

This chapter addresses the organizational and technical aspects of implementing Endeavor, which you must consider and understand before proceeding.

2.2 Organizational Considerations

When implementing Endeavor, it is important that you understand both the technical and organizational considerations before proceeding.

Organizations can expect higher productivity, less production downtime, and better overall software quality shortly after the implementation of Endeavor. However, any change to organizational procedures requires careful project planning, education, awareness, management support, and internal selling. This chapter addresses these issues to help you understand, and therefore better manage, these aspects of implementing Endeavor.

2.2.1 Accessing Receptiveness to Change

To assess how Endeavor will be received in your organization, consider the following points:

- How large is your organization? Would you consider your organization small (50 programmers or less), medium (50-200 programmers), or large (more than 200 programmers)?
- How concise are the standards for software management now? Are they documented? Are they enforced? If the standards are not clear, what will it take for the organization to agree upon what the standards will be?
- Will Endeavor be used extensively across groups within the organization? For example, will development, QA, and production turnover all use it?
- Were all groups that will use Endeavor involved in the evaluation and buying decision? If any of those groups were not involved in the original assessment, you may need to show them the benefits of Endeavor.
- Are the organization's reasons for purchasing Endeavor clearly understood by all groups impacted by the purchase? Different levels of objectives include control, standardization, and auditability.

2.2.2 Developing a Strategy

When all considerations are addressed, you need to develop a strategy for proceeding. Make decisions about the following issues:

- How much of the implementation effort do you want to spend on organizational activities, such as education and internal selling? How much effort on technical activities, such as writing processors?

For an average size company (100 programmers or so), the ratio of time spent should be about 50/50. For larger companies, the percentage of time spent on organizational issues must be higher.

- How extensively should you document current procedures, and how much time is needed to define new procedures?

- How do you get different groups to agree on a standardized set of procedures?
- How much internal selling is needed to obtain management support for Endeavor in the organization?
- What is the scope of the implementation? For example, should you implement source management first, and output and configuration management at a later time? Or, do you want a phased approach such as JCL for the entire company, then copybooks, then COBOL, and so on?

2.3 Selecting an Implementation Team

It is strongly recommended that at least one full-time person is devoted to coordinating the Endeavor implementation effort. You can add more resources as needed.

2.3.1 Required Technical Knowledge

People with the following knowledge sets should be on the team, or readily accessible:

- IBM OS/390 JCL
- Application development software (such as compilers, linkage editors, and so on)
- Utilities, report writers, and so on
- Operation and use of ISPF/PDF facilities
- Your security system (usually RACF, eTrust CA-Top Secret Security for OS/390 and z/OS, or eTrust CA-ACF2 Security for OS/390 and z/OS)
- Audit requirements
- Testing and production standards and turnover procedures
- Applications to be implemented
- Current procedures (standard and emergency changes, approval procedures, and so forth)

2.3.2 Required Organizational Involvement

Ownership of Endeavor varies by organization. However, representatives from the following areas will probably be required for the implementation:

- **Technical Services**—System software specialists are needed to install and verify the installation as described in the *Installation Guide*. Operations analysts are needed to address JCL, production scheduling, and other general production turnover questions.
- **Production Control**—Representatives are needed to assist in automating the movement of new or changed software into production. This ensures that new procedures satisfy the business needs of the organization.
- **Applications**—At least one analyst from each application must be available during that application's implementation, to assist in application classification, compile procedures, training, and so on.
- **Security Administration**—An authorized person must define the appropriate security rules.
- **Auditing**—A person from the auditing department must ensure that audit requirements are defined and that the new procedures satisfy these requirements.
- **Training and documentation**—These groups need to be involved in developing training and internal documentation to support the system.

2.3.3 Public Relations Requirements

Some very large organizations have a dedicated person handling "public relations" work related to Endeavor. How you choose to handle this aspect of the implementation—the need to gain acceptance from many parts of the organization—consider involving people with the following skills on the implementation team:

- **Experience with the environment**—Having an experienced person who understands the environment and is easy to communicate with is highly beneficial. This person should be able to relate to the production control and operations perspectives as well as the development and QA perspectives.
- **Organizational skills**—Often, a representative from the implementation team is asked to mediate discussions between different groups trying to derive standards for software management.

2.4 Laying the Groundwork

When your organization made the decision to acquire Endeavor, Computer Associates representatives most likely spent time with the people performing the evaluation, explaining how Endeavor could suit the needs of the organization. Whatever the scope of your implementation, the implementation team will probably have to do some internal selling as well. Management, production turnover staff, developers from different groups, auditors, the QA staff, and other groups will need to learn about and accept Endeavor before offering their support for the implementation.

2.4.1 Prepare a Presentation

Plan on preparing a presentation to introduce Endeavor to the people in your organization who are not familiar with Endeavor. Depending on the complexity of your organization, you may need to create several presentations. Your presentations should include items such as those listed below:

- A summary of the problems faced under the current software management procedures, and the justification for the acquisition of Endeavor. Whenever possible, use objective numbers and cost justifications, such as number of system outages per month due to software problems.
- An overview of Endeavor functionality.
- An explanation of how Endeavor will help the organization meet its goals.
- A brief explanation of the proposed Endeavor implementation, including a walkthrough of the software life cycle. As the presentation proceeds, explain the benefits and relate the solution to the problems identified above.
- Time frames for the implementation, and what is expected of the group you are addressing.

2.4.2 Tailor the Presentation

Tailor your presentation to your audience. For a presentation to higher-level management, for example, you may want to de-emphasize the life cycle and technical issues surrounding the implementation and focus on the resources required, the payback expected, and the time frames for implementation. For programmers, focus on how Endeavor can increase their productivity by allowing them to view change history for their programs online, or show them how to use the footprint display panels to view source directly from load modules.

Emphasize goals in all presentations and public relations efforts. Work into each presentation the goals of productivity, availability of the production software environment, and any additional objectives for which Endeavor was acquired.

2.5 Make the Presentation

Once the groundwork is in place for the implementation, it is time to introduce Endeavor to the various groups within the organization that will use the product and benefit from it.

2.5.1 Introducing Endeavor to an Organization

When making a presentation to the following groups, consider stressing the points indicated:

- **Management**—Management must accept the proposed implementation and give it the appropriate priority over other tasks. Inevitably, a manager has limited resources, so you must convince the manager that Endeavor is the best way to increase software quality and personnel productivity.

Show management that Endeavor improves productivity, improves software quality, and standardizes the software management process. You may want to discuss approvals and reporting, areas in which management may be involved with Endeavor.

- **Development**—It is important to show development personnel how Endeavor can improve the process of solving problems, viewing changes, preventing regression, and improving production availability (resulting in fewer late night phone calls!). Show them that Endeavor can make their jobs easier by reducing software development and maintenance time.

- **Production turnover**—Show how Endeavor will automate the production turnover process, move source and executables together, facilitate backing out and addressing quick fixes, and provide a better audit trail of who made changes. Endeavor also enforces standards for things like JCL (by automating the invocation of JCL-checkers like JOB-SCAN) without the production staff's involvement. By the time a change gets to the production staff, they can be sure it has already passed standards.

- **Audit**—The audit group can have a great deal of influence over software management procedures, particularly in regulated industries. In addition to the other benefits of Endeavor, audit groups should be shown the change tracking, SMF recording, and footprinting features in Endeavor.

Emphasize that Endeavor can provide auditors with preventive controls rather than detective controls; that is, prevent the exception before it occurs rather than improve the process through which exceptions are detected.

2.5.2 After the Presentation

After the presentation, keep in touch with the attendees to gain feedback from the presentation, to alert them when their particular group will be affected by the implementation process, and to address ongoing questions or concerns from them or anyone in their group.

Presentations are not the only way to introduce Endeavor to an organization. Additional implementation aids include the following:

- **Computer Associates sales materials**—Computer Associates representatives are more than happy to provide sales materials that you can distribute to coworkers.
- **Demonstrations**—You may want to use the supplied MVSTEST and MVSPROD environments to show people how Endeavor works. Use a simple demo (RETRIEVE, ADD, MOVE) to show functionality. Tailor your demo to the audience (for example, auditors would probably like to see footprinting, while programmers would like to see online change history tracking).
- **Personal contact**—Consider setting up a special phone number that people can call for answers to their questions about Endeavor. Have someone readily available to dispense reliable product and implementation information.

2.6 Using a Pilot Application

One of the best ways to implement Endeavor is to start with a pilot application. No matter how you decide to approach the implementation (for source, output, or configuration management), a successful pilot makes the rest of the implementation infinitely easier.

The implementation of a pilot system allows you to:

- Validate your software life cycle design and procedures.
- Build momentum and excitement about Endeavor within the organization.

2.6.1 Characteristics of the Pilot Application

A pilot application ideally has the following characteristics:

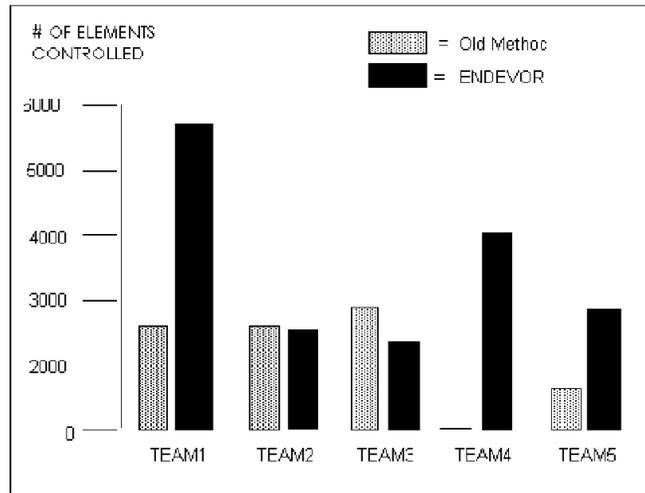
- One system, two to four subsystems, and a good sampling of element types to be managed.
- Approximately 200 - 400 or more elements, with the maximum amount of 1000. Remember that an element is any component that belongs to the application—whether it is a copybook, program, JCL, or other type.
- Relatively little change activity.
- Available personnel. Because this is the first application to be implemented, more time than usual may be required of the people involved with the pilot group.

