OpenVMS Migration Software for VAX to Alpha Systems

(formerly DECmigrate)

Translating Images

June 2002

This manual describes how to use the VAX Environment Software Translator (VEST) and other OpenVMS Migration Software for VAX to Alpha Systems tools for translating and porting OpenVMS VAX applications to OpenVMS Alpha systems.

Revision/Update Information:	This is a new manual.
Operating System and Version:	OpenVMS Alpha Version 6.2 or higher
Software Version:	OpenVMS Migration Software for VAX to Alpha Systems 1.2

Compaq Computer Corporation
Houston, Texas

Table of Contents

Preface

Intended Audience

OpenVMS Migration Software for VAX to Alpha Systems (OMSVA) facilitates migrating OpenVMS VAX applications to OpenVMS Alpha systems by allowing you to translate OpenVMS VAX images into equivalent OpenVMS Alpha images. OMSVA consists of the VAX Environment Software Translator (VEST) utility and a collection of programs and command files designed to ease the translation process. OpenVMS Migration Software for VAX to Alpha Systems Translating Images documents the VEST utility and explains its use as part of a strategy for migrating OpenVMS VAX applications to OpenVMS Alpha systems.

This manual addresses:

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

In addition, this manual describes VEST's analytical capability and addresses:

- Users who are analyzing OpenVMS VAX images to help determine the best migration strategy for an application
- Users who are analyzing OpenVMS VAX images to facilitate migrating source files to OpenVMS Alpha or modifying source files to create translatable OpenVMS VAX images

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 VAX images.
- Chapter 3 describes how to run translated images on OpenVMS Alpha 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 VEST to enhance translation and to identify source problems that affect migration.
- Chapter 5 describes image information files, which VEST 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 Alpha system.

Part III: Reference Information

Information in Part III is applicable to all user categories:

Appendix A provides a detailed description of the DSTGRAPH, FLOWGRAPH, VEST, and VEST /DEPENDENCY command lines and qualifiers.

Appendix B provides an alphabetical listing of all VEST, FLOWGRAPH, and DSTGRAPH error messages with explanations and recommended user actions, if applicable.

Appendix C describes translation problems and suggests ways to debug them.

Appendix D lists coding practices and other restrictions that affect the translatability of OpenVMS VAX images.

Appendix E lists all the VAX instructions in alphabetical order and describes how VEST handles each one.

Associated Documents

The following OpenVMS Alpha manuals also pertain to migrating OpenVMS VAX applications:

Migrating to an OpenVMS Alpha System: Planning for Migration provides an overview of the OpenVMS VAX to OpenVMS Alpha migration process and information to help you plan a migration. It discusses the decisions you must make in planning a migration and the ways to get the information you need to make those decisions. In addition, it describes the migration methods available so that you can estimate the amount of work required for each method and select the method best suited to a given application.

Migrating to an OpenVMS Alpha System: Recompiling and Relinking Applications describes how to build an OpenVMS Alpha version of your OpenVMS VAX application by recompiling and relinking it. It discusses dependencies your application may have on features of the VAX architecture (such as assumptions about page size, synchronization, and condition handling) that may need modifying to create a native OpenVMS Alpha version. In addition, it describes how you can create applications in which native OpenVMS Alpha components interoperate with translated OpenVMS VAX components.

Migrating to an OpenVMS Alpha System: Porting VAX MACRO Code describes how to port MACRO code to an OpenVMS Alpha system using the VAX MACRO-32 Compiler for OpenVMS Alpha. It describes the features of the compiler, presents a methodology for porting VAX MACRO code, identifies nonportable coding practices, and recommends alternatives to such practices.

The manual also provides a reference section with detailed descriptions of the compiler's qualifiers, directives, and built-ins, and the system macros created for porting to OpenVMS Alpha Systems.

