HP OpenVMS Migration Software for Alpha to Integrity Servers

Guide to Translating Images

February 2005

This manual describes how to use the Alpha Environment Software Translator (AEST) and other OpenVMS Migration Software for Alpha to Integrity Servers tools for translating and porting OpenVMS Alpha applications to OpenVMS Industry Standard 64 systems.

Revision/Update Information:	This is a new manual
Software Version:	OpenVMS Migration Software for Alpha to Integrity Servers Version 1.0
Operating System and Version:	OpenVMS I64 Version 8.2

Hewlett-Packard Company
Palo Alto, California

Contents

Preface vii

Part I User's Guide to Translating Images

1 Introduction to Image Translation

1.1 Overview of OMSAI 1–1
1.1.1 OMSAI Features 1–2
1.1.2 OMSAI Roles within a Migration Strategy 1–2
1.2 Image Translation Tools and Support 1–3
1.2.1 Alpha Environment Software Translator Utility 1–3
1.2.1.1 Alpha Image Information Files 1–3
1.2.1.2 How AEST Works 1–3
1.2.1.3 Code Analysis 1–3
1.2.1.4 Code Generation 1–3
1.2.2 Translated Image Environment 1–4

2 Translating Images

2.1 Before Translating an Image 2–1
2.2 Running AEST to Translate an Image 2–1
2.2.1 AEST Return Status 2–2
2.2.2 AEST Qualifiers 2–3
2.2.3 AEST Output Files 2–3
2.3 Translation of VESTed images 2–3

3 Running Translated Images

3.1 Running the Translated Image 3–1
3.2 Handling References to a Translated Image 3–1

Part II Developer's Guide to Translating Images

4 Analyzing Images

4.1 Using the /AUDIT Qualifier 4–1

5 Using Information Files

5.1 Alpha Image Information Files 5–1
5.2 AIIF File Syntax 5–1

6 Translating and Replacing OpenVMS Alpha Shareable Images

6.1 Interoperability Requirements 6–1
6.1.1 /TIE and /NONATIVE_ONLY Qualifiers 6–1
6.1.2 Preserving Upward Compatibility 6–2
6.2 Procedures for Building Shareable-Image Variants 6–2
6.2.1 Building the Original OpenVMS Alpha Shareable Image 6–3
6.2.2 Creating the Translated Shareable Image 6–3
6.2.2.1 Translated Main Image Calls Translated Shareable Image 6–4
6.2.2.2 Native Main Image Calls Translated Shareable Image 6–4
6.2.3 Building a Replacement Native Shareable Image 6–4

Part III Reference Information

A Command Summaries

B Error and Status Messages

B.1 Interpreting AEST Messages B–1
B.2 AEST Messages Descriptions B–1

C Translation and Performance Restrictions

C.1 Identifying Restrictions and Performance Issues C–1
C.2 Untranslatable Images C–1
C.3 Images Translatable with Warnings C–2
C.4 Images with Undetectable Translation Problems C–2

Examples

2–1 Translating an Image 2–2
4–1 Audit Information for SIEVE.EXE 4–1
A–1 Summary Format A–2

Figures

1–1 AEST Input, Processing, and Output 1–4
1–2 Run-Time Components 1–5

Tables

2–1	AEST Output Files	2–3
C–1	Untranslatable Images	C–1
C–2	Images Translatable with Warnings	C–2

Preface

Intended Audience

HP OpenVMS Migration Software for Alpha to Integrity Servers (OMSAI) facilitates migrating OpenVMS Alpha applications and OpenVMS VAX images translated to Alpha by OMSVA (further referred as VESTed images) to OpenVMS Industry Standard 64 (I64) systems by allowing you to translate OpenVMS Alpha images and VESTed images into equivalent I64 images. OMSAI consists of the Alpha Environment Software Translator (AEST) utility and a collection of programs and command files designed to ease the translation process. OpenVMS Migration Software for Alpha to Integrity Servers Translating Images documents the AEST utility and explains its use as part of a strategy for migrating OpenVMS Alpha applications to OpenVMS Industry Standard 64 systems.

This manual is for:

- Users who are translating all or part of an OpenVMS Alpha application as part of a strategy for migrating to an OpenVMS Industry Standard 64 system
- Users who are developing translated shareable images for OpenVMS Industry Standard 64 systems

Document Structure

This manual consists of three parts:

- Part I: User's Guide to Translating Images

  Information in Part I is applicable to all users:

  – Chapter 1 describes the image translation process and the supporting software components.

  – Chapter 2 describes how to use the utilities provided to translate OpenVMS Alpha images.

  – Chapter 3 describes how to run translated images on OpenVMS Industry Standard 64 systems.

- Part II: Developer's Guide to Translating Images

  Information in Part II is applicable to users who need to maximize translated image performance; users with access to source code that can be edited either to improve translation or to prepare source files for rebuilding on OpenVMS Alpha systems; and users preparing translated shareable images:

  – Chapter 4 describes how to use the analytical capabilities of AEST to enhance translation and to identify source problems that affect migration.

  – Chapter 5 describes image information files, which AEST creates and uses in the process of image translation.

  - Chapter 6 describes how to develop translated shareable images that interoperate with native shareable images on an OpenVMS I64 system.
- Part III: Reference Information

  Information in Part III is applicable to all users.
  - Appendix A provides a detailed description of the AEST command lines and qualifiers.
  - Appendix B provides an alphabetical listing of all AEST error messages with explanations and recommended user actions, if applicable.
  - Appendix C describes translation problems and suggests ways to debug them.

Conventions

The following conventions may be used in this manual:

Ctrl/*x*	A sequence such as Ctrl/*x* indicates that you must hold down the key labeled Ctrl while you press another key or a pointing device button.
PF1 *x*	A sequence such as PF1 *x* indicates that you must first press and release the key labeled PF1 and then press and release another key or a pointing device button.
$\boxed{\text{Return}}$	In examples, a key name enclosed in a box indicates that you press a key on the keyboard. (In text, a key name is not enclosed in a box.) In the HTML version of this document, this convention appears as brackets, rather than a box.
. . .	A horizontal ellipsis in examples indicates one of the following possibilities: • Additional optional arguments in a statement have been omitted. • The preceding item or items can be repeated one or more times. • Additional parameters, values, or other information can be entered.
. . .	A vertical ellipsis indicates the omission of items from a code example or command format; the items are omitted because they are not important to the topic being discussed.
( )	In command format descriptions, parentheses indicate that you must enclose choices in parentheses if you specify more than one.
[ ]	In command format descriptions, brackets indicate optional choices. You can choose one or more items or no items. Do not type the brackets on the command line. However, you must include the brackets in the syntax for OpenVMS directory specifications and for a substring specification in an assignment statement.
|	In command format descriptions, vertical bars separate choices within brackets or braces. Within brackets, the choices are optional; within braces, at least one choice is required. Do not type the vertical bars on the command line.