You may want to rate several applications according to the above criteria to determine the best application to use as the pilot.

2.7 Building Momentum

Building momentum in the organization is the best way to keep the implementation process on track. In addition to implementing a successful pilot application, you can also build momentum by doing the following:

- **Talking Endeavor**—For example, when you hear of a programmer overlaying a colleague's code, do not hesitate to mention how the situation could have been avoided with Endeavor.
- **Using the pilot project as a reference**—When people ask how Endeavor works, refer them to the pilot application staff to share their experiences.
- **Charting the progress of each group in the implementation effort**—The figure below shows how you can accomplish this simply yet effectively, perhaps in the context of an implementation newsletter.



Many sites that have tried this approach found that it improved the motivation of teams that were on or ahead of schedule, as well as those teams that were behind schedule.

2.8 Summary

Each organization's software management requirements are unique. An organization must therefore develop an implementation strategy for Endeavor that conforms to these requirements. An awareness of the organizational considerations pertaining to software management at your site can help you ensure a successful Endeavor implementation.

Chapter 3. Implementing Endeavor

3.1 Overview

This chapter briefly describes the steps to use when implementing Endeavor.

3.1.1 Endeavor Implementations

Endeavor is a fully functional software management tool. As mentioned in Chapter 1, “Basic Concepts” on page 1-1, you can implement Endeavor to provide many levels of functionality.

You can implement Endeavor for the following:

- Source management only
- Output management as well as source management
- Configuration management as well as source and output management

3.1.2 Assumptions

This chapter was written with the following assumptions:

- Endeavor has been installed and the installation verified.
- The MVSTEST and MVSPROD environments provided on the installation tape are available.
- You are familiar with basic Endeavor concepts. These concepts are explained in Chapter 1, Basic Concepts.

Where appropriate, this chapter refers you to the Endeavor documentation set. To make the best use of this chapter, this documentation set should be available.

3.1.3 The Software Life Cycle

The life cycle shown in this chapter, or a slight variation of it, is used by a majority of Endeavor implementations. It is the basis for the supplied MVSTEST and MVSPROD environments, and will be used later when explaining data set naming conventions and other implementation considerations.

3.2 Implementation Alternatives

There is no one set way to implement Endeavor. Some sites implement Endeavor in phases. In the first phase, a site might implement all applications for source management. In the next phase, the site implements each application for output management.

Other sites implement each application for both source and output management.

You must decide which type of implementation best suits the requirements of your site.

Tip: As you proceed with the implementation, document the procedures you develop and the standards you use. This is important for the training phase that follows implementation.

3.2.1 A Phased Implementation

With a phased implementation, you have the following advantages:

- Inventory is established
- Signin and signout can be enabled
- Base (forward delta) or images (reverse delta) and change levels are available
- Benefits of source management are realized quickly
- Security is activated for all application source
- All personnel are trained at the same time on product functionality

3.2.2 A Full Implementation

A full implementation shares the advantages of a phased implementation, and also provides the following advantages:

- Online approvals are allowed
- Procedures are standardized
- The Automated Configuration Manager (ACM) is enabled
- Source can be related to executables

3.3 Implementing for Source Management

There are several reasons why you might decide to implement Endeavor only for source management:

- You do not want to use Endeavor to manage executables.
- You plan a phased-in implementation of Endeavor.
- You are looking for a quick replacement for your existing library management system.

3.3.1 What's Involved?

There are six steps involved in implementing source management:

Step	What You Do
1	Define your software life cycle
2	Analyze your inventory
3	Define the logical structure to Endeavor
4	Define the physical structure to Endeavor
5	Define CCIDs (optional)
6	Enable package processing

3.4 Step 1: Define Your Software Life Cycle

Software life cycles are site-specific. To determine the best life cycle for your site, you need to spend as much time as necessary to accomplish the following:

- Understand and document current procedures
- Develop procedures for normal and emergency processing
- Understand current library configuration and ownership
- Reach consensus on software management objectives

Review the discussion of the software life cycle in Chapter 1, Basic Concepts, to refresh your understanding of the life cycle and its components.

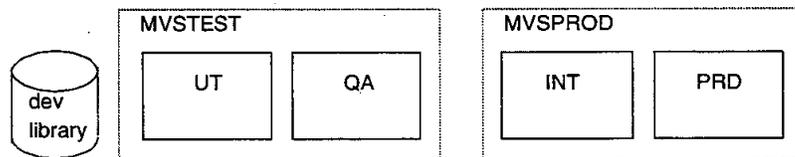
3.4.1 Decide Stages for Endeavor Control

Decide which stages of the life cycle you want Endeavor to control. You can put some, or all, of the stages in your life cycle under control of Endeavor.

3.4.2 Define Environments

Environment is the Endeavor term for functional areas in your organization. The illustration below shows how environments are set up in the Endeavor sample application (which represents a fairly typical life cycle).

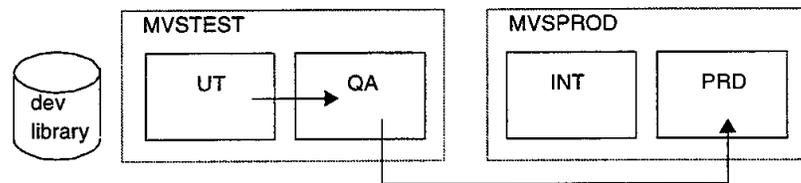
The testing (UT) and quality assurance (QA) stages in the life cycle are part of the development function. Production applications (PRD) and problem fixes (INT) are part of the production function. The administrator defines environment MVSTEST to include Stage UT and Stage QA, and environment MVSPROD to include Stage INT and Stage PRD. Development activities take place in a development library, outside of Endeavor.



3.4.3 Define the Map

Applications in each life cycle follow a unique route through the environment and stage locations you have defined. You can set up as many routes as you need to accommodate different life cycles at your site. These routes make up the map for your site. Endeavor uses these routes to automate the processes of adding, displaying, retrieving, moving, and generating inventory in a particular life cycle.

The Endeavor administrator might decide to establish a route for inventory at this site that promotes the inventory from Stage UT to Stage QA to Stage PRD. This map is illustrated below:



CAUTION:

You must enter the first environment in the map through Stage 1. Stage 1 always maps to Stage 2 of the environment. You can, however, map Stage 2 of one environment to Stage 1 or Stage 2 of the next environment.

A map route must always exit an environment from Stage 2. The route can enter either Stage 1 or Stage 2 of the next environment.

3.4.4 "Normal" Development Procedure

Normal development proceeds as follows:

Step	Action
1	Retrieve inventory from Stage PRD to the development library
2	Work on the inventory and add it into Stage UT
3	Promote the inventory along the map to Stage QA and eventually back to Stage PRD

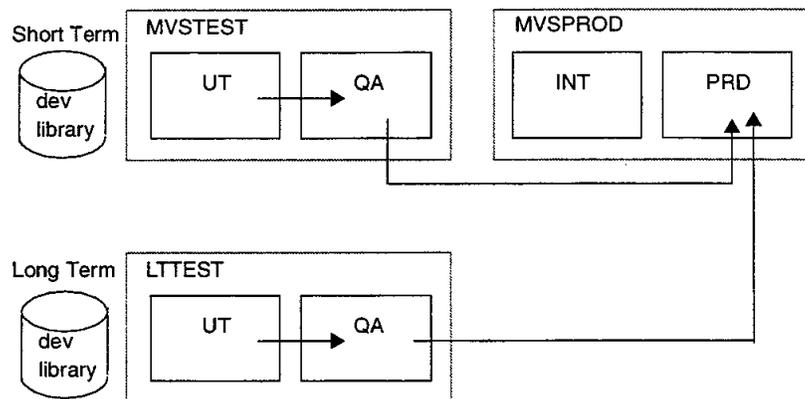
3.4.5 Quick Fix or Emergency Procedure

Quick fixes or emergencies are handled as described below:

Step	Action
1	Retrieve inventory from Stage PRD to a maintenance library
2	Work on the inventory and add it into Stage INT
3	Promote the inventory to Stage PRD or have the INT types compile directly into override emergency libraries

3.4.6 Mapping for Parallel Development

For a parallel development situation, many users find it advantageous to create an additional route, for long-term development use. The map in these situations has two routes and can be depicted as shown:



To define a map, see the *Administration Guide*.

3.4.7 Define Master Control Files

You will need one Master Control File (MCF) per stage for your software life cycle. For example, the MVSTEST and MVSPROD environments each have two stages, and therefore a total of four Master Control Files. These are VSAM data sets.

- Suggested naming conventions:

NODE1	User-defined (for example, Endeavor)
NODE2	Stage ID

NODE3	MCF
NODE4	INDEX or DATA (this node is supplied by the system)

An example of a VSAM cluster name for a Master Control File is:

ENDEVOR.TEST.MCF

- **Occurrence**—One per stage
- **Size**—Approximately five cylinders per 1000 elements to be maintained
- **Format**—VSAM
- **Where defined**—In the Endeavor Defaults Table

3.4.8 Environment Implementation Checklist

When defining your environments and map, be sure you do the following:

1. Set up routes that reflect your software life cycle. Draw a diagram of the routes first, using the previous examples as models.
2. Establish the routes in the Endeavor Defaults Table. The environments in the routes must appear in the same order (top-down) as they were established in the map. For example, you set up the following map route:

TEST->QA->PROD

These environments must appear in the Endeavor Defaults Table in the following order:

TEST

QA

PROD

3. Edit the Endeavor Defaults Table. Make sure that you edit the table to include your Master Control File data set definitions.

You must also ensure that the appropriate site options are correctly specified. For example, if you plan to implement Endeavor for configuration management, ensure that the Endeavor Automated Configuration Manager (ACM) is enabled in the Endeavor Defaults Table.

CAUTION:

Do not activate the Interface for External Security until the required preparatory steps are completed.

3.5 Step 2: Analyze the Inventory

Review Chapter 1 if you need to refresh your memory about the Endeavor inventory structure. During this step, you classify your present inventory according to this structure.

First you must decide, in broad terms, how you want to classify your inventory. You can classify inventory by:

- Business function (Applications, Tools, Systems)
- Application group (Finance, Human Resources)
- Another method

One of the most common ways of structuring your inventory is to classify your inventory by application group (for example, Finance), and by application within this group (for example, Accounts Payable). This structure maps to the system-subsystem structure of Endeavor, and is the format used in the sample application.