Conventions

The following conventions are used in this manual:

. . .	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 format descriptions, parentheses indicate that, if you choose more than one option, you must enclose the choices in parentheses.

	one option, you must enclose the choices in parentheses.
[ ]	In format descriptions, brackets indicate optional elements. You can choose one, none, or all of the options. (Brackets are not optional, however, in the syntax of a directory name in an OpenVMS file specification, or in the syntax of a substring specification in an assignment statement.)
{ }	In format descriptions, braces surround a required choice of options; you must choose one of the options listed.
boldface text	Boldface text represents the introduction of a new term or the name of an argument, an attribute, or a reason. Boldface text is also used to show user input.
italic text	Italic text emphasizes important information, indicates variables, and indicates complete titles of manuals. Italic text also represents information that can vary in system messages (for example, Internal error number), command lines (for example, /PRODUCER= name), and command parameters in text.
UPPERCASE TEXT	Uppercase text indicates a command, the name of a routine, the name of a file, or the abbreviation for a system privilege.
-	A hyphen in code examples indicates that additional arguments to the request are provided on the line that follows.
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

The chapters in Part I contain the following information:

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

1. Introduction to Image Translation

This chapter discusses the following topics:

Topic	See
An overview of OMSVA	Section 1.1
A description of OMSVA tools and support	Section 1.2

1.1 Overview of OMSVA

OpenVMS Migration Software for VAX to Alpha Systems (OMSVA) facilitates migrating OpenVMS VAX applications to OpenVMS Alpha systems. A OMSVA utility, the VAX Environment Software Translator (VEST), converts an OpenVMS VAX executable or shareable image into a translated image that runs on an OpenVMS Alpha system. When the translated image runs, the OpenVMS Alpha system transparently supports the image with an environment that allows it to run as if it were on an OpenVMS VAX system. In that support environment, the average translated image runs as fast as or faster than the original running on an equivalent OpenVMS VAX system.

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

```
$ vest sieve 1
$ run sieve_tv 2
Sieve of Eratosthenes 3
500 iterations
1899 primes found
time taken : 2 seconds
```

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

1 The VEST command used to translate an image. The command assumes the file extension to be .EXE.

2 The DCL command used to run the translated image. The same command is used to run SIEVE.EXE on an OpenVMS VAX system.

3 The output displayed by the translated image.

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

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

1.1.1 OMSVA Features

OMSVA features include:

Language independence

OMSVA translates an image regardless of the source language.

Automated translation

OMSVA translates an image automatically; it requires no human intervention to analyze and translate code.

Image analysis

OMSVA analyzes an image and reports its findings in various forms (messages, listings, and graphs, for example); these findings are useful not only for translation, but also for preparing original sources for recompiling and relinking on an OpenVMS Alpha system.

VEST cannot translate all OpenVMS VAX images; some restrictions apply. For example, VEST will not translate images linked on versions prior to Version 4.0. Furthermore, VEST does not support certain coding practices because the OpenVMS Alpha system cannot reproduce the corresponding VAX code correctly. VEST issues an error message when it encounters unsupported code and may or may not create a translated image, depending on the specific problem reported.

See Appendix D for a summary of coding practices that are either not supported or not recommended.

1.1.2 OMSVA Roles within a Migration Strategy

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

Rebuilding application source files

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

Translating the application's OpenVMS VAX images

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

Combining 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 third option is possible because the OpenVMS Alpha 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. For a more detailed discussion of your migration options, see the OpenVMS Alpha manual Migrating to an OpenVMS Alpha System: Planning for Migration.

1.2 Image Translation Tools and Support

The image translation tools and support include:

The VEST utility, the primary translation tool (see Section 1.2.1).

The VEST/DEPENDENCY command, which determines the order in which to translate a main image and any shareable images it refers to (see Section 1.2.2).

FLOWGRAPH command, which creates graphical representations, called flowgraphs, of the OpenVMS VAX image that VEST has analyzed. The flowgraphs are in PostScript format and are based on VEST-generated files (see Section 4.3).

DSTGRAPH command, which illustrates unaligned and skewed data items that VEST has found in an OpenVMS VAX image. The illustrations are in PostScript format. VEST can collect the necessary information only if the image has been compiled and linked with the /DEBUG qualifier (see Section 4.4).

Translated Image Environment (TIE), a native shareable image and other components within the OpenVMS Alpha operating system that supports translated images at run time (see Section 1.2.4).

1.2.1 VAX Environment Software Translator Utility

The VEST utility translates executable and shareable images; it accepts an OpenVMS VAX image file (image.EXE) as input, analyzes the image file to locate VAX code, and then creates a translated image file (image_TV.EXE). The translated image, which performs the exact same functions as the original, is an OpenVMS Alpha image consisting of both Alpha AXP code and the original OpenVMS VAX image. This section first describes text files VEST uses for finding and analyzing code and then briefly explains how VEST works.