{ }	In command format descriptions, braces indicate required choices; you must choose at least one of the items listed. Do not type the braces on the command line.
bold type	Bold type represents the introduction of a new term. It also represents the name of an argument, an attribute, or a reason.
italic type	Italic type indicates important information, complete titles of manuals, or variables. Variables include information that varies in system output (Internal error *number*), in command lines (/PRODUCER=*name*), and in command parameters in text (where *dd* represents the predefined code for the device type).
UPPERCASE TYPE	Uppercase type indicates a command, the name of a routine, the name of a file, or the abbreviation for a system privilege.
`Example`	This typeface indicates code examples, command examples, and interactive screen displays. In text, this type also identifies URLs, UNIX commands and pathnames, PC-based commands and folders, and certain elements of the C programming language.
-	A hyphen at the end of a command format description, command line, or code line indicates that the command or statement continues on the following line.
numbers	All numbers in text are assumed to be decimal unless otherwise noted. Nondecimal radixes—binary, octal, or hexadecimal—are explicitly indicated.

Part I

User’s Guide to Translating Images

Part I contains the following information:

The image translation process and supporting software components	Chapter 1
Using the Alpha Environment Software Translator (AEST) utility to translate OpenVMS Alpha images	Chapter 2
Running translated images on an OpenVMS I64 system	Chapter 3

1 Introduction to Image Translation

This chapter discusses the following topics:

An overview of OMSAI	Section 1.1
A description of OMSAI tools and support	Section 1.2

1.1 Overview of OMSAI

HP OpenVMS Migration Software for Alpha to Integrity Servers (OMSAI) facilitates the migration of OpenVMS Alpha applications to OpenVMS Industry Standard 64 (I64) systems. An OMSAI utility, the Alpha Environment Software Translator (AEST), converts an OpenVMS Alpha executable or shareable image into a translated image that runs on an OpenVMS I64 system. When the translated image runs, the OpenVMS I64 system transparently supports the image with an environment that allows it to run as if it were on an OpenVMS Alpha system.

Translating and running an image can be as simple as the following example:

OpenVMS Alpha system

```
$ aest sieve  ❶
```

OpenVMS I64 system

```
$ run sieve_av  ❷
Sieve of Eratosthenes   ❸
500 iterations
1899 primes found
time taken : 2 seconds
```

The OMSAI kit includes the image SIEVE.EXE, which you can find in the directory SYS$SYSROOT:[SYSHLP.EXAMPLES.AEST] after the product has been installed. Try the commands shown on your own system. AEST creates the translated image in the current directory and names it by appending *_AV* to the input image file name. The translated version of SIEVE.EXE is named SIEVE_AV.EXE. In the previous example, the callouts highlight:

❶ The AEST command used to translate an image. The command assumes the file extension to be .EXE.
❷ The DCL command used to run the translated image. The same command is used to run SIEVE.EXE on an OpenVMS Alpha system.
❸ The output displayed by the translated image.

AEST can translate many images easily. However, when you are translating shareable images or images that are linked against user-written or third-party shareable images, you might need to take some additional steps. For example, an image might contain dependencies on the Alpha architecture or OpenVMS Alpha operating system that can affect translation. In some cases, you can use AEST qualifiers to accommodate such dependencies. In other cases, you may need to modify and rebuild source files, if they're available, to avoid the Alpha dependencies.

The remainder of this overview discusses OMSAI features (Section 1.1.1) and OMSAI's role within a migration strategy (Section 1.1.2).

1.1.1 OMSAI Features

OMSAI features include:

- Automated translation

  OMSAI translates images automatically; it requires no human intervention to analyze and translate code.

- Image analysis

  OMSAI analyzes images and reports its findings in various forms (messages and listings); these findings are useful not only for translation, but also for preparing original sources for recompiling and relinking on OpenVMS I64 systems.

  AEST cannot translate all OpenVMS Alpha images; some restrictions apply. For example, AEST cannot translate images linked on versions prior to Version 6.1. Also, AEST does not support certain coding practices because OpenVMS I64 systems cannot reproduce the corresponding Alpha code correctly. AEST issues an error message when it encounters unsupported code, and may or may not create a translated image, depending on the specific problem reported.

1.1.2 OMSAI Roles within a Migration Strategy

To migrate an OpenVMS Alpha application to an OpenVMS I64 system, the following options are available:

- Rebuild application source files (preferred)

  Rebuilding source files by recompiling and relinking them is the preferred option because it achieves better I64 system performance than image translation and results in smaller images. If the source files and an appropriate compiler are available, you can recompile and relink an application.

- Translate the application's OpenVMS Alpha images

  If either the application sources or the appropriate compiler is not available, then translating images is the only alternative.

- Combine source rebuilding with image translation

  Recompiling and relinking source files for some of the application and then translating images for the remainder of the application is a third migration option. This option is possible because the OpenVMS I64 system supports interoperability; that is, it allows native and translated images to issue calls to and receive calls from one another.

  The combination of application rebuilding and image translation provides a great deal of flexibility in your migration strategy.

1.2 Image Translation Tools and Support

The image translation tools and support include:

- The AEST utility, the primary translation tool (Section 1.2.1).
- The Translated Image Environment (TIE), a native shareable image and other components within the OpenVMS I64 system that supports translated images at run time (Section 1.2.2).

1.2.1 Alpha Environment Software Translator Utility

The AEST utility translates executable and shareable images; it accepts an OpenVMS Alpha image file (IMAGE.EXE) as input, analyzes the image file to locate Alpha code, and then creates a translated image file (IMAGE_AV.EXE). The translated image, which performs the identical functions as the original, is an OpenVMS I64 image that consists of both I64 code and the original OpenVMS Alpha image. This section first describes text files AEST uses for finding and analyzing code, and then briefly explains how AEST works.

1.2.1.1 Alpha Image Information Files

Alpha Image Information files (also called .AIIF files) are text files that provide AEST with additional information to be used during image translation. An Alpha Image Information file contains the information used by AEST utility to redirect external references (such as procedure calls) to replacements in (probably different) shared images. The .AIIF file describes the properties of a shareable image's exported interface, that is, the precise locations that are described in the image's symbol vector. These files contain mapping between Alpha and I64 entry points. Chapter 5 explains how AEST accesses the information files.

1.2.1.2 How AEST Works

AEST processes an OpenVMS Alpha image in two major phases to generate an equivalent OpenVMS I64 image: an analysis phase and a code generation phase. Figure 1–1 illustrates AEST input, processing, and output.