You also need to categorize the application components with which you are working; for example, JCL, COBOL source, or copybooks. This categorization maps to types in Endeavor.

You can use the Endeavor Inventory Analyzer to identify the different kinds of components that exist in your current inventory and to classify them according to Endeavor types and processor groups. The reports produced by the Analyzer are a useful way to identify the types and processor groups that you need to define to Endeavor.

You can also use the Inventory Analyzer, along with your data set or member naming conventions, to classify your inventory into the Endeavor logical structure (system and subsystem) that you decide to use.

See the *Inventory Analyzer Guide* for more information and instructions.

CAUTION:

If you are using security, the inventory structure you implement may impact the security rules that you have to define. Enabling security is one of the final implementation steps. See 3.16, “Step 10: Enable Security” on page 3-28 for more information.

3.6 Step 3: Define the Logical Structure

Based on the inventory grouping that you have designed in Steps 1 and 2, you can now define your logical structure to Endeavor.

Tip: Before actually defining the logical and physical structures, write down the proposed naming standards for the logical structure and libraries, and review them to make sure the structures are consistent and meet your needs.

3.6.1 Implement the Logical Structure

There are two ways you can implement your logical structure: online and in batch. The advantage of online processing is that it is realtime, and options pop up immediately. The advantage of batch processing is that you create a template that you can use repeatedly to define your inventory structure.

The most beneficial (and easiest) method of implementing your inventory structure involves a combination of both online and batch processing. This procedure is described below:

Step	Action
1	Build an environment by defining a system, subsystem(s), type(s) and processor group(s) for a subset of your inventory. Do so online by selecting the ENVIRONMENT option from the Primary Options Menu (define a small subset of model definitions; for example, define 1 system, 1 subsystem, 1 type, and 1 processor group).
2	Once the environment is built, use the batch environment BUILD SCL action to create DEFINE SCL for the environment.
3	Modify the SCL created above to create the definitions for the remainder of your inventory structure.

If you decide not to use the combination of methods, it is recommended that you implement your logical structure using batch processing.

See the *Administration Guide* for information about online implementation. See the *SCL Reference Guide* for information about batch implementation.

3.6.2 Define Systems

When defining systems to Endeavor, you must do the following:

- Define a system to each environment in which you plan to use the system.
- Define at least one subsystem to each system.

You can map systems across environments.

3.6.3 Define Subsystems

You must define a particular subsystem to each system in which you plan to use the subsystem.

You can map subsystems across environments.

3.6.4 Define Types

If you used the Endeavor Inventory Analyzer, it has indicated most or all of the types you should define. Remember the following:

- You must define types to each system and stage combination in which you plan to use them. All subsystems defined to a system can use the types defined to that system.
- You can map types across the environment/stage locations in your map.

Tip: Mapping system, subsystem, or type names allows these names to change automatically as source moves up the map. See the *Administration Guide* for more information about mapping system, subsystem, and type names.

3.6.5 Type Naming Conventions

Consider using generic type names, such as COBOL. You can then create as many processor groups as you need to handle the variations within each type. For example, for type COBOL, you could create a processor group for each type of COBOL that must be processed. See the *Extended Processors Guide* for more information on processor groups.

Suggested naming standards for types are presented below. This list is not complete and is provided as a guideline only.

ASM	LINKCARD	SORTCNTL	FORTRAN
BASIC	MACRO	SPECS	RPG
CLIST	MARKV	TABLES	JCL
COBOL	NETRON	TELON	REPORTS
COPYBOOK	PLI	TRANSFORM	SCL

EASYTREV	PROC	UTILITY
----------	------	---------

The Inventory Analyzer uses these naming conventions for types.

3.6.6 Verifying the Definitions

After you have defined your logical structure to Endeavor, run CONRPT07, System Definition Profile, to verify that you have set up the correct definitions. You should also run CONRPT07 whenever you change a map route, to verify that you have made the modifications correctly.

See the *Reports Guide* for a sample of CONRPT07.

3.7 Step 4: Define the Physical Structure

This step involves creating a physical data set structure to support the logical structure you have just defined.

For any Endeavor implementation, you must define and allocate Master Control Files and base and delta libraries. You defined and allocated Master Control Files in Step 1. In Step 4, you define and allocate base and delta libraries.

If you plan to use Endeavor packages, you must also define and allocate a package data set. See 3.9, “Step 6: Enable Package Processing” on page 3-18 for related information.

3.7.1 Naming Conventions

Use the following suggested naming conventions when defining your base and delta libraries. These libraries typically are partitioned data sets or of another standard partitioned organization. See the discussion of format in 3.7.7, “Where Libraries Are Defined” on page 3-16.

NODE1	System
NODE2	Stage ID
NODE3	Base
	Delta

An example of a base library name is:

```
FINANCE.TEST.BASE
```

3.7.2 Using Symbolics to Define Endeavor Libraries

Endeavor allows you to define base, delta, source output, and Include libraries using symbolics. The library name in the example above could be written as:

```
&C1SYSTEM..&C1STAGE..BASE
```

When you define this data set using symbolics, the definition is reusable for multiple type definitions. The symbolics you can use are listed below:

Organize Libraries By	Using this Symbolic	Or this Alias
Site	&C1SITE	(no alias)
Environment	&C1ENVMNT	&C1EN
Stage	&C1STAGE	&C1ST
Stage 1	&C1STAGE1	&C1ST1

Organize Libraries By	Using this Symbolic	Or this Alias
Stage 2	&C1STAGE2	&C1ST2
Stage ID	&C1STGID	&C1SI
Stage 1 ID	&C1STGID1	&C1SI1
Stage 2 ID	&C1STGID2	&C1SI2
Stage number	&C1STGNUM	&C1S#
System	&C1SYSTEM	&C1SY
Subsystem	&C1SUBSYS	&C1SU
Type	&C1ELTYPE	&C1TY

Use these symbolics when specifying base and delta libraries on type definition panels. Endeavor replaces the symbolic with the proper information from the action request. Note the following example:

```
&C1SYSTEM..&C1SUBSYS..&C1STGID..SRCLIB
```

This library name refers to a library called SRCLIB for each subsystem and stage combination within a given system.

3.7.3 Occurrence

It is recommended that you allocate at least one base/delta library pair per system per stage for all components that are not needed in PDS format outside of Endeavor (for example, compiled source). For components that are needed in PDS format outside of Endeavor, see the information in 3.7.7, “Where Libraries Are Defined” on page 3-16.

Base and delta libraries can be shared across stages within the same environment.

3.7.4 Sizing

To estimate the size of these data sets, use the Disk Space Requirements Worksheet provided in the *Installation Guide*.

For directory sizing and data set maintenance information, see the *Utilities Guide*.

3.7.5 Format

Base and delta libraries can be in PDS, PDS/E, Endeavor LIB, Panvalet, or Librarian format. In addition, the base library can also be an HFS directory, however, the delta library cannot. For information about Endeavor LIB data sets, see the *Utilities Guide*.

3.7.6 Element Storage Formats

You can store elements in either *reverse delta* or *forward delta* format.

Reverse Delta Format—In *reverse delta* format, the element base is the current image of the element. Endeavor recreates previous levels of the element by applying delta levels to this base. All previous changes to the element are maintained in the delta library.

Computer Associates recommends this format for Endeavor base product users. It is also recommended that you use reverse delta format for types when the current level of the element might be used by another process. For example, a COPYBOOK or a COBOL program might be used as part of a compile process; or JCL may be submitted for execution.

If you select this format, you can store the element base as is (unencrypted), or encrypted and compressed:

Store the Element Base	For these Advantages
Unencrypted	<p>Source is directly readable by programs outside of Endeavor, such as compilers.</p> <p>Base/delta format is the same as regular PDS format, eliminating the need for source output libraries.</p> <p>List and query functions perform more efficiently.</p>
In encrypted and compressed format	<p>You receive DASD savings of 10-40%.</p> <p>A single base library can contain multiple elements with the same name, but different system, subsystem, or type specifications.</p>

Be aware of the following if you select the reverse delta storage format:

- Endeavor does not allow two unencrypted elements with the same name to be stored in the same base library. Keep this in mind when planning your inventory structure.
- A source output library is required if elements will be backed out.

Forward Delta—In *forward delta* format, the element base is the source form of the element when it is first added into Endeavor. When Endeavor processes elements stored in this format, it applies all delta levels to the base to create the current image of the element.

If you select this format, Endeavor encrypts and compresses both the base and subsequent deltas, with the same advantages of encryption and compression that occur for reverse delta format.

3.7.7 Where Libraries Are Defined

Endevor libraries are defined on the Type Definition panel or using the DEFINE TYPE SCL.

3.7.8 More About Base and Delta Libraries

The following information applies to base and delta libraries:

- If you plan to use the Automated Configuration Manager (ACM), increase size estimates by 20% and double the number of directory blocks.

If you decide to use reverse delta format, increase the size estimate of the delta library by 30% and triple the number of directory blocks. The increased size is necessary because both the base and delta members for the component lists are stored in the element's delta library.

- The LRECL for base and delta libraries must be set to 255.

If you decide to use reverse delta format, only the delta library LRECL must be set to 255.

- Before allocating the base and delta libraries, you must make a decision regarding the format in which you want to store element source and change history.

3.7.9 Backup and Recovery

Planning backup and recovery procedures is a very important part of implementation.

Put backup and recovery procedures in place in accordance with the instructions in the *Utilities Guide*.

3.8 Step 5: Define CCIDs (Optional)

Endevor change control identifiers (CCIDs) most often correspond to mechanisms such as work order requests or request-for-service numbers. If your shop uses this type of mechanism, consider implementing CCIDs.

If you decide to implement CCIDs, you must make the following decisions:

- Do you want to simply make the tool available, without validation or administration?
- Do you want to predefine CCIDs so they can be validated? Predefining CCIDs allows you to do the following:
 - Validate CCIDs entered by users against the predefined CCIDs.
 - Associate user IDs or specific inventory areas with a CCID.

3.8.1 CCID Definition Data Set

If you decide to validate CCIDs, you must allocate a CCID Definition data set.