1.2.1.1 VEST Information Files

Information files are text files that provide VEST with additional information to be used during image translation. The types of information files are as follows:

Image information files

An image information file (also called an .IIF file) describes the interface to a shareable image by detailing its entry point properties. When an input image has been linked against a shareable image, VEST reads the .IIF file for that shareable image. Information about its entry points allows VEST to generate translated code that properly creates the linkage between the main and shareable images. When VEST translates a shareable image, it creates an .IIF file for the image. When VEST translates another image that refers to the shareable image, it reads the .IIF file created when that shareable image itself was translated. The OpenVMS Alpha operating system includes .IIF files for all the translated and native system run-time libraries (RTLs). See Chapter 5 for further details.

Hand-edited information files

A hand-edited information file (also called an .HIF file) contains information about the input image VEST is currently translating. The contents of the .HIF file either override assumptions VEST would make about specific entry points in the image, or augment what VEST is able to find out on its own. See Chapter 5 for further details.

Symbol information files

A symbol information file (also called a .SIF file) is a text file that precisely describes and controls the contents of the global symbol table (GST) and symbol vector in a translated shareable image. Chapter 6 explains when and how to use .SIF files.

Library information files

A library information file (also called a .LIF file) is a text file that describes how to rename shareable image references in the translated image depending on the referenced image section id. This feature allows keeping multiple translated versions of an image (each with its own .IIF file) on the same system.

Chapter 5 explains how VEST accesses the information files.

1.2.1.2 How VEST Works

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

1.2.1.3 Code Analysis-Pass 1 and Pass 2

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

VEST makes two passes over the image, pass 1 and pass 2, in its search for code. During pass 1, VEST searches for and reads an .IIF file for every shareable library referenced by the input image. VEST also searches for an optional .HIF file for the input image itself. The .HIF file helps VEST resolve ambiguities or locate hidden code. During pass 2, VEST tries to parse and translate portions of the image that did not yield code during pass 1. As VEST analyzes the input image, it saves information about the image's program flow and structure. VEST eventually uses the information to create the translated image and, if requested, to produce a flowgraph file. You can use the flowgraph file to generate PostScript output that graphically represents all or part of the input image in selectable levels of detail. The flowgraphs are a useful means of understanding program flow and pinpointing locations in the image that represent problem areas in the source.

1.2.1.4 Code Generation

The second phase of translation generates the translated image, an Alpha AXP image that includes translated code as well as the complete original OpenVMS VAX image. Translated code is native Alpha AXP code that performs the same function as the corresponding VAX code in the original image. When the translated image runs on an OpenVMS Alpha system, it reproduces the behavior of the original image. VEST command line qualifiers allow you to control the code generation phase to direct tradeoffs between precise VAX architecture conformance and run-time performance on the OpenVMS Alpha system.

If the input image is shareable, VEST also generates an .IIF file, named image.IIF, for the image. Section 5.2 describes how VEST uses image.IIF files.

1.2.2 VEST/DEPENDENCY Command

The VEST/DEPENDENCY command identifies the dependencies that one or more images have on shareable images. Using this information, you can determine the correct order in which to translate a set of images. By translating images in the correct order, you ensure that VEST has the information it needs about shareable images as it translates each image. (See Section 5.2.)

VEST/DEPENDENCY output includes:

A DEC/Module Management System (MMS) for OpenVMS VAX description file. If your system includes MMS, a component of the optional layered product DECset for OpenVMS Alpha, you can use a command file OMSVA provides to invoke MMS and pass it the description file. The description file directs MMS to execute a series of VEST commands that translates a set of OpenVMS VAX images in the right order.

A dependency graph file that you process to create a PostScript representation of the image dependencies. You can use the image dependency graph to determine the correct order in which to translate the images.

Section 2.3 describes how to use the VEST/DEPENDENCY command, how to use the MMS input file, and how to process the dependency graph file.

1.2.3 FLOWGRAPH Command

The FLOWGRAPH command creates one or more flowgraphs, formatted in PostScript, that represent all or part of an image's program flow or that show an image's dependency on shareable images. The command bases the flowgraphs on an input file created by a VEST or VEST /DEPENDENCY command. Depending on the selections you make when you issue the VEST command, a flowgraph can illustrate the internal flow of an entire image, can illustrate the parts of the image that prompted VEST to issue an error message, or can illustrate the image's calling structure. The VEST/DEPENDENCY command produces a file from which the FLOWGRAPH command creates a dependency graph.

Section 4.3 describes how to use the VEST and FLOWGRAPH commands.