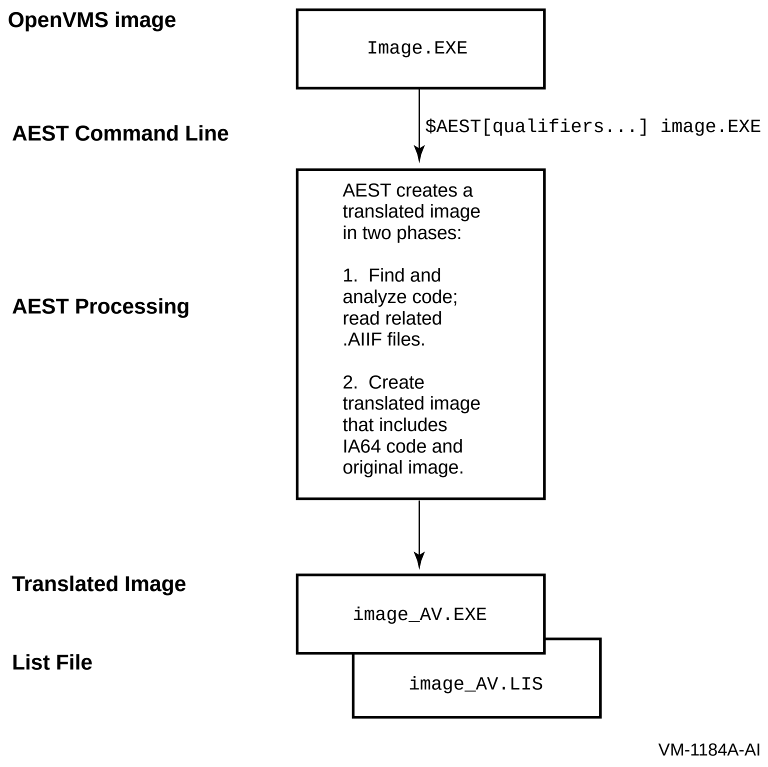
1.2.1.3 Code Analysis

During the analysis phase, AEST extensively analyzes the input image file to find the entry points, to separate the code and data, and to detect anomalies that cannot be correctly reproduced in the OpenVMS I64 environment. AEST tries to find as much code as possible since the TIE must interpret any unfound code at run time.

1.2.1.4 Code Generation

The second phase of translation generates the translated image, an I64 image that includes translated code as well as the complete original OpenVMS Alpha image. Translated code is native I64 code that performs the same function as the corresponding Alpha code in the original image. When the translated image runs on an OpenVMS I64 system, it reproduces the behavior of the original image.

Figure 1–1 AEST Input, Processing, and Output


OpenVMS image
Image.EXE
AEST Command Line
$AEST[qualifiers...] image.EXE
AEST Processing
AEST creates a translated image in two phases:
1. Find and analyze code; read related .AIIF files.
2. Create translated image that includes IA64 code and original image.
Translated Image
image_AV.EXE
List File
image_AV.LIS
VM-1184A-AI

1.2.2 Translated Image Environment

The Translated Image Environment (TIE) provides the OpenVMS I64 system with the resources that a translated image needs in order to run. A variety of components work together to support execution of translated images:

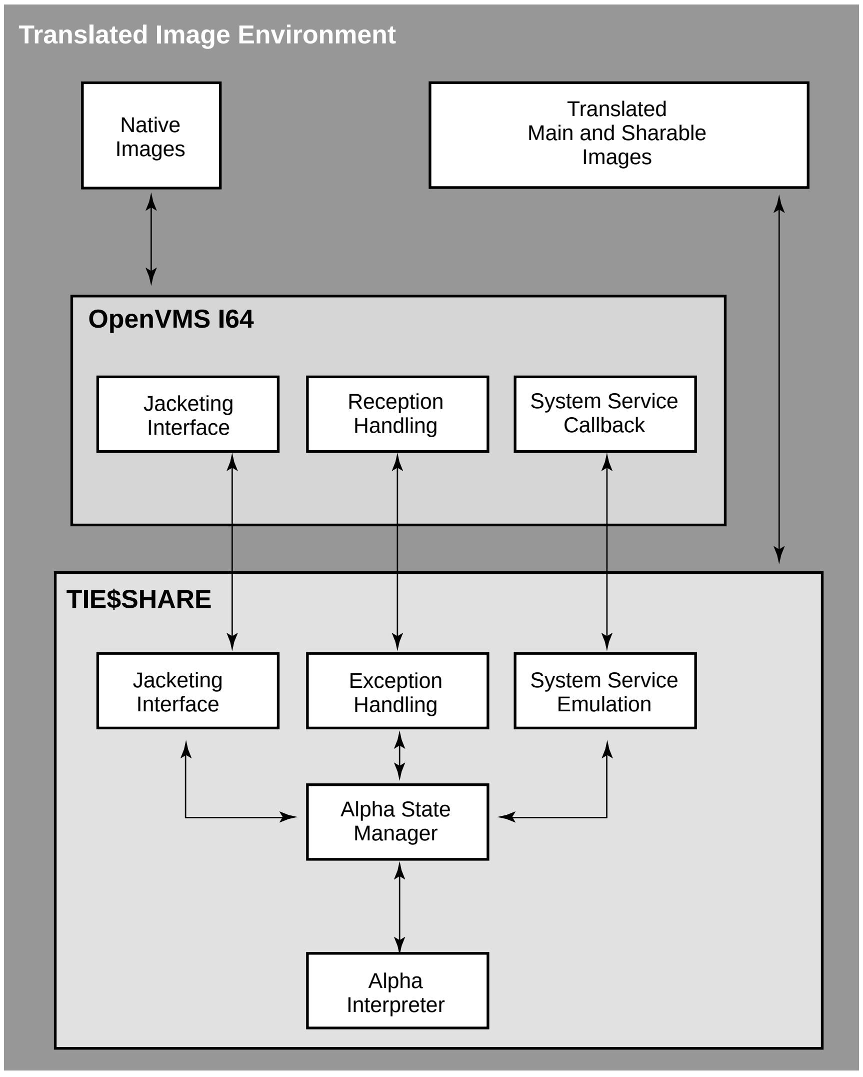
- The translated image itself, which includes both the original OpenVMS Alpha image and translated code. The translated code includes calls, inserted by AEST, to TIE$SHARE. These calls initiate processing that is not native to OpenVMS I64 systems.
- TIE$SHARE, which is an OpenVMS I64 shareable image. TIE$SHARE provides functions that enable the translated image to execute as if it were on an OpenVMS Alpha system. TIE$SHARE functions include:
  - Managing Alpha state information and other information that defines the relationship between the original Alpha code and the translated code.
  - Implementing OpenVMS Alpha features that the translated image requires, such as exception processing.
  - Interpreting Alpha code that AEST did not translate.
- Translated versions of some OpenVMS Alpha run-time libraries to be used by translated images.
- Native I64 and translated Alpha jacketing libraries, which mediate nonstandard calls from translated images to native I64 run-time libraries.

- Other features of the OpenVMS I64 operating system working cooperatively with TIE$SHARE to perform exception processing, to deliver ASTs and to enable communication between translated and native images.

Automatic jacketing, described in Chapter 6, provides the interoperability mechanism for most communication between translated and native images; it provides the bridge between the Alpha and I64 calling standards.

Figure 1–2 shows the interrelationship of the run-time components.

Figure 1–2 Run-Time Components



2 Translating Images

This chapter discusses the following:

What to consider before translating an image	Section 2.1
Running AEST to translate an image	Section 2.2

Related topics in other chapters include:

Special considerations when translating shareable images	Chapter 6
Running translated images	Chapter 3

2.1 Before Translating an Image

Not all images can be translated. Examples of such images are those that contain privileged code or images that must be run in privileged mode, or those that contain code written in a language that is not supported by AEST. To verify that your image can be translated, try running AEST with the /AUDIT qualifier, before you perform the actual translation of the image. Adding the /DUMP qualifier will produce a listing file containing some additional information on your image internal structure.