- Suggested naming conventions:

NODE1	project
NODE2	data set
NODE3	type

An example of a CCID Definition data set is:

ENDEVOR.CCIDVAL.SAMCIPO

- **Occurrence**—One per site
- **Size**—This data set is usually quite small—if you allocate one primary track with one secondary extent, you should have enough space
- **Format**—Sequential
- **Where defined**—In the Endevor Defaults Table

For more information on CCIDs and CCID validation, see the *Administration Guide*.

3.9 Step 6: Enable Package Processing

Packages are the online approval mechanism within Endeavor. If you want to replace or upgrade your existing approval system (for example, a paper system), packages are an excellent way to do so.

Endeavor packages allow you to do the following:

- Create sets of actions that can be tracked, maintained, and reused as a unit.
- Establish approval procedures for packages.
- Ship package outputs to remote locations.

See the *Packages Guide* for information on packages and approver groups, as well as shipping package outputs.

3.9.1 Set Up a Naming System

Decide on a naming convention for packages. For example, you may want to increment package numbers sequentially. Or, you might want to include codes that indicate which group or application is using a particular package, or which CCIDs are involved with a package.

3.9.2 Define and Allocate a Package Data Set

If you decide to use a package data set, you must define and allocate it as follows:

- Suggested naming conventions:

NODE1	uprpx
NODE2	uqual
NODE3	PACKAGE

An example of a package data set name is:

uprpx.uqual.PACKAGE

- **Occurrence**—One per site
- **Size**—For package data set size information, see the *Installation Guide*
- **Format**—VSAM
- **Where defined**—In the Endeavor Defaults Table

3.9.3 Enable Component Validation

The component validation feature is invoked during package CAST, and ensures the following:

- An element cannot be moved without its corresponding dependencies.
- An element cannot be moved if it has not been assembled, compiled, or linked with the current version of a dependent element.

For instructions on enabling component validation, see the *Packages Guide*.

3.9.4 Approver Groups

Decide who will approve packages in specific inventory areas. Define approver groups for these approvers, then relate the groups to the necessary inventory areas.

See the *Packages Guide* for more information about approver groups and the procedures involved in performing the above tasks.

3.9.5 What to Do Next

When you have finished Steps 1-6, you can do one of the following:

- Continue with Step 7 in the next section if you are implementing Endeavor for output or configuration management.
- Go to 3.15, “Final Implementation Steps” on page 3-27 if you are implementing Endeavor for source management only.

3.10 Implementing for Output Management

To implement Endeavor for output management, you must first complete Steps 1-6, as presented in the "Implementation for Source Management" section of this chapter.

Two additional steps are involved in implementing Endeavor for output management:

Step	What You Do
7	Define and allocate output libraries
8	Define processors

3.11 Step 7: Define and Allocate Output Libraries

In addition to the libraries allocated in Step 4, implementing Endeavor for output management requires the allocation of additional libraries to store the outputs of Endeavor processors. The information below summarizes which user-defined libraries should be allocated to implement Endeavor for output management.

- Executable libraries, by type
- Source output libraries, by type
- Processor load libraries
- Include libraries, by type (if applicable)
- Listing libraries (if you are storing listings online)

3.11.1 Output Library Information

When you define output libraries, use the following conventions:

- Suggested naming conventions:

NODE1	System
NODE2	Stage
NODE3	COPYLIB LISTINGS LOADLIB JCLLIB MACLIB OBJLIB PROCLIB SRCLIB and so on

An example of an output library name is:

FINANCE.TEST.LOADLIB

- **Occurrence**—Usually one per system/stage/type
- **Size**—See the following for size information:
 - To estimate the size of these data sets, see Appendix B, Disk Space Requirements, in the *Installation Guide*.
 - For directory sizing and data set maintenance information, see the *Utilities Guide*.
 - For information about Endeavor LIB data sets, see the *Utilities Guide*.
- **Format**—PDS, PDS/E, Endeavor LIB, Panvalet, Librarian. Output library format is usually dictated by executable requirements. Source output and INCLUDE libraries can also be HFS directories.

- **Where defined**—In the type definition and in processors

3.11.2 More About Output Libraries

Endevor can write outputs to any output library. For example, if your site has centralized load libraries or JCL libraries, Endeavor can write to them. The naming conventions presented here are only a suggestion.

3.11.3 Output Library Allocation

Consider setting up a worksheet for output library allocation. Use a format similar to the sample worksheet shown below:

Library Name	Format	DCB information	Size	Comments
FINANCE.TEST.- LOADLIB	PDS	RECFM=U, LRECL=80,...	1 cyl	
FINANCE.QA.- JCLLIB	PDS		1 cyl	
FINANCE.EMER.- SRCLIB	PDS		10 cyl	
FINANCE.PROD.- MACLIB	PDS		1 cyl	
FINANCE.TEST.- JCLLIB	PDS		1 cyl	

For examples of output library allocations, see the type definitions in the sample application.

3.12 Step 8: Define Processors

Processors are JCL streams that create executables from Endeavor elements. To use Endeavor for output management, you must write and maintain processors to create the desired executables.

3.12.1 Writing Processors

See the *Extended Processors Guide* for an explanation and illustration of processor creation, testing, and management, as well as the various types and uses of processors.

If you plan to implement Endeavor for configuration management, be sure to include the MONITOR=COMPONENTS statement in your processors. See the *Automated Configuration Option Guide* for instructions on adding MONITOR=COMPONENTS to your processors.

CAUTION:

You should test your processors thoroughly, using representative examples of each element type of the application to be implemented.

3.12.2 Processor Groups and Types

Using the list of types and processor groups produced by the Inventory Analyzer, define the processor groups and associate them with their respective types.

See the *Extended Processors Guide* for information about processor groups, including naming conventions.

3.12.3 Managing Processors

See the *Extended Processors Guide* for information about managing processors.

3.12.4 Determining Type Processing Sequence

A type processing sequence is used when multiple element types are processed within a single batch request.

See the *Administration Guide* for information on type sequence definition.

3.12.5 Sample Processors

The sample implementation includes the following processors:

Processor Name	Description
GCOBNBL	A COBOL Compile and Link-edit processor.

Processor Name	Description
GCIINBL	A COBOL II Compile and Link-edit processor.
GCOBDBL	A processor that performs a DB2 Precompile, Compiles and Link-edits a COBOL program, binds the DB2 plan, and footprints the plan table.
GCIIDBL	A processor that performs a DB2 Precompile, Compiles and Link-edits a COBOL II program, binds the DB2 plan, and footprints the plan table.
GCOBNBO	A Compile-only processor for COBOL.
GCOBNBL1	A COBOL Compile and Link-edit processor with linkage editor input.
GCOBNCL	A CICS Precompile, COBOL Compile, and Link-edit processor.
GLECNNL	A Link-edit-only processor (composite link).
GASMNBL	An Assembler Compile and Link-edit processor.

3.12.6 What to Do Next

When you have finished Steps 1-8, you can do one of the following:

- Continue with Step 9 in the next section if you are implementing Endeavor for configuration management.
- Go to 3.15, “Final Implementation Steps” on page 3-27 if you are implementing Endeavor for source and output management only.

3.13 Implementing for Configuration Management

To implement Endeavor for configuration management, you must first complete Steps 1-8, as presented in the "Implementation for Source Management" and "Implementation for Output Management" sections of this chapter. There is one more step required for implementing Endeavor for configuration management:

Step	What You Do
9	Enable the Automated Configuration Manager (ACM) capability

3.14 Step 9: Enable ACM

If you did not enable the Endeavor Automated Configuration Manager (ACM) at the end of Step 1, do so now by editing the Endeavor Defaults Table.

CAUTION:

The final step in implementing for configuration management is to create the ACM baseline. This must be done after you have loaded your inventory into Endeavor. See 3.17, “Step 11: Load the Inventory” on page 3-29 for instructions.

3.15 Final Implementation Steps

Here are the required final four steps for all levels. No matter which level of implementation you have chosen, you must perform these four final steps in the implementation process:

Step	What You Do
10	Enable security for your Endeavor system
11	Load inventory into Endeavor
12	Train (or set up training for) the Endeavor users in the organization
13	Go into production with Endeavor

3.16 Step 10: Enable Security

See the *Security Guide* for instructions on implementing security for your Endeavor system. It is recommended that this be done now rather than earlier, so as not to interfere with the implementation work you have been doing.

Test the security implementation across all functions and user levels, before full implementation, to make sure it works properly.

3.17 Step 11: Load the Inventory

Before loading the entire inventory, verify your implementation using a small subset of the applications to be controlled.

3.17.1 Overview

For applications that are actively changing, quiesce the change activity as close to implementation as possible. You may want to rerun the Inventory Analyzer before implementing Endeavor, to account for any new components.

When you have completed Steps 1-10, load the inventory that you want Endeavor to manage. Run parallel tests for a week or so to make sure Endeavor is functioning correctly, then reload the inventory.

Using the Load Utility is the easiest way to load the inventory into Endeavor. See the *Utilities Guide* for information on the load utility.

3.17.2 Enable the ACM Baseline

If you are implementing Endeavor for configuration management, you should create a baseline component list for those inventory elements that have components.

Endeavor ACM produces component lists when a Generate processor is executed against an element. To make sure that this process does not replace existing load modules, do the following:

- Take the Generate processor in production and temporarily change the SYSLMOD or other output library to DUMMY.
- Issue a Generate action against all programs or elements that have components.
- Change SYSLMOD back to its original value.

CAUTION:

If this is a source-management-only implementation, you may have to change your Compile procedures to accommodate storing source under Endeavor. For example, if you have Panvalet and have used PAN#1 for retrieval prior to compiling, you may have to change the RETRIEVE action in Endeavor.

3.18 Step 12: Provide Internal Training

All people who will be using Endeavor should be trained. Typically, end-user training for use of basic Endeavor functionality should take about two to four hours.

Training can be delivered by Computer Associates personnel, through the Computer Associates CBT (self-training) courses, or through internal training programs.

You should also consider using the procedures you have documented during the implementation process as the basis for a user's guide for your site.

3.19 Step 13: Go into Production

As mentioned in Step 11, after initially loading your inventory into Endeavor, run Endeavor in parallel with your production system for a week or so to make sure everything is functioning as designed. Then go into production with Endeavor.