1.2.4 Translated Image Environment

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

The translated image, which includes both the original OpenVMS VAX image and translated code. The translated code includes calls, inserted by VEST, to TIE$SHARE. These calls initiate processing that is not native to the OpenVMS Alpha operating system.

TIE$SHARE, which is an OpenVMS Alpha shareable image. TIE$SHARE provides functions that enable the translated image to execute as if it were on an OpenVMS VAX system. TIE$SHARE functions include:

- Managing VAX state information and other information that defines the relationship between the original VAX code and the translated code.
- Implementing OpenVMS VAX features that the translated image requires, such as exception processing and asynchronous system trap (AST) delivery.
- Interpreting VAX code that VEST did not translate.

TIE$EMULAT_TV, which is a translated shareable image that TIE$SHARE calls to perform complex VAX instructions.

Other features of the OpenVMS Alpha operating system, which work 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 VAX and Alpha AXP calling standards.

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

2. Translating Images

This chapter discusses the following:

Topic	See
What to consider before translating an image	Section 2.1
Running VEST to translate an image	Section 2.2
Using the VEST/DEPENDENCY command to identify image dependencies	Section 2.3

Related topics in other chapters include:

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

2.1 Before Translating an Image

Some questions to consider before translating an image are as follows:

Do you want a summary of an image's migration characteristics, that is, features of the image that affect decisions to rebuild or translate?

> The VEST qualifier /AUDIT analyzes an image and generates a brief summary of migration characteristics, including whether or not the image sources can be recompiled; whether or not the image is translatable; whether or not the image includes code that adversely affects performance; and what the source language or languages are. See the Appendix A description of VEST for further details about the /AUDIT qualifier.

Are you translating images in the correct order?

> If the OpenVMS VAX image to be translated depends on user-supplied shareable libraries, use the VEST /DEPENDENCY command to identify the image dependencies and the proper order in which to translate the interrelated images (Section 2.3). You need to translate images in the proper order to ensure that required .IIF files are accessible to VEST as it translates each image. The OpenVMS Alpha operating system includes .IIF files for all the translated system libraries. If the DEC /Module Management System (MMS) for OpenVMS VAX is available on your system, you can use the .MMS description file that VEST/DEPENDENCY generates to do the translations in the correct sequence. If MMS is not available, you can use VEST/DEPENDENCY to create a graph that illustrates the dependencies.

Are required information files available to VEST?

> Make sure that any information files VEST needs to translate the image are available. VEST searches for relevant .IIF, .HIF, and .SIF files in the following locations and in the following order:
>
> The current default directory
>
> The directory or directories specified as values to the qualifier /INCLUDE_DIRECTORY, if present in the VEST command line
>
> The directory or directories, if any, defined by the VEST$INCLUDE logical name

Does the OpenVMS VAX image include debugging information?

VEST uses information in an OpenVMS VAX image's debugger symbol table (DST) to find code and to trace an error encountered at a specific address in the image back to the line of source corresponding to that address. If you are using VEST to help you identify migration problems in source files, you need the DST information to be present. Section B.1.1 describes the relationship between DST information and the /DEBUG and /TRACEBACK qualifiers used to create the image.

2.2 Running VEST to Translate an Image

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

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

Where image is the file name of the OpenVMS VAX image to be translated and the default extension is .EXE. Section 2.2.2 describes the VEST qualifiers.

If the translation is successful, VEST creates the translated image in your current directory and names it by appending "_TV" to the input image file name as follows:

```
image_TV.EXE
```

Note

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

If VEST 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. (See Appendix D for the definition of a translatable OpenVMS VAX image.)

Example 2-1 shows the successful translation of an image called DHRYSTONE.EXE.

In Example 2-1, the callouts note the following:

1 A brief directory listing for an OpenVMS VAX image called DHRYSTONE.EXE.

2 The VEST command line to translate DHRYSTONE.EXE.

3 A brief directory listing showing the original image and two new files created by VEST:

DHRYSTONE_TV.EXE-the translated image

DHRYSTONE_TV.LIS-the listing file

4 A command to display the DHRYSTONE_TV.LIS file. The amount of information included in the listing file depends on the /SHOW qualifier setting. See the description of /SHOW in Appendix A for details. This listing file first describes the version of VEST, the date and time of the translation, the command line issued, and header information for the image being translated.