For further details about the /AUDIT qualifier, see the description of AEST in Appendix A.

2.2 Running AEST to Translate an Image

The following command translates an OpenVMS Alpha executable or shareable image:

```
AEST[/ qualifier,...] image [.EXE]
```

where *image* is the file name of the OpenVMS Alpha image to be translated. The default extension is .EXE.

Section 2.2.2 describes the AEST qualifiers. If the translation is successful, AEST creates the translated image in your current directory and names it by appending *_AV* to the input image file name, as follows:

```
image_AV.EXE
```

Note

A file name cannot exceed 39 characters in length. Because of this limitation, AEST truncates any input image file name that exceeds 36 characters in order to append the characters *_AV*.

If AEST encounters errors that prompt ERROR or FATAL level messages, it does not create a translated image. In this event, the messages explain why the translation was unsuccessful.

Example 2–1 shows the successful translation of an image called DHRYSTONE.EXE. In Example 2–1, the callouts note the following:

❶ A brief directory listing for an OpenVMS Alpha image called DHRYSTONE.EXE.
❷ The AEST command line to translate DHRYSTONE.EXE.
❸ A brief directory listing showing the original image and two new files created by AEST:

– DHRYSTONE_AV.EXE—the translated image

– DHRYSTONE_AV.LIS—the listing file
❹ A command to display the DHRYSTONE_AV.LIS file. This listing file first describes the version of AEST, the date and time of the translation, the command line issued, and header information for the image being translated.
❺ A summary of the AEST messages incurred during the translation. The summary categorizes the messages as follows:

– Standard messages AEST displays by default

– Verbose messages that report on AEST progress during translation

Example 2–1 Translating an Image

```
$ directory/brief  ❶
Directory AST_00:[AEST.TEST]
DHRYSTONE.EXE;1
Total of 1 file.
$ aest dhrystone   ❷
$ directory/brief  ❸
Directory AST_00:[AEST.TEST]
DHRYSTONE.EXE;1      DHRYSTONE_AV.EXE;1
DHRYSTONE_AV.LIS;1
Total of 3 files.
$ type dhrystone_av.lis   ❹
AEST XA29 DEV_0.05A (May  7 2004) starting at May 14 42004 18:28:06 with command line:  ❺
AEST DHRYSTONE.EXE
%AEST-I-TRANSOK, Translation completed successfully
```

AEST qualifiers, introduced in Section 2.2.2, allow you to tailor how AEST translates or analyzes an image.

2.2.1 AEST Return Status

When AEST completes its run, it returns one of the following messages as exit status to DCL:

- TRANSOK, Translation completed successfully
- TRANSWARN, Translation completed with warnings-review them before using the output image
- TRANSERROR, Translation unsuccessful-no output image created
- TRANSFATAL, Translation was impossible

The return status indicates the highest level of severity (INFO, WARNING, ERROR, or FATAL) of all the messages that AEST issued during its run.

2.2.2 AEST Qualifiers

The AEST command accepts qualifiers that control processing in various ways. For a complete description of the AEST command line and each of the qualifiers listed, see Appendix A.

2.2.3 AEST Output Files

AEST can generate several types of output files, depending on the type of image you are translating and the qualifiers you specify on the command line. Table 2–1 describes each type of output file, including its default name and the AEST qualifier that controls it.

Table 2–1 AEST Output Files

Default File Name	Qualifiers	Description
image_AV.EXE	/EXECUTABLE[=*filespec*] Default: /EXECUTABLE	The translated image. AEST truncates any input image file name that exceeds 36 characters in order to append _TV.
image_AV.LIS	/LIST[= *filespec*] Default: /LIST	The listing file.

2.3 Translation of VESTed images

The translation of VESTed images (original OpenVMS VAX images previously translated to OpenVMS Alpha images by the VEST utility from OMSVA Migration Software package) is similar to native OpenVMS Alpha images. No additional information files or AEST utility switches are required to translate VESTed applications.

3 Running Translated Images

This chapter discusses the following topics:

Running a translated image	Section 3.1
Handling references to a translated image	Section 3.2

3.1 Running the Translated Image

Normally, translated images can be run on OpenVMS I64 operating system in the same way as any other OpenVMS image. The OpenVMS operating system contains all components required. To run a translated image, use the DCL command RUN. For example, to run the sample program SIEVE_AV.EXE (translated in Chapter 1), enter the following command:

```
$ run sieve_av
Sieve of Eratosthenes
500 iterations
1899 primes found
time taken : 2 seconds
```

The Translated Image Environment (TIE) (see Section 1.2.2) issues error messages whenever it encounters errors while the translated image is running.

3.2 Handling References to a Translated Image

Depending on how a translated image is activated, you might need to change the name from *image*.EXE to *image_AV*.EXE:

If you are using the RUN command, just specify the translated image name as shown in Section 2.2. If the image name is specified in a command language definition (CLD) file, either modify the image name within the CLD file or define a logical name pointing the old name to the new name, as in following example:

```
$ DEFINE MYMAIN YOUR$DISK:[YOUR_DIR]MYMAIN_AV.EXE;
```

If a foreign command symbol is used to activate the image, change the symbol definition to specify the translated image name.

If your translated application includes shareable images that are not located in SYS$SHARE, you must define logical names that correctly point to them, that is, that reflect the correct location and translated image names. For example:

```
 $ DEFINE MYMATH_AV YOUR$DISK:[YOUR_DIR]MYMATH_AV.EXE;
```

Note

When you run a translated image linked against an HP supplied shareable image in SYS$SHARE, the OpenVMS I64 system automatically activates the correct image; you do not need to redefine the shareable image's logical name. For translated shareable images in SYS$SHARE not supplied by HP, you must explicitly define the appropriate logical name.

In some cases, you might want to redirect some external references to different shared images. You must write your own .AIIF file. For a description of .AIIF file format, see Chapter 5.

Part II

Developer’s Guide to Translating Images

Part II contains the following information:

Improving the performance of translated images; identifying problems in source code that can affect how you rebuild an application for the OpenVMS Alpha operating system	Chapter 4
Using image information files	Chapter 5
Creating translated and native shareable images	Chapter 6

4 Analyzing Images

This chapter describes how to use the /AUDIT qualifier to learn about image characteristics that affect translating and rebuilding an application.

4.1 Using the /AUDIT Qualifier

The /AUDIT qualifier instructs AEST to analyze an image and to provide a brief summary assessment that can help you decide on a migration strategy for your application. The summary, which is based on error messages issued during the AEST analysis, answers the following questions:

- Can the image be recompiled and rebuilt on an OpenVMS Industry Standard 64 system if the sources are available? Yes or No.
- Can the image be translated?
- What source languages were used?