You should hold a post-implementation review two to four weeks after production cutover. Use the feedback from the review to fine-tune the Endeavor implementation to better meet the needs of the site.

Appendix A. Upgrading to Endeavor 4.0

A.1 New Features

The following describes the new features introduced for Endeavor 4.0.

- Endeavor Release 4.0 is downward compatible with Endeavor Release 3.9 only if no Endeavor Release 4.0-only features are implemented. See the Endeavor 4.0 Release Summary for details on all the new features.
- Endeavor control tables (such as C1DEFLT, ENDICNFG, ENCOPTBL, and so on) are now validated at Endeavor start up to ensure that all tables are compatible with Release 4.0.
- A special DDNAME, EN\$TROPT, causes Endeavor Release 4.0 to format and print all option table settings at Endeavor start up (such as C1DEFLT, ENDICNFG, ENCOPTBL, and so on).
- The Endeavor Release 4.0 Element Registration feature can be activated in WARN, CAUTION, or ERROR mode (and switched from one mode to the other at any time). If the processor group output type registration option is activated, no conflicts will occur because the default value for the processor group output type (TYPE NAME+PROC GROUP NAME) is unique.
- A new utility, BC1PXCAT, is provided to create the Endeavor Release 4.0 catalog from existing MCF control files. This is not a file conversion utility—it is a build utility.
- Component validation (occurs at Package CAST) logic was greatly improved—especially in the area of diagnostic messages.

A.2 Changes for Endeavor 4.0

The following describes the changes you should know about for Endeavor Release 4.0.

- No type name changes across the map are allowed ("NEXT TYPE" name can no longer differ from the current type name).
- If long element names are used, the LRECL of the delta library must be at least 259.
- ACMQ is now required if ACM is active. If ACMQ was not implemented under Release 3.9 (or not implemented at all), the ACMQ load procedure must be executed to populate the ACMQ repository. The C1DEFLT5 ACMIDXUP parameter was removed. The Endeavor alternate ID can now be used to secure the ACMQ VSAM linear data sets.
- All Environments defined in an Environment Map **MUST** be referenced by one Element Catalog and **ONLY** that Element Catalog.

A.3 Recommendations for Endeavor 4.0

The following section describes recommendations for Endeavor Release 4.0.

- RLS or CA-LServ implementation is highly recommended for the Endeavor Release 4.0 Catalog data set, MCFs, and the PCF (Package Control File).
- Due to the key structure of the catalog, it is highly recommended that element searches are done with some kind of element and type specification (the catalog key is element name—type name).

Glossary

Access Security Table. Endeavor table that defines the environment(s) to which each user has access. There is one Access Security Table for each site.

ACM. See *Automated Configuration Manager (ACM)*.

actions. Commands used to maintain or otherwise act against elements. See *Add, Archive, Copy, Delete, Display, Generate, List, Move, Print, Restore, Retrieve, Signin, Transfer, and Update*.

Add. An Endeavor action used to place members from an external data set under control of Endeavor.

alternate ID. Facility that protects the Endeavor-controlled data sets (such as Master Control Files, package data sets, and base and delta libraries) from access by an individual user, while still allowing the Endeavor system to have access.

Administrator ID that has authority to the data sets. When Endeavor has to access the data sets, it takes on the security authorization assigned to that ID.

Information is provided for the alternate ID through three parameters in the Endeavor Defaults Table: RACFGRP, RACFPWD, and RACFUID. See the *Installation Guide* for more information.

analysis utility. See *Inventory Analyzer*.

approval. An electronic *signoff* mechanism for packages. Approval may be required for a package before it can be executed.

approved. Status of a package after required approvers have reviewed and *signed off* on a package by approving it.

approver. A person authorized to *signoff* on a package prior to execution. Signing off on a package means reviewing the information contained in a package and approving or denying it.

approver group. A collection of one or more approvers. Approver groups are defined within each environment and can be associated with particular inventory areas.

approver group relationship. The relationship established between an approver group and one or more inventory areas, authorizing members of that approver group to review (then approve or deny) packages related to those inventory areas.

approver type. Specifies the kinds of packages that an approver group can review. When the approver type is STANDARD, that approver group can only review standard packages related to its authorized inventory areas. When the approver type is EMERGENCY, that approver group can only review emergency packages related to its authorized inventory areas.

Archive. Action used to write an element and all related Endeavor information to a sequential data set. The DCB must specify variable blocked records (RECFM=VB), a minimum LRECL of 1021, DSORG=PS, and a blocksize equal to your LRECL + 4 (minimum 1025). When archiving to tape, the recommended blocksize is 32,000.

authorization. The ability to perform certain privileged functions within the IBM OS/390 environment.

Automated Configuration Manager (ACM). Optional facility that allows you to monitor selected libraries and data sets and maintain a *component list* for each element in the monitored areas. The component list provides an audit trail of program-component information at the time of each compile. Also called AllFusion Endeavor Change Manager Automated Configuration Option.

automatic consolidation. See *consolidation*.

backin. To restore the executable members of a package to the state they were in before the package was backed out. Reverses the backout process.

backout. To return the executable members of a package to the state they were in prior to package execution.

base level. When storing elements with forward deltas, the lowest level of an element within a particular stage. This level represents the source for the element in that stage. If an element exists in both stages, there is a base level in each stage.

When storing elements with reverse deltas, the base level is the current level. See also *image*.

base library. Partitioned data set (PDS), an AllFusion CA-Panvalet for OS/390 and z/OS, an AllFusion CA-Librarian for OS/390 and z/OS, or Endeavor LIB file that stores the base members for elements defined to Endeavor. A base library is defined for each element type, but can be shared across types. See also *image library*.

base member. Member in a base library. Each base member corresponds to an element, and contains the source for the base level of that element. Base member names are generated internally by Endeavor, and do not correspond to the element name unless reverse delta with non-encryption is used.

base regression. The amount (percent) by which the statements stored in a new level of an element change the statements stored in the base level. See also *regression percent*.

batch. An IBM term referring to an environment in which non-interactive programs are executed. In Endeavor, *batch* refers to the execution of actions and reports in a non-interactive region (vs. execution in foreground).

BDT. See *Bulk Data Transfer*.

browse. To view the contents of a data set, without being able to change its contents.

Bulk Data Transfer (BDT). IBM transmission utility supported by the Endeavor package shipment utility. Abbreviated as follows when establishing destinations:

- BDT2—for BDT Version 2
- BDTN—for BDT vis NJE/NJI

C1DEFLTS Table. See *Endeavor Defaults Table*.

cast a package. To freeze the actions included in a package. A package cannot be edited after it is cast, and only approvers can work with it.

CCID. See *Change Control ID (CCID)*.

CCID definition data set. A data set that identifies the CCIDs to be used within Endeavor. The definition file must be a card-image data set (80-byte, fixed-format records).

CCID validation. Checking a CCID specified on an action against the CCIDs defined in the CCID definition data set.

Change Control ID (CCID). A logical grouping mechanism by which user-specified portions of the Endeavor inventory can be tagged, then viewed, tracked, and manipulated. The use of CCIDs is optional, but may be required on a system-by-system basis. The same is true of comments.

change regression. The amount (percent) by which the statements stored in a new level change the revisions made by the previous level. See also *regression percent*.

checksum. An internally calculated value within a package. Endeavor uses the checksum to determine if a package has been changed.

command field. Field appearing in the upper-left corner (second line) of those Endeavor screens on which you can specify a TSO command. You can enter any appropriate TSO command in the command field.

comment. A 1- to 40-character user-defined remark associated with an action or package, generally describing the reason for the action or purpose of the package. Used in conjunction with CCIDs. See also *Change Control ID (CCID)*.

committed. Status of a package after it has been committed.

commit a package. To record all events related to a given package, and remove all backin and backout information. After a package is committed it can no longer be backed out or backed in.

complementary data sets. Data sets that can be shipped along with package shipments. The complementary data sets for a given shipment contain a backout of that shipment.

component. The output produced or input read in by a generate or move processor. This term applies primarily to the Endeavor ACM product.

The components of a generated element include the following:

- The *input components* that were included to produce an output of the generate processor; for example, copybooks would be considered an input component to a COBOL compiler.
- The *element* itself.
- All *outputs* created by the generate or move processor, for example, an object deck for a COBOL compiler.
- The *processor* that generated or moved the element.
- User-defined related data.

Components are referenced by element names (including related Endeavor location information) or member names.

component list. A list of all components created or read by a generate or move processor. The component list can be viewed using the Print action or through the Display Element/Component panel. This panel is available only with the Endeavor ACM product.

The component list provides an audit trail of program-component information at the time of each compile.

component monitoring. Feature of ACM that allows you to check selected data sets for component relationships.

component validation. When casting a package, Endeavor validates that all dependent components are present in the package and that those components have not changed since they were last used.

configuration management. The capture and storage of program-component relationships and the tracking of these relationships over time.

CONNECT:Direct. Formerly known as Network DataMover. CONNECT:Direct is a network transmission utility provided by Sterling Commerce and supported by the Endeavor package shipment utility. Abbreviated NWD when establishing destinations.

consolidation. Endeavor facility that allows you to specify a number of delta levels to retain when a member reaches the consolidation level specified for its type. If you do not specify the number of levels to retain, Endeavor consolidates all levels.

Copy. An Endeavor action used to copy an element from an archive data set to a data set external to Endeavor.

copyback. To search for an element along the map, beginning at a designated stage; find the element; then copy it back to the initial stage. Copyback is available as an explicit option with the Generate action. Endeavor also uses copyback when adding, transferring, and moving elements.

create a package. To build the SCL for a package, then associate this SCL with other package-related information such as a package ID, an execution window, etc.

CSECT. Control section. An IBM term for that part of a program that is a relocatable unit and for which all components are loaded into adjoining main storage locations.

current level. The most recent source for an element. When using forward deltas, the current level of an element comprises the base level plus all subsequent change levels. When using reverse deltas, the current level of an element is the current source.

data set mapping rule. See *DSN mapping rule*.

data set validation. The optional capability of verifying that retrieved elements are added or updated from the data set to which they were last retrieved. This ensures that the same copy of the element (revised as appropriate) is placed back in Endeavor.