5 A summary of the VEST messages incurred during the translation. The summary categorizes the messages as follows:

- Standard messages VEST displays by default
- Performance messages that note code in the input image that may cause the translated image to run more slowly
- Source analysis messages that point out unconventional code patterns in the input image that might indicate latent bugs or nonportable features in the source
- Synchronization messages that note possible uses of features requiring some form of synchronization
- Verbose messages that report on VEST progress during translation

The amount of detail included in the listing file about these messages depends on the /WARNINGS qualifier setting. See the description of /WARNINGS in Appendix A. Appendix B provides a complete listing of all the VEST messages alphabetized by the message identifier (for example, READING). Refer to Appendix B for an explanation of each message and suggested user actions, if any.

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

2.2.1 VEST Return Status

When VEST 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 VEST issued during its run. The only exception to this rule is that VEST ignores WARNING messages that you have explicitly disabled. (See the description of /WARNING in Appendix A.)

VEST/DEPENDENCY returns the equivalent status messages prefixed with DEPEND (DEPENDOK, for example).

2.2.2 VEST Qualifiers

The VEST command accepts qualifiers that control processing in various ways. Table 2-1 categorizes the qualifiers by function. Refer to Appendix A for a complete description of the VEST command line and each of the qualifiers listed.

2.2.3 VEST Output Files

VEST 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 2-2 describes each type of output file, including its default name and the VEST qualifier that controls it.

2.3 Using VEST/DEPENDENCY to Identify Image Dependencies

The VEST/DEPENDENCY command analyzes one or more input images to identify external references to shareable images. If a referenced shareable image in turn refers to another shareable image, VEST/DEPENDENCY searches that image for references to any shareable images, and so on. The VEST/DEPENDENCY command has its own set of qualifiers and its own syntax; see Appendix A for details.

You can choose how the command deals with the dependency information it collects:

VEST/DEPENDENCY/MMS_DESCRIPTION, the default, creates an MMS description file called image.MMS. Section 2.3.1 explains how to use a OMSVA command file to execute the MMS description file.

VEST/DEPENDENCY/FLOWGRAPH creates a file called image.GRAPH. Section 2.3.2 explains how to submit this file to the FLOWGRAPH command to create an image dependency graph.

In the following example, VEST/DEPENDENCY analyzes the SIEVE.EXE image's dependencies on other shareable images, by default creates an .MMS file, and requests a dependency graph file:

```
    $ vest/dependency/flowgraph sieve
    %VEST-I-READIMAGE, Reading image file VST_00:[VEST.TEST]SIEVE.EXE;
    %VEST-I-READIMAGE, Reading image file SYS$COMMON:[SYSLIB]VAXCRTL.EXE;
    %VEST-I-READIMAGE, Reading image file SYS$COMMON:[SYSLIB]LIBRTL.EXE;
    %VEST-I-READIMAGE, Reading image file SYS$COMMON:[SYSLIB]UVMTHRTL.EXE;
```

VEST/DEPENDENCY creates the following files in the current directory:

SIEVE.MMS-An MMS description file

SIEVE.GRAPH-An input file used to create a PostScript graph of image dependencies

2.3.1 Using the VEST_MMS_DRIVER.COM Command File

If you have MMS on your system, you can use the OMSVA command file VEST_MMS_DRIVER.COM to execute the MMS description file. Look for VEST_MMS_DRIVER.COM in the SYS$SYSTEM: directory, which is where the OMSVA installation procedure places the command file. The description file instructs MMS to issue VEST commands in the right order as appropriate; no command is issued to translate any image for which an up-to-date translated version is already available.

Use the VEST_MMS_DRIVER.COM command file to process the MMS description file; do not submit it directly to MMS. The command file defines the logical name VEST$FULL_INCLUDE, required by the MMS description file to execute correctly. The MMS description file itself contains definitions that describe all the components for the translation and the command needed to invoke VEST. Depending on individual circumstances, you may choose to edit the MMS file before using it.

The syntax for invoking the command file is as follows:

```
    @VEST_MMS_DRIVER image [.MMS]
```

Where:

image[.MMS] is the name of the MMS description file.

The following example processes the SIEVE.MMS file:

```
    $ @vest_mms_driver sieve
```

In this example, SIEVE.EXE is the only image actually translated. The shareable libraries SIEVE refers to are VAXCRTL, LIBRTL, and MTHRTL. The translated versions of these libraries and their information files already exist; retranslating them is unnecessary.

2.3.2 Processing the Dependency Graph File

When the command line includes the /FLOWGRAPH qualifier, the VEST/DEPENDENCY command creates a .GRAPH file. The FLOWGRAPH command (see Section 4.3) converts the .GRAPH file into a PostScript image dependency graph. This graph displays the dependencies as a tree, as shown in

Figure 2-1.