For a detailed description of the /AUDIT qualifier, see Appendix A. The description includes a suggestion for building a file of summary descriptions by issuing a series of AEST/AUDIT commands and then using DCL commands to extract and compile the summary descriptions.

Example 4–1 shows the listing file for an audit of the SIEVE.EXE program.

Example 4–1 Audit Information for SIEVE.EXE

```
$ AEST/AUDIT SIEVE.EXE
AEST (V1.0, Bld DEV_0.8/Feb 21 2005)

$ TYPE SIEVE_AV.LIS
AEST (V1.0, Bld DEV_0.8/Feb 21 2005)
Image "SIEVE", "V1.0", 24-FEB-2005 15:29:40.96
<SUM>  Image name                                                  Tran Languages
<SUM> ------------------------------------------------------------ ---- ----------------
<SUM> $1$DGA120:[VERIFICATION.BINTRAN.EXAMPLES]SIEVE.EXE             YES  C
$
```

5 Using Information Files

This chapter discusses the Alpha Image Information files (.AIIF).

5.1 Alpha Image Information Files

Alpha Image Information files are text files that provide AEST with additional information to be used during image translation. AEST uses the information in the AIFF file to rename the image references in the Alpha file being currently translated. Such files are used to relink the translated image against the different shareable images and to modify fixup information.

The .AIIF file describes the properties of a shareable image's exported interface, that is, the precise locations that are described in the image's symbol vector. These files contain mapping between Alpha and Itanium entry points.

5.2 AIIF File Syntax

C-style multiline comments are allowed in any place in the file.

The following delimiters are allowed between fields of file line:

- ' '
- '\ t'
- ','

Two type of lines are allowed for .AIIF files:

1. The first type of line is needed for reassigning the old symbol vector index of a linked library to a new symbol vector index and probably in another library. Use the following format for this purpose:

   ```
   old_symv_idx new_symv_idx [, "new_name"]
   ```

   where:

Argument	Description	Usage
old_symv_idx	old symbol vector index	(as decimal)
new_symv_idx	new symbol vector index	(as decimal)
new_name	optional, newly linked library	(as ASCII, must be enclosed in double quotation marks (" "))

2. The second type of line is needed for assigning new GSMATCH fields for newly linked libraries. Use the following format for this purpose:

   ```
    "name"    match_ctl minor_id  major_id
   ```

Where:

Argument	Description	Usage
name	Name of newly linked library	(as ASCII, must be enclosed in double quotation marks (" "))
match_ctl	matching	(as decimal). Identifies the match algorithm used by the image activator. Specify one of the following values: 0 - ALWAYS 1 - EQUAL 2 - LEQUAL
minor_id	minor id	(as decimal)
major_id	major id	(as decimal)

6 Translating and Replacing OpenVMS Alpha Shareable Images

This chapter discusses the following topics:

Interoperability requirements	Section 6.1
Procedures for translating and replacing OpenVMS Alpha shareable images	Section 6.2

6.1 Interoperability Requirements

The OpenVMS Industry Standard 64 system allows translated and native images to interoperate by sending and receiving calls to and from one another. The calls are routed through system jacketing routines, which perform necessary conversions between the Alpha (or VAX) and the I64 calling standards. When you create native images that call or communicate with translated images, you need to use specific linking and compiling qualifiers. Furthermore, if you create a native shareable image that replaces an OpenVMS Alpha image, you need to maintain compatibility with that image.

The information in this chapter refers to the test C program called MYMATH to illustrate an Alpha shareable image, and a C program called MYMAIN to illustrate a main image that calls MYMATH. These example programs help to describe how to create native images that make calls to and receive calls from translated images, and highlight the following requirements for interoperability:

- Use the compiler qualifier /TIE and linker qualifier /NONATIVE_ONLY to create a native image that can interoperate with a translated image.
- Ensure upward compatibility between the symbol vectors in a native shareable image that supersedes an Alpha shareable image.

6.1.1 /TIE and /NONATIVE_ONLY Qualifiers

The /TIE and /NONATIVE_ONLY qualifiers instruct the compiler and linker, respectively, to include code that enables OpenVMS I64 systems to jacket calls to and from a translated image. When you specify the /TIE qualifier, the compiler creates procedure signature blocks (PSBs) that the Translated Image Environment (TIE) needs to jacket calls properly between translated and native images. When you specify the /NONATIVE_ONLY qualifier with the LINK command, the linker includes PSB information created by the compilers in the image. Note that the interoperability settings for these qualifiers are not the defaults. You must set the /TIE and /NONATIVE_ONLY settings explicitly. A native image does not interoperate with translated images unless you use these settings.

For further information about these qualifiers, see appropriate compiler documentation and the *HP OpenVMS Linker Utility Manual*.

6.1.2 Preserving Upward Compatibility

Successive versions of the same shareable image need to be upward compatible by maintaining the same calling interface so that calling images do not need to be relinked. The *HP OpenVMS Linker Utility Manual* describes how to maintain upward compatibility by using symbol vectors for OpenVMS Alpha and OpenVMS I64 system images. Maintaining upward compatibility is also advisable when creating any of the following:

- a translated shareable image
- a native image that replaces an OpenVMS Alpha shareable image
- a native image that replaces a translated shareable image

Just as you create consistent symbol vectors when building successive versions of the same image, you create consistent symbol vectors for translated images and native replacement images to achieve the same goal. This ensures upward compatibility as you migrate to OpenVMS I64 systems. When you link a native replacement image, construct the symbol vector so that it matches the transfer vector of the original image. If the native image includes new routines, place symbol vector entries for them after the older routine entries rather than disrupt the original order. You can use whatever mechanism is convenient to create a symbol vector that keeps the original order.

If you do not ensure that the symbol vectors maintain the same entry order, you run the risk of breaking calling images. AEST may create the correct symbol vector order when translating one version of the image, but not when translating a subsequent version of the image. From a different perspective, if you create a native replacement image without regard to the original order, translated images may not be able to call it because routines are not located at the expected address. If you have to create a jacket image because not all routines can be reproduced in native mode, you might have to rebuild the jacket image to accommodate the new entry order. This process is complicated and is not recommended. Use the procedures shown in this chapter instead.

For further information about transfer vectors, symbol vectors, and compatibility, refer to the *HP OpenVMS Linker Utility Manual* and *VAX MACRO and Instruction Set Reference Manual*.

6.2 Procedures for Building Shareable-Image Variants

This section describes procedures for the following tasks:

- Building the original OpenVMS Alpha shareable image (Section 6.2.1)
- Creating a translated shareable image (Section 6.2.2)
- Building a native replacement image (Section 6.2.3)

The OMSAI kit includes all the example programs used in this section, as well as command files to build and run them. After installing the kit, the source and command files are located in the AEST subdirectory of SYS$EXAMPLES. The command files must be executed in the following order:

1. First execute BUILD_MYMATH_AXP.COM on an OpenVMS Alpha system. The HP C compiler is required.
2. Then execute BUILD_MYMATH_IA64.COM on an OpenVMS I64 system. The HP C compiler is required.