Data set validation can be specified separately by system. If this facility is in effect for a system, you can override it to add the element back from a different data set, provided you have proper authority to do so.

default. A default value is the value that Endeavor assumes to be in a field or statement if the user does not provide an alternative value. On foreground panels, fields usually display default values.

Defaults Table. See *Endeavor Defaults Table*.

Delete. An Endeavor action used to erase base and delta forms of an element and remove element information from a Master Control File or a component list.

delete processor. Processor that is run when an element is deleted from a stage. Typically, the delete processor deletes the output created by the corresponding generate processor. See also *processors*.

delta level. Record of a change to the base level of an element. Each change to an element creates a delta level. Endeavor compares the current level to the new source and

builds a delta level containing just the changes to the source.

delta library. Partitioned data set (PDS), an AllFusion CA-Panvalet for OS/390 and z/OS, an AllFusion CA-Librarian for OS/390 and z/OS, or an Endeavor LIB file that stores the delta members for elements defined to Endeavor. A delta library is defined for each element type.

delta member. Member in a delta library. Each delta member corresponds to an element, and contains all the levels for that element subsequent to the base level. Delta member names are generated internally by Endeavor and do not correspond to the element name, unless reverse deltas are being used.

denied. Status of a package when it has been reviewed, but denied, by an approver.

deny a package. An option for a package approver. If one approver denies a package, it cannot be executed.

destination. Package outputs are shipped to destinations. A destination record contains the information needed by Endeavor to ship package outputs to that destination.

Display. An Endeavor action used to view environment definitions, element information, and footprint-related data.

DSN mapping rule. A user-defined correspondence between host data set names and remote data set names. DSN mapping rules are used when shipping package outputs.

element. Partitioned data set (PDS) members, or an AllFusion CA-Panvalet for OS/390 and z/OS, AllFusion CA-Librarian for OS/390 and z/OS, or sequential data sets that have been placed under control of Endeavor. The default element name is the member name. Actions are performed against elements. Elements are identified by the environment and stage in which they are located, and by the system, subsystem, and type in which they are classified.

element change. A view of element information that shows the current level of an element, annotated to indicate the level at which each line was added to the source.

element component. In Endeavor ACM, the part of a component list referred to as *element information*. This information includes the footprint of the Endeavor source.

element history. A view of element information that shows all lines that have ever been present in a piece of source code, annotated to show the level at which the line was added and/or deleted from source.

element master. A view of Master Control File information about an element.

element name. The name assigned to an element, used to identify that element within Endeavor. It is recommended that any outputs created by output management be assigned the name of the corresponding element.

Element names must be unique within each system, subsystem, and type combination. An element name can include any of the following characters (and only these characters): A-Z, 0-9, @, #, and \$.

element summary of levels. A summary view of activity against an element at all levels. Information provided includes the number of statements at each level, the number of lines added and the number of lines deleted.

element type. See *type*.

emergency approval. The kind of approval given to emergency packages. An approver group must be given the authority to approve emergency packages.

emergency package. One of two types of packages. A package is identified as standard or as emergency when it is created. Emergency packages require approval from emergency approver groups.

enable backup. Option when creating a package. You can decide whether or not to allow the package to be backed out.

Endeavor classification. The system, subsystem, and type associated with an element.

Endeavor Defaults Table. Table of site-specific information necessary for Endeavor operation. The Endeavor Defaults Table includes environment and stage definitions, installed options, and site-specific hardware settings. There is one Endeavor Defaults Table for each site.

Endeavor LIB. High performance alternative to OS partitioned data sets under Endeavor. Endeavor LIB data sets do the following:

- Reorganize member space automatically as members are rewritten or deleted, thereby eliminating the need to compress the data set.
- Exploit 31-bit storage for VSAM-organized data sets, thereby reducing 24-bit storage contention.
- Expand directories and data sets automatically.
- Provide improved directory processing.
- Maintain additional statistical information about member size.

Endevor Link. Computer Associates product that enables communication between the Endevor Workstation product and Endevor. Also named AllFusion Endevor Change Manager Link Option.

Endevor listing libraries. Libraries used to store compressed compiler listings produced by processors.

Endevor location. Refers to the stage and environment where an element resides.

ENDEVOR return code (NDVR RC). Return code from action processing. Values are:

- 00—The action executes successfully.
- 04—A warning message is issued before a processor is invoked. This can occur, for example, when you specify override signout on an action, or add or update a member with no source changes.
- 08—The regression percentage exceeds the limit specified on a type definition, and the default severity of C is in effect.
- 12—The processor return code is greater than the MAXRC for any step in a processor, or there is an error in action processing before or after invoking a processor.
- 16—An abend has occurred.

Endevor symbolics. See *symbolics*.

environment. The top level of the logical structure used to classify elements in Endevor. Environments usually correspond to functional levels in an organization, for example development, quality assurance, and production. Each environment has two stages. There is no limit to the number of environments you can use.

environment name. The 1- to 8-character name assigned to each environment, used to identify that

environment within Endevor. The name can include any of the following characters: A-Z, 0-9, @, #, and \$.

environment title. The 1- to 40-character title assigned to each environment, used in various displays and reports to describe the environment.

ESI. See *External Security Interface (ESI)*.

event. See *package events*.

execute a package. To run a package. Packages that execute successfully can be backed out or committed.

Execution Report. Report output when you run Endevor actions. The Execution Report documents the actions requested and the processing that took place. The report can be viewed on-line by browsing data sets *userid.C1TEMPR1.MSGS* or *userid.C1TEMPR2.MSGS*. Endevor prints the report as member C1MSG1 on the batch SYSOUT.

execution window. A start date and time and an end date and time within which a package must be executed.

export a package. To copy package SCL into an external data set.

exit. The Endevor exit interface is designed for use with exits written in either assembler or in high-level languages such as COBOL.

External Security Interface (ESI). Optional interface used to implement *external security* at your site. If installed, this interface replaces the native security facility supplied on the installation tape (and implemented through the security tables), with calls to RACF, eTrust CA-ACF2 Security for OS/390 and z/OS, or eTrust CA-Top Secret Security for OS/390 and z/OS. Also named AllFusion Endevor Change Manager Interface for External Security.

fetch. See *copyback*.

footprint. Encrypted data added by processors to individual source, object, or load modules, to identify the Endevor element associated with that module. Endevor uses this data to display or otherwise process information related to the element.

A footprint includes (in encrypted format) the following information: site ID, environment name, stage number, system name, subsystem name, element name, element type, element version/level, and the date and time the footprint was assigned.

foreground. An IBM term referring to an environment in which interactive programs are executed. In Endeavor, you run actions in foreground by requesting those actions through the Endeavor Foreground Options Menu.

forward delta. A method for recording changes that stores a base version of code, then builds current versions by applying changes made to the base.

forward recovery. The process of taking an old level of an element and making it the current (new) level, thereby backing out any changes made by the levels between. To perform forward recovery, you first retrieve the older (to-be-recovered) level, then add or update the element using the retrieved source to create the new level.

Generate. An Endeavor action used to translate source into executables, then populate output libraries with these executables by executing the generate processor for an element.

generate processor. Generate processors translate source into executables, then populate output libraries with these executables.

group name. Within the definition of the Access and User Security Tables, a name associated with a particular security configuration that applies for multiple users. The name is then associated with any number of specific user IDs, to associate those IDs with the group-level security. This is a convenient way to assign security to several users having identical levels of access to the Endeavor environment.

identify record (IDR). An IBM term for a record in a load module that contains user-defined data. An IDR is created by the linkage editor when it encounters an IDENTIFY statement in the object deck. Within Endeavor, IDRs are used to store the footprint(s) associated with load modules.

image. The current level of an element, when that element is stored in reverse delta format, using non-encryption. See also *base level*.

image library. A library that contains elements stored in reverse delta format, using non-encryption. See also *base library*.

import a package. To create a package by copying SCL from an external data set.

INCLUDE library. AllFusion CA-Panvalet for OS/390 and z/OS library, an AllFusion CA-Librarian for OS/390

and z/OS library, or partitioned data set (PDS) that contains INCLUDE members referenced within Endeavor elements. This library is optional and can be defined for each element type.

The INCLUDE library is used by Retrieve actions if you specify that you want to expand INCLUDEs at the time the element is retrieved. It is also used by the CONWRITE utility, if you specify that you want to expand INCLUDEs during CONWRITE processing.

input component. When using Endeavor ACM, the components that were included to produce an output when executing the generate or move processor. A copybook, for example, is the input component when compiling a COBOL program.

inventory. The software components that make up your application software systems.

Inventory Analyzer. Computer Associates product that allows you to analyze your software inventory, classifying it according to Endeavor types. Used when implementing Endeavor.

inventory area. A subset of a software inventory, defined by its Endeavor location (environment and stage) and classification (system, subsystem, and type).

jump. To move an element from stage 2 in one environment to a stage in another environment on a map route, when a version of the element exists at an intermediate stage that is not part of the map route.

last action. Most recent action executed for an element. Once executed, each action is recorded as the last action except Archive, Delete, Display, List, and Print.

last action CCID. The CCID specified for the last action executed against an element.

level. The source for an element at a particular time. When an element is first added to a stage there is one level, known as the base level. Each time Endeavor actions change the source thereafter, a new—delta—level is created. See also *base level*, *delta level*, and *image*.

level number. Identifier for a specific level of an element. Endeavor assigns each set of changes a level number that is one higher than the number assigned to the preceding level.

library management. The classification, control, and storage of the physical components of a software inventory.

List. An Endeavor action used to list, in the form of action requests, elements from a Master Control File or archive data set, or members from a library. List can also be used for text scanning.

If the Automated Configuration Manager facility is installed, List can also search a component list based on specified criteria.

list panel. Panel used by Endeavor to display lists of systems, subsystems, types, elements, or members; also displays selection options for users. List panels are prepared and processed by the ISPF Table Display Facility.

load utility. Endeavor utility used to load members from an external data set into any stage in an environment.

location. See *Endeavor location*.

map. The promotion routes established for software inventory at a site. Environments and stages are mapped to each other in the Endeavor Defaults Table. Systems, subsystems, types, and processor groups are mapped to each other on their respective definition panels.

mapping rule. See *DSN mapping rule*.

Master Control File (MCF). Endeavor file that contains the definitions of stages, systems, subsystems, element types, and elements themselves. This file is accessed and updated by Endeavor, to manage the element definitions, to execute processors, and for other miscellaneous functions. There are two Master Control Files (MCFs) for each environment—one per stage.

MAXRC. A processor keyword that defines the highest acceptable return code for a processor step. If a step exceeds this return code, the Endeavor return code (NDVR RC) is set to 12. When this occurs the Element Master display shows *FAILED* in the NDVR RC field.

MCF. See *Master Control File (MCF)*.

model transmission control statements. Statements that control the functioning of data transmission programs used by the package shipment utility.

Move. An Endeavor action used to move elements between stages, within or across environments.

move processor. Move processors copy outputs, element information, and component lists from the source location to the target location of a Move or, optionally, a Transfer action.

name mask. Name masking enables you to use the wildcard (*) and placeholder (%) characters when performing actions. The wildcard character enables you to specify all names or all names beginning with a particular search string. The placeholder character defines a specific position within the search string. For example, the search string UPD% would return all four character names beginning with UPD.

native security. Security option supplied with the Endeavor installation tape. See also *External Security Interface (ESI)* and *security*.

NDVR RC. See *ENDEAVOR return code (NDVR RC)*.

notification facility. Endeavor facility that allows you to notify users of events that require a response from them.

output component. When using Endeavor ACM, the components created as a result of executing the generate or move processor. For example, an object deck is an output component when compiling a COBOL program.

output library. Any of several libraries used during output management, including the Endeavor processor listing library, processor load library, and source output library; as well as user copy libraries, load libraries, listing libraries, macro libraries, JCL libraries, databases, etc.

output management. That aspect of Endeavor which deals with the creation and maintenance of various outputs that relate to an element. The exact nature of these outputs varies depending on the corresponding element type, and is defined by the output management for that type. For example, output management might store a copy of the current source for the element in the source output library, or it might create a load module for the element or a listing associated with that load module.

package. A group of Endeavor actions that requires approval before it can be executed. Creating packages allows you to do the following:

- Group specific actions so they can be maintained and tracked as a single unit.
- Establish formal approval procedures to ensure data integrity through modifications.
- Centralize specific action groups so you can see them across environments and reuse them.

package data set. Data set where packages are stored. There is one package data set per environment.

package events. An audit trail recording the events that have occurred involving a package, logged by user ID, date, and time. Package events relate to the various steps of the package processing procedure, and include the following:

- Created
- Last Updated
- Cast
- Approved
- Executed
- Backed Out
- Backed In
- Committed

package exits. Exits that are called before and/or after package functions and subfunctions. See also *exit 7*.

package shipment. The transmission of package outputs, and optionally their backouts, from host sites to remote sites.

package status. Indicates the status of a package at any given time. Status levels for packages include the following:

- **In-edit**—A package is initially set to this status. It can be modified only when in this status. When you reset a package, its status is automatically set to *In-edit*.
- **In-approval**—When a package is cast, its status changes to *In-approval*, indicating that approvers can now review and approve or deny the package. No editing can be done once the package is cast.
- **Denied**—The status changes to *Denied* when an approver denies approval of the package during review.
- **Approved**—The status changes to *Approved* when all necessary approvers (required and optional) grant approval of the package during review, and when the quorum requirement is met for the package. A package is also considered Approved when it has been cast and no approvers have been identified.
- **In-execution**—The status changes to *In-execution* when package execution has begun.
- **Executed**—When the package has been executed successfully, its status changes to *Executed*.
- **Exec-failed**—When the package has aborted or failed execution, its status changes to *Exec-failed*.

- **Committed**—When a package has been committed, its status changes to *Committed*.

Endevor Parallel Development Manager (PDM).

Computer Associates product that automatically compares and integrates three versions of source code, allowing you to resolve conflicts resulting from concurrent development or from applying vendor updates to applications that have been customized in-house. Also named AllFusion Endevor Change Manager Parallel Development Option.

partitioned data set (PDS). A data set in direct access storage that is divided into partitions, called members. Each member can contain a program, part of a program, or data.

PDM. See *Endevor Parallel Development Manager (PDM)*.

PDS. See *partitioned data set (PDS)*.

Print. An Endevor action used to print element or member information.

PROC RC. See *processor return code (PROC RC)*.

PROC statement. A job control statement used in catalogued or in-stream procedures. PROC statements can be used to assign default values to symbolic parameters contained in a procedure. A PROC statement is also used to mark the beginning of in-stream procedures.

processors. Processors are standard OS JCL job streams that manipulate elements and their outputs: object modules, load modules, listings, and the like. There are three types of processors:

- *Generate processors* translate source to executables, then populate output libraries with these outputs.
- *Delete processors* delete outputs created by generate processors.
- *Move processors* copy or regenerate outputs, element information, and component lists from the source location to the target location of a MOVE or, optionally, a TRANSFER action.

Endevor supports both Endevor symbolics and user-defined symbolics in processors. This capability allows you to write one processor that you can use in multiple processor groups by changing the values assigned to one or more symbolics.

Endevor also provides a set of utilities for use when writing processors, and supports the use of in-stream data in processors.

See also *delete processor*, *generate processor*, and *move processor*.

processor component. When using Endevor ACM, the part of a component list that includes processor information. This information includes the footprint of the processor.

processor group. A processor group identifies a set of processors for a specific element type, as well as the default symbolic overrides for the processors' JCL. A group can include up to three processors—one generate, one delete, and one move processor, or any combination thereof. (For example, you could have a group that consisted only of a move processor and its symbolic overrides.)

Processor groups are useful when elements of one type require slightly different processing. For example, a site may have programs coded in batch COBOL and CICS COBOL. In this case, processor groups allows you to create a single COBOL type with two processor groups, one to handle each variation of COBOL code.

When you define a type to Endevor, you can also identify a default processor group for that type. Using symbolics when writing the processors for the default processor group can allow you to use the same processors, by changing symbolic definitions, for other processor groups associated with this type.

processor group symbolics. Symbolics defined in PROC statements in one or more processors in a processor group. These symbolics and their default values appear on the Processor Group Symbolics panel. By modifying these default values, you can use one processor in more than one processor group.

processor listing library. Optional library that stores the listings output from the Computer Associates-supplied processor named GPPROCSS.

processor load library. Endevor library that contains the load-module form of each processor. The modules from this library are executed when processors are invoked.

processor output library. Library referred to in a processor, to which that processor writes output. Processor output libraries can be source libraries, executable libraries, or listing libraries.

processor return code (PROC RC). Highest return code from the execution of a processor. Set to **FAILED** if the return code for any step in a processor exceeds the MAXRC for the processor. Set to **PROC'D?** if the element has not been generated after being restored or transferred from an archive data set or added/updated.

production data set. Data set used to store production code. This term is used in the package shipment utility to refer to host and remote production data sets.

program pathing. Security option under RACF, eTrust CA-ACF2 Security for OS/390 and z/OS, and eTrust CA-Top Secret Security for OS/390 and z/OS that allows you to restrict the data sets available to particular users, as well as the programs and load libraries from which those data sets can be accessed. This is *not* an Endevor option, but is specific to RACF, eTrust CA-ACF2 Security for OS/390 and z/OS, and eTrust CA-Top Secret Security for OS/390 and z/OS.

promote. To move an element from one inventory area to another inventory area.

promotion management. The task of coordinating and validating successive changes to the various inventory areas in a software development setting.

quorum (quorum size). The minimum number of approvers whose approval is required in order to execute a package. When a quorum size is indicated, at least that many approvers must review and approve a package.

regression. Term that refers to the condition where one set of changes to element source is overwritten by a subsequent set of changes. Endevor flags regression when the changes stored for a specific level of an element overwrite more than a predefined percentage of the element statements. See also *regression percent*.

regression percent. A percent of acceptable change to element source, defined for each element type. This is a percentage that, if exceeded, results in a user specifiable Endevor return code (0, 4, 8, or 12).

Each time a new level is created for an element, Endevor checks the changes stored in that level against this percent, both in terms of change to the statements stored for the base level (known as *base regression*) and change to the statements stored for the previous level (known as *change regression*). If the amount of change in either case exceeds the defined percent, Endevor issues a message of user-defined severity.

Reload Utility. The Reload utility allows you to recover an Endeavor VSAM control file (Master Control File, package data set) or a base/delta data set that was lost as a result of a physical device failure or site disaster. The RELOAD action restores data from data sets created by the unload process.

remote footprint synchronization. Procedure in which footprinted executables are shipped from a remote site to a host site, where footprint reports are run to compare the executables' footprints with host Master Control File information.

remote nodename. Part of a package shipment destination. Identifies the site to which package outputs are to be shipped. The name must be valid for the chosen data transmission program.

request data set. Data set that contains action requests to be submitted for batch processing. You create request data sets in foreground, using the SCL Generation facility.

request for data. Package exit capability, allowing package exit programs to make multiple, successive requests for Endeavor information on a single invocation of the exit.

reset a package. To erase all package event records, returning the package to In-edit status.

Resource Security Table. Endeavor table defining those element names that are restricted to a particular system(s) and subsystem(s), within a specific environment. The Resource Security Table is defined by the Endeavor administrator, using the CONSDEF macro. There can be at most one Resource Security Table for each Endeavor environment.

Restore. An Endeavor action used to restore an element to Endeavor from an archive data set.

Retrieve. An Endeavor action used to copy any level of an element to an external data set.

return code. See *ENDEVOR return code (NDVR RC)* or *processor return code (PROC RC)*.

reverse delta. A method for recording changes that stores the most recent version of the code, rebuilding prior versions by backing out individual changes from the current version.

review a package. To review the contents of a package. After reviewing a package an approver either approves or denies the package.

route. A series of environment and stage locations that make up the stages in a software life cycle. Taken together, all the routes at a site constitute the map for that site. See also *map*.