The procedure descriptions all follow the same format:

- A description of the types of images being created (for example, a translated main program calling a translated shareable image).
- Command or code sequences introduced by headers demonstrate the procedure.
- When necessary, descriptions clarify what the command or code sequences are doing.

Note

These procedures are a simplified illustration of the fundamental steps of creating interoperable images and they do not consider all cases. You might have to alter this procedure to match your application requirements.

6.2.1 Building the Original OpenVMS Alpha Shareable Image

First, create and build the OpenVMS Alpha shareable image:

```
SYMBOL_VECTOR=( myadd=PROCEDURE,-
mysub=PROCEDURE,-
mydiv=PROCEDURE,-
mymul=PROCEDURE)
GSMATCH=LEQUAL,2,0
[EXIT]
```

Compile MYMATH, name the object file AXP_MYMATH, and link it to create the shareable image.

Then create the OpenVMS Alpha main image:

```
$ CC MYMAIN/OBJ=AXP_MYMAIN.OBJ
$ LINK AXP_MYMAIN.OBJ,SYS$INPUT/OPTIONS
AXP_MYMATH/SHAREABLE
 [EXIT]
```

Compile MYMAIN, name the object file AXP_MYMAIN, and link it to the shareable image AXP_MYMATH.

Then define logical name and run main image:

```
$ DEFINE AXP_MYMATH YOUR$DISK:[YOUR_DIR]AXP_MYMATH.EXE;
$ RUN/NODEBUG AXP_MYMAIN
```

Define the logical name AXP_MYMATH so that it points to the location of AXP_ MYMATH.EXE.

6.2.2 Creating the Translated Shareable Image

This section describes two procedures to be carried out on an OpenVMS Alpha system:

- Create a translated shareable image and a translated main image that calls it (Section 6.2.2.1).
- Create a translated image and a native main image that calls it (Section 6.2.2.2).

6.2.2.1 Translated Main Image Calls Translated Shareable Image

Suppose you have a main image that uses a shareable library on OpenVMS Alpha system. Translate both of them by entering the following commands:

```
$ AEST AXP_MYMAIN
$ AEST AXP_MYMATH
```

The translated images receive the names AXP_MYMAIN_AV .EXE and AXP_MYMATH_AV.EXE, respectively. If you simply run AXP_MYMAIN_AV on an OpenVMS I64 system, it will fail because the OpenVMS I64 system will look for AXP_MYMATH_AV.EXE in default directories.

After you translate the images, you have two possibilities: define a logical name pointing to the actual place where AXP_MYMATH_AV.EXE resides (preferred), or copy the translated image into one of these default directories.

Enter this command to define a logical name for the location of AXP_MYMATH_AV.EXE:

```
$ DEFINE AXP_MYMATH_AV YOUR$DISK:[YOUR_DIR]AXP_MYMATH_AV.EXE
```

Now you can run your translated image:

```
$ RUN AXP_MYMAIN_AV
```

6.2.2.2 Native Main Image Calls Translated Shareable Image

This procedure shows how to translate the OpenVMS Alpha shareable image and to create a native main image that calls it.

To translate an OpenVMS Alpha shareable image, enter this command:

```
$ AEST AXP_MYMATH
```

Create a native version of the main image MYMAIN called IA64_MYMAIN. Use the /TIE qualifier when compiling the source and the /NONATIVE_ONLY qualifier when linking the object file. Link IA64_MYMAIN and AXP_MYMATH_AV together in the same way you would link IA64_MYMAIN and a native shareable image. For example:

```
$ CC/TIE MYMAIN/OBJ=IA64_MYMAIN
$ LINK/NONATIVE_ONLY IA64_MYMAIN, SYS$INPUT:/OPTIONS
AXP_MYMATH_AV.EXE/SHAREABLE
[EXIT]
```

Define the logical name MYMATH_AV so that it points to the location of AXP_MYMATH_AV.EXE. When you run MYMAIN, it successfully calls AXP_MYMATH_AV. For example:

```
$ DEFINE AXP_MYMATH_AV YOUR$DISK:[YOUR_DIR]AXP_MYMATH_AV.EXE;
$ RUN IA64_MYMAIN
```

6.2.3 Building a Replacement Native Shareable Image

Compile the native I64 shareable image using the /TIE qualifier, and link it using the /NONATIVE_ONLY qualifier. Include a linker options file that orders the entries according to the order in the original OpenVMS Alpha shareable image. For example:

```
$ CC/TIE MYMATH/OBJ=IA64_MYMATH
$ LINK/SHAREABLE/NONATIVE_ONLY IA64_MYMATH, SYS$INPUT:/OPTIONS
SYMBOL_VECTOR=( myadd=PROCEDURE,-
mysub=PROCEDURE,-
mydiv=PROCEDURE,-
mymul=PROCEDURE)
GSMATCH=LEQUAL,2,0
 [EXIT]
```

Then define the logical name and run the main image:

```
$ DEFINE AXP_MYMATH_AV YOUR$DISK:[YOUR_DIR]IA64_MYMATH.EXE;
$ RUN/NODEBUG AXP_MYMAIN_AV
```

Part III

Reference Information

Part III contains the following information:

A detailed description of the AEST command line and qualifiers	Appendix A
An alphabetical listing of all AEST messages with explanations and recommended user actions	Appendix B
Debugging problems with translations	Appendix C

A
Command Summaries

AEST

The AEST utility translates OpenVMS Alpha executable and shareable images into functionally equivalent OpenVMS I64 system images. AEST also allows you to analyze OpenVMS Alpha images to assess their translatability.

Format

```
AEST[/qualifier,...] image[.EXE]
```

Qualifier	Default
/AIIF	None
/AUDIT	None
/DEBUG	/NODEBUG
/DUMP	/NODUMP
/EXECUTABLE	/EXECUTABLE
/INTERPRET	/NOINTERPRET
/LIST	/LIST
/VERBOSE	/NOVERBOSE

/AIIF

Instructs AEST to use the specified Alpha Image Information File (AIIF) when translating current image. Normally, AEST forms the translated image references by adding the suffix _AV to Alpha image names. The /AIIF qualifier allows you to change this behavior.

Default	None
Format	/[NO]AIIF=(*file-path*[,*file-path*...])
Qualifier Values	*filespec*[,*filespec*...]

Description

Alpha Image Information File (.AIIF file) contains the information used by AEST to rename the image references in the Alpha file that is currently translated. Such files are used to relink the translated image against the different shareable images and to modify fixup information.

The .AIIF file describes the properties of a shareable image's exported interface, that is, the precise locations that are described in the image's symbol vector. These files contain mapping between Alpha and I64 entry points.

The /NOAIIF qualifier tells AEST not to use the specified AIIF file, even if it is found.

AEST searches for .AIIF files in the following locations and in the following order:

1. The directory or directories specified as values to the /AIIF qualifier in the AEST command line.
2. Locations specified by logical names. The logical name for each AIIF file must be formed appropriately from the shareable image list plus the extension (.AIIF). For example, for the dynamic image SYS$PUBLIC_VECTORS, the appropriate logical name would be SYS$PUBLIC_VECTORS.AIIF.
3. The current directory.
4. The directory pointed to by the SYS$LIBRARY logical name.

/AUDIT

Instructs AEST to analyze the input image and to summarize its migration characteristics, but not to perform image translation. Note that, by default, AEST performs an audit before it translates the image, but it does not print verbose information into list file.

Format	/AUDIT
Qualifier Values	None

Description

The summary specifies:

- Whether the image is translatable
- Source language or languages of the image (if known)

The audit summary is a one-line description of an image's migration characteristics that may help you decide what migration option to choose for the image. You can find the summary information in the list file after all other messages. Example A–1 shows the format of the summary.

Example A–1 Summary Format

```
<SUM>  Image name                                                    Tran  Languages
<SUM> ------------------------------------------ --------------- ----  ---------------
<SUM> AST_00:[GROUP.TEST]DHRYSTONE.EXE                               YES  C
```

The image name column on the left provides the full file specification of the input image.

The two columns on the right define the image's migration characteristics:

Tran	YES or NO to indicate whether the image is translatable.
Languages	Lists the source languages identified in the image's debug symbol table (DST). The languages pertain only to the image being analyzed and not to any shareable images it calls.

By issuing a series of AEST/AUDIT commands and then using OpenVMS commands to extract the summary information, you can build a file of summary descriptions. For example:

```
$ AEST/AUDIT IMAGE1
$ AEST/AUDIT IMAGE2
.
.
.
$ AEST/AUDIT IMAGEn
$ SEARCH/OUTPUT=TEMP.1 *.LIS "<SUM>"
$ SEARCH/OUTPUT=TEMP.2 TEMP.1 "<SUM>"
$ SORT/NODUPLICATE TEMP.2 TEMP.3
$ PRINT TEMP.3
```

Using /AUDIT forces the following AEST qualifiers:

```
/NOEXECUTABLE
```

For example:

```
$ AEST/AUDIT DHRYSTONE
```

This example requests an audit summary for the sample program DHRYSTONE.EXE. The list file DHRYSTONE_AV.LIS contains the following text:

```
$ AEST/AUDIT DHRYSTONE
AEST (V1.0, Bld DEV_0.8/Feb 21 2005)
$
$ TYPE DHRYSTONE_AV.LIS
AEST (V1.0, Bld DEV_0.8/Feb 21 2005)
Image "DHRYSTONE", "V1.0", 24-FEB-2005 15:29:42.22
<SUM> Image name                                                    Tran Languages
<SUM> ------------------------------------------------------------ ---- ----------------
<SUM> $1$DGA120:[VERIFICATION.BINTRAN.EXAMPLES]DHRYSTONE.EXE          YES  C
$
```

/DEBUG

Instructss AEST to produce all verbose messages.

Default	The default is /DEBUG if the input image was linked with the /DEBUG qualifier or /NODEBUG if the input image was linked with /NODEBUG.
Format	/[NO]DEBUG
Qualifier Value	None

/DUMP

Default	/DUMP=NONE
Format	/DUMP=(list)
Qualifier Values	One or a combination of: ALPHA[=*list*], IA64[=*list*], INTERNAL[=*list*]

Description

/DUMP instructs AEST to print information about the source Alpha image, the resulting I64 image, and possibly some internal information into a listing file.

If /DUMP is specified it is equivalent to:

```
/DUMP=(ALPHA=ALL,IA64=ALL,INTERNAL=NONE)
```

/DUMP=([...] ALPHA[=...])

Default	ALL
Format	/DUMP=(ALPHA[=*keywords*])
Qualifier Values	NONE ALL—default, prints all information listed below, except for code NONE—prevents printing source image information HEADERS—prints image headers SECTIONS—prints list of image sections SHR_IMAGES—prints list of shared images referred to by the translated image FIXUPS—prints list of fixups RELOCATIONS—prints list of relocations SYMB_VECTOR—prints contents of symbol vector (if present) SYMB_TABLE—prints contents of symbol table (if present) INSTRUCTIONS—prints Alpha code instructions BLOCKS—prints Alpha code basic blocks with instructions ENTRY_POINTS—prints list of found entry points VERBOSE—adds some verbosity to listing, particularly to the section list

/DUMP=([...] IA64 [=...])

Default	ALL
Format	/DUMP=(IA64[=*keywords*])
Qualifier Values	ALL—default, prints all information listed below (except for BUNDLES) NONE—prevents printing of translated image information HEADERS—prints translated image headers (in internal representation) SEGMENTS—prints brief list of translated image segments SHR_IMAGES—prints list of shared images referred by the translated image FIXUPS—prints list of fixups RELOCATIONS—prints list of relocations SYMB_VECTOR—prints contents of symbol vector (if present) SYMB_TABLE—prints contents of symbol table (if present) BUNDLES—prints generated I64 code BLOCKS—prints Alpha basic blocks together with correspondent I64 bundles. Initialization code (and other generated I64 code) is printed as well. TII—prints TIE information structures TRANS_VECTOR—prints transfer vector GOT—prints global offset table contents

/DUMP=([...,] INTERNAL [=...])

This qualifier is intended for problem-solving purposes only. Normally, you do not need the corresponding printouts.

Default	ALL
Format	/DUMP=(INTERNAL[=*keywords*])
Keywords	ALL—prints all information below NONE—prevents printing INTFIX—prints internal fixups (don´t appear in translated image) INTSYM—prints internal symbols (don´t appear in translated image) FDS—prints list of function descriptors generated by translator.

/EXECUTABLE

Enables the creation of a translated image and, optionally, includes a file specification.

Default	/EXECUTABLE
Format	/EXECUTABLE [=*filespec*]
Qualifier Values	*filespec*

Description

By default, AEST creates a translated image unless the command line specifies qualifiers that are incompatible (the /AUDIT qualifier, for example), and as long as AEST does not encounter error conditions that prevent image translation. If AEST issues ERROR or FATAL messages, it does not create a translated image.

If you do not provide a file specification, AEST writes the translated image to the current directory and names it by appending _AV to the input image's file name. For example, if the name of the OpenVMS Alpha image is PROGRAM.EXE, the default name of the translated image is PROGRAM_AV.EXE.

Note

A file name cannot exceed 39 characters in length. Because of this limitation, AEST truncates any input image file name that exceeds 36 characters to append the characters _AV.

/INTERPRET

Controls whether image´s code should be translated or just interpreted in run-time.

Default	/NOINTERPRET
Format	/INTERPRET
Qualifier Values	None

Description

AEST tries to find, parse, and translate as much Alpha code as possible to minimize the need for interpreting Alpha code at run time. By default, AEST uses the EISD$V_EXE ISD flag to translate all code that it finds in image sections marked as "executable."

If you specify /INTERPRET, AEST still analyzes all the code found in "executable" Alpha image sections and creates corresponding structures in the resulting I64 image, but it did not generate corresponding I64 code. This minimizes the AEST execution time and virtual memory required for translation.

Note that run-time performance of a translated image using /INTERPRET will be slower than one where the code is translated.

/LIST

Requests a list file and, optionally, specifies a file name.