SCL. See *Software Control Language (SCL)*.

security. Endeavor feature that allows you to restrict action requests and access to elements. The security system supplied with Endeavor, known as *native security*, is implemented using three tables. See also *Access Security Table*, *Resource Security Table*, *External Security Interface (ESI)*, and *User Security Table*.

sharable. Characteristic of a package. A sharable package can be edited by people who did not create the package.

ship utility. Endeavor utility that allows you to ship package outputs to remote sites.

shipment confirmation. Confirmation occurs at two points in a package shipment:

- After execution of the data transmission utility.
- After execution of the remote copy/delete job step.

shipment staging. Creation and population of host staging data sets with package outputs or backout members.

Signin. An Endeavor action used to remove the current signout for an element. Signin can be implicit or explicit.

signout. The assignment of a user ID to an element, establishing *ownership* of that element. Signout is automatic when adding or updating elements in or when retrieving elements from Endeavor.

site. Location at which Endeavor is installed. The site is defined in the Endeavor Defaults Table, where it is assigned a site ID.

SMF records. Records written out if SMF recording is in effect, to document various Endeavor processing, as follows:

- An Action Record is written out at the end of (any) action processing.

-
- A Security Record is written out for each security violation (or each error returned from the security exit, 01).

SMF interface. Optional interface to IBM's System Management Facilities (SMF) that allows you to record historical information through SMF records (called Action Records or Security Records in Endeavor). This information is used to generate Historical Reports.

The implementation of the SMF interface is optional at each site. The recording of historical information is optional within each environment.

Software Control Language (SCL). Endeavor language used in batch to maintain or otherwise act against elements within Endeavor.

software distribution. The automated distribution and synchronization of software changes and the tracking of the implementation of those changes.

software life cycle. The stages through which software passes at a site during the development and maintenance process. A software life cycle might consist of development, testing, quality assurance, and production.

software management. The process of tracking changes to software components and their interrelationships over time. Includes configuration management, library management, software distribution, and version control.

source. The non-executable form of an element.

source library. Any of several libraries used during source management, including Endeavor base libraries, delta libraries, and INCLUDE libraries.

source management. The aspect of Endeavor that deals with the creation and maintenance of element source. Element source is maintained in base and delta libraries, in either an internal format or in standard IBM format (if reverse deltas and non-encryption are selected).

source output library. Endeavor library that contains the latest full source version of each element. This library is designed for use with copybooks, macros, procedures, etc., that are copied elsewhere. This library is optional but, if used, is specified in the definition of the corresponding element type.

stage. A stage in the software life cycle. There are two stages defined for each Endeavor environment.

stage ID. Identifier for the stage, used during processing to select (identify) the stage you want to process.

stage name. Name assigned to each stage during installation. A stage name can include any (and only) the following characters: A-Z, 0-9, @, #, and \$.

stage number. Relative number for the stage within the environment: 1 or 2.

stage title. The 1- to 20-character title assigned to each stage, used in displays and reports to describe the stage.

standard approval. One of two types of package approval. Standard packages can only be approved by standard approver groups.

standard package. One of two types of packages. A package is identified as standard or as emergency when it is created. Standard packages require approval from standard approver groups.

subsystem. Part of the Endeavor classification of an element. Subsystems are used for specific applications within a system. For example, there might be a purchase order subsystem and an accounts payable subsystem within the financial system.

symbolics. Endeavor supports two kinds of symbolics in processors:

- *Endeavor symbolics.* Any of several names, preceded by **&C1**, that are used within Endeavor processors to represent a value specific to an individual run of the processor. Values are assigned to Endeavor symbolics when the processor is executed.
- *User symbolics.* Defined by users in JCL PROC statements in processors. Allow one processor to be used in multiple processor groups. See also *processors* and *processor group*.

synchronize. When transferring or moving with history, if the current level of the target does not match any level of the source, a synchronize conflict is detected. Endeavor searches for the level of the target to match a level of the source; this level becomes the synchronization level. When there is a synchronize conflict, Endeavor does not allow the element to be transferred or moved unless the synchronization flag (SYNC option) is set to *Y* (SYNC=Y).

The SYNC option tells Endeavor to create a *sync level* at the target that reflects the differences. All levels after the sync level (the change history) associated with the FROM

location element are then appended to the TO location element and renumbered.

system. A means of classifying elements within Endeavor. A system typically represents the applications at a site. For example, there might be financial and manufacturing applications. A system must be defined to each environment where it will be used.

Transfer. An Endeavor action used to transport elements from a source location to a target location. Each location can be either an Endeavor location or an archive data set.

transmission method. Part of package shipment destinations. Identifies the transmission utility to be used to ship packages to the destination. See also *Tivoli NetView File Transfer Program (FTP)*, *Bulk Data Transfer (BDT)*, and *CONNECT:Direct*.

transportable footprints. Endeavor footprints that can be imbedded in DOS/VSE- and VM/CMS-bound object modules, using an OS/390 compiler, a DOS/VSE compiler, or a VM/CMS compiler. (See the *Administration Guide* for complete information.)

type. A category of source code used as part of the classification of an element in Endeavor. For example, there might be the following types: COBOL (for COBOL code); COPYBOOK (for copybooks); JCL (for JCL streams).

type processing sequence. Relative sequence of processing for the element types defined to each system. By defining a processing sequence, you could ensure, for example, that copybooks (type COPYBOOK) are updated before any COBOL programs (type COB) that might use those copybooks.

The Unload utility unloads and validates the contents of the VSAM Master Control File(s), base and delta files associated with the environments and systems specified on the job request. The file created by the Unload function contains a backup of all internal MCF definitions (system, subsystem, type, type sequence, data set, element master record) and base/delta data (element base, element delta, component base, component delta). Packages contained within a package data set can also be unloaded.

Unload utility. The Unload utility may be run for an entire environment or for selected systems within an

environment. Unload may also be directed to backup an entire package data set or individual packages.

Update. An Endeavor action used to add a member to Endeavor when an element with the same name is located in the target Stage 1.

user exit table. Table identifying exit programs to be called at each Endeavor exit point.

user ID. For actions run in foreground, the TSO user ID for the session. For actions run in batch, the job name or the ID specified through the USER= parameter on the job card, depending on how your Endeavor Defaults Table is set up.

user menu facility. The user menu facility allows the Endeavor administrator to attach user-defined functions to the Endeavor TSO/ISPF front end.

User Security Table. Endeavor table that defines the systems and subsystems to which each user has access, and for each system/subsystem, the type of processing (authorization level) allowed. There is one User Security Table for each environment.

user symbolics. See *symbolics*.

Validate. The Validate function allows you to ensure the integrity of one or more existing Endeavor environments and systems, and their related elements and components. These are the same checks performed as part of Unload processing, allowing this function to operate in a stand-alone mode.

version. Two-digit version identifier associated with an element. Two versions of an element are not allowed in the same environment.

version control. The maintenance, tracking, and auditing of modifications to an application over time, allowing prior development versions to be restored.

version number. Identifier for the version assigned to an element.

vv.ll. Identifier that refers to a particular version (vv) and level (ll) of element source.

Index

A

- Actions list of availability 1-16
- Activating security 1-19
- Alternatives for implementation 3-3
- Analyzing inventory 3-9
- Approval systems
 - replacing or upgrading existing system 3-18
- Automated Configuration Management (ACM) 3-16
- Automated Configuration Manager (ACM)
 - baseline creation 3-26

B

- Backup and recovery procedures 3-16
- Base libraries 3-14
- Baseline creation of Automated Configuration Manager (ACM) 3-26

C

- Change Control Identifiers (CCIDs)
 - allocating and identifying 3-17
- Configuration management 1-4, 3-25
- Conventions
 - naming of base/delta libraries 3-13
 - type naming 3-11

D

- Data sets
 - estimating size 3-14
- Defining
 - allocation of a package data set 3-18
 - environments 3-5
 - JCL streams 3-23
 - libraries using symbolics 3-13
 - logical structure 3-10
 - map 3-6
 - master control files 3-7

Defining (*continued*)

- physical structure 3-13
 - processors 3-23
 - software life cycle 3-5
 - stage 1-9
 - subsystem 1-9
 - subsystems 3-11
 - systems 1-9, 3-11
 - types 3-11
- Definition verification 3-12
 - Delta libraries 3-14

E

- Elements 3-15
- Emergency or quick fix procedures 3-7
- Enabling
 - package processing 3-18
 - security 3-27
- Endevor
 - definition, uses, and functionality 1-2
 - going into production 3-31
 - normal change procedures (figure) 1-6
 - organizational implementation 2-3
- Environment
 - checking implementation 3-8
 - definition 3-5
 - development and production 1-8
- Executing commands (tables) 1-16

F

- Footprints 1-18
- Forward delta format 3-15

I

- Implementation
 - alternatives 3-3
 - creating the ACM baseline 3-26

Implementation (*continued*)
environment checklist 3-8
for an inventory structure (table) 1-13
for source management 3-4
for source, output, or configuration management 1-4
momentum 2-11
process 3-27
sample processors 3-23
strategies 2-7
team selection 2-5
Integration of stage operations 1-7
Internal training 3-30
Inventory 3-29
analysis 3-9
loading 3-27
structure (table) 1-13

J

JCL streams
definition 3-23
Job function actions 1-17

L

Library
definition, allocation, and organization by
symbolics 1-13
Life cycle development by normal procedure 3-6
Logical structure
definition 3-10
LRECL settings for base and delta libraries 3-16

M

Managing the software life cycle 1-8
Map 3-6, 3-7
Master control files definition 3-7

N

Naming Conventions
for base/delta libraries 3-13
for output libraries 3-21

O

Online and batch processing 3-10
Organizational change
planning 2-3
Output libraries 3-21

Output management 1-4

P

Package data set
definition, allocation, and processing 3-18
usage 1-18
Parallel development mapping 3-7
Physical structure definition 3-13
Pilot application 2-10
Post-presentation feedback and education 2-9
Procedures for backup and recovery 3-16
Processors 1-18, 3-23

R

Reverse delta format 3-15

S

Security
data set and functional security 1-19
enabling 3-27
Software life cycle 1-5
definition 3-5
Source management 1-4
implementation 3-4
Stage 1-9
Stages for Endeavor control 3-5
Structure for classification of software 1-8
Subsystem
creation requirements 1-9
Symbolics
defining Endeavor libraries 3-13
Systems
profile (CONRPT07) 3-12

T

Training, internally 3-30
Type
naming conventions 3-11

U

Using sample processors for implementation 3-23

V

Verifying definitions 3-12