Default	/LIST
Format	/LIST [=*filespec]*

Qualifier Values	*filespec* Identifies a file specification for the list file.

Description

If you do not specify a file specification, AEST writes the list file to the current directory, names it by appending _AV to the input image's file name, and uses the extension LIS. For example, if the input image is SIEVE.EXE, the default list file is called SIEVE_AV.LIS.

/VERBOSE

Instructs AEST to produce all verbose messages.

Default	/NOVERBOSE
Format	/[NO]VERBOSE
Qualifier Value	None

This example demonstrates using the /VERBOSE option.

```
$ AEST/VERBOSE DHRYSTONE
AEST V1.0 DEV_0.8 (Feb 21 2005) starting at Feb 25 2005 15:38:05 with command line:
AEST/VERBOSE DHRYSTONE Reading Source image...
Done.
Analyzing Alpha image code...
Done.
Translating data structures...
Done.
Translating code...
Done.
Adding init code...
Done.
Building TIE information...
Done.
Processing external symbols...
Done.
Fixing internal structures...
Building SDS...
Fixing code...
Freeing code...
Building TIE segment...
Fixing data...
Done.
Preparing to write target image...
Done.
Writing target image...
Done.
%AEST-I-TRANSOK, Translation completed successfully
$
$ TYPE DHRYSTONE_AV.LIS
AEST V1.0 DEV_0.8 (Feb 21 2005) starting at Feb 25 2005 15:38:05 with command line:
AEST/VERBOSE DHRYSTONE %AEST-I-TRANSOK, Translation completed successfully
$
```

B
Error and Status Messages

This appendix discusses the following topics:

Interpreting OMSAI messages	Section B.1
Alphabetical list of AEST messages	Section B.2

B.1 Interpreting AEST Messages

AEST error and status messages identify many different conditions within an image. For example, AEST can identify code that prevents successful translation. Each message has one of the following severity levels:

- INFO messages provide descriptive information about the AEST run.
- WARNING messages describe questionable code encountered that you need to investigate. WARNING messages do not prevent AEST from creating a translated image.
- ERROR messages indicate a problem in the code that prevents AEST from translating the input image.
- FATAL messages indicate a problem that prevents the translation altogether (input file not found, for example).

Section B.2 provides explanations for each message and, when appropriate, suggests a user action. An AEST message as it is displayed or written to the log file consists of the facility name (AEST), a letter indicating the severity level (I, W, E, or F), a message identifier, and a brief explanation of the error or status. For example:

```
%AEST-E-HASSECNAME, Image contains secondary image name -- not translatable
```

B.2 AEST Messages Descriptions

AIIF_COLLISION

Explanation: .AIIF file contains inconsistent information. Match control fields for some newly linked images are defined more than once, and the definitions are contradictory.

BADEXE

Explanation: Executable image has a bad format. This problem occurs when a user tries to translate a file that is not a valid OpenVMS Alpha image.

BADVEST

Explanation: OpenVMS Alpha image to be translated is produced by unsupported VEST versions (prior to VEST 1.0)

HASSECNAME

Explanation: The translatable image is a CLI image. The translation of CLI images is not allowed.

INVALID_SYMBOL

Explanation: The Symbol virtual address is out of range (it does not find any translated segment).

ISDINIT

Explanation: The image contains an OpenVMS initialization section, and therefore, is not translatable.

ISDRESIDENT

Explanation: The image contains a memory resident section, and therefore, is not translatable.

LNKEXEC

Explanation: Translatable image is linked against a specific OpenVMS Alpha kernel version and is not translatable.

LNKSYS

Explanation: Translatable image is linked against OpenVMS Alpha and references symbols in it. It is not translatable.

OPENIN

Explanation: Image file to be translated cannot be opened.

PRIVOPC

Explanation: Privileged instruction (opcode) was detected in the OpenVMS Alpha image. Translated image has been generated, but the execution of this image will fail if execution with privileged Alpha instruction is requested.

RTLNOTSUPP

Explanation: The OpenVMS Alpha image to be translated was linked against a run-time library not officially supported by OMSAI. The translated image has been generated, but the correct execution of this image is not guaranteed.

TRANSERROR

Explanation: Translation completed with errors, although executable image has been created. It is likely that the translated image will not work correctly, so please check all error messages issued during the translation.

TRANSFATAL

Explanation: Translation did not complete because of fatal errors and translated executable image was not generated. The translation of this image is impossible.

TRANSOK

Explanation: Translation completed successfully and a translated image has been generated.

TRANSWARN

Explanation: Translation completed with some warnings, but translated executable image was generated. Please review warning message before using translated image.

C
Translation and Performance Restrictions

This appendix discusses the following topics:

How to identify images with translation or performance restrictions	Section C.1
Images that cannot be translated	Section C.2
Images that translate with WARNING error messages	Section C.3
Images with compatibility problems not detectable during translation	Section C.4

C.1 Identifying Restrictions and Performance Issues

You can use the Alpha Environment Software Translator (AEST) utility to identify most translation restrictions in OpenVMS Alpha images. During the code analysis phase, AEST issues messages that flag and describe problematic code it encounters. If any message severity is either FATAL or ERROR, AEST does not create a translated image. If the most severe message level is WARNING, AEST creates a translated image that can run properly on an OpenVMS I64 system. However, HP recommends that you examine each WARNING message carefully.

This appendix identifies specific AEST messages that correspond to restrictions and performance issues. See the message explanations in Appendix B for descriptions of the coding problems identified.

C.2 Untranslatable Images

Images described in Table C–1 incur FATAL or ERROR messages.

Table C–1 Untranslatable Images

Description	Message
Images linked with OpenVMS Alpha version earlier than 6.1	BADEXE
Images that have user-written system services or other nonuser-mode code	HASSECNAME LNKEXEC ISDINIT ISPROTECT
Images linked against a specific version of OpenVMS	LNKSYS
VESTed binary images produced by DECmigrate tool in case the original OpenVMS VAX image was linked on an OS version earlier than 5.5	BADVEST

C.3 Images Translatable with Warnings

Images described in Table C-2 incur WARNING messages. HP recommends that you examine each WARNING message carefully. You might determine that the code flagged is not likely to interfere with the image running translated. In other cases, you might need to take steps to ensure that the translated version of the image executes properly on an OpenVMS I64 system. You might need to make changes to the source code, if available, and then recompile and relink the image before translating; or you might need to relink the image with different options.

Note that AEST can also issue WARNING messages for reasons other than those listed in Table C–2.

Table C–2 Images Translatable with Warnings

Description	Message
Images that contain privileged instructions	PRIVOPC
Images that reference standard OpenVMS run-time libraries not supported in the current version of OMSAI	RTLNOTSUPP

C.4 Images with Undetectable Translation Problems

Unfortunately, some problems cannot be found during the translation of OpenVMS Alpha images and occur only when translated applications are running on OpenVMS I64 system. These problems include the following:

- Images that use user-mode thread managers other than DECthreads/Pthreads
- Images that implement their own thread management
- Images that use system services to directly access kernel threads mechanisms.