If MMS is not available, you can translate the images in the proper order by starting with the image at the bottom of the tree and moving up. You do not need to retranslate any libraries for which .IIF files are already available.

The following example uses the FLOWGRAPH command to create a dependency graph of the image SIEVE.EXE called SIEVE.PS and then queues it to a PostScript printer:

```
    $ FLOWGRAPH SIEVE
    $ PRINT/QUEUE=HALL01/PARAM=(DATA=POSTSCRIPT) SIEVE.PS
```

The default input file extension is .GRAPH; the default output file takes the name of the input file and appends the extension .PS. Note that you cannot rotate the output to be printed in landscape mode; this is a PostScript restriction.

3. Running Translated Images

This chapter discusses the following topics:

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

Other run-time topics are discussed in Chapter 4:

Topic	See
Capturing run-time statistics on a translated image	Section 4.2.2
Using run-time feedback in hand-edited information (.HIF) files to improve image translation	Section 4.2.3

3.1 Running the Translated Image

To run a translated image, use the DCL RUN command. For example, to run the sample program SIEVE_TV.EXE (translated in Chapter 1), enter the following command:

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

The Translated Image Environment (TIE) (see Section 1.2.4) issues error messages whenever it encounters errors while the translated image is running. For message descriptions, use the OpenVMS Alpha online Help Message utility or refer to the OpenVMS system messages documentation.

If the logical name TIE$DISPLAY_STATISTICS is defined to be any string other than 0 or FALSE, the TIE displays statistics about translated image execution when a program exits. See Section 4.2.2 for details.

3.2 Handling References to a Translated Image

Depending on how a translated image is activated, you may need to reflect the name change from image.EXE to image_TV.EXE:

If you are using the RUN command, specify the translated image name, as shown in Section 3.1.

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.

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 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_TV YOUR$DISK:[YOUR_DIR]MYMATH_TV.EXE
```

Note

When you run a translated image linked against a Compaq supplied shareable image in SYS$SHARE, the OpenVMS Alpha system automatically activates the correct image-you don't need to redefine the shareable image's logical name. For translated shareable images in SYS$SHARE not supplied by Compaq, you must explicitly define the appropriate logical name.

Part II Developer's Guide to Translating Images

The chapters in Part II contain the following information:

Topic	See
Improving translated image performance; identifying source code problems 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. Enhancing Performance and Analyzing Images

This chapter discusses the following topics:

Topic	See
Using the /AUDIT qualifier to learn about image characteristics that affect translating and rebuilding an application	Section 4.1
Affecting a translated image's performance by selecting specific VEST qualifiers, by studying TIE run-time statistics, and by using run-time feedback	Section 4.2
Creating and processing VEST flowgraphs	Section 4.3
Identifying unaligned and skewed data	Section 4.4

4.1 Using the /AUDIT Qualifier

The /AUDIT qualifier instructs VEST to analyze an image and to provide a brief summary assessment that may help you decide on a migration strategy for your application. The summary, which is based on error messages issued during the VEST analysis and on information in the image's debugger symbol table (DST), answers the following questions:

> Can the image be recompiled and rebuilt on an OpenVMS Alpha system if the sources are available? Yes or No.
>
> Can the image be translated? Yes or No.
>
> Does the image include code that would slow its performance as a translated image? Slow or OK
>
> What source languages were used?

If, for example, auditing determines that an image includes code that is unsupported on the OpenVMS Alpha operating system, you can modify the sources (if available) to eliminate the unsupported code. Or if auditing flags a potential performance problem, you can either correct the source or translate the image with a VEST qualifier that will minimize its effect on performance.

See Appendix A for a detailed description of the /AUDIT qualifier. The description includes a suggestion for building a file of summary descriptions by issuing a series of VEST/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.

4.2 Considering Performance

You can use VEST both to enhance the performance of translated images as well as to ensure exact VAX behaviors. This section discusses performance-related VEST qualifiers, TIE run-time statistics, and run-time feedback you can use to improve translated image performance.

4.2.1 Using Performance-related Qualifiers

VEST includes several qualifiers that influence, directly or indirectly, the performance of a translated image. Table 4-1 briefly describes these qualifiers. Refer to the full qualifier descriptions in Appendix A for details.

4.2.2 Run-Time Statistics

By defining a logical name, you can request that the Translated Image Environment (TIE) display run-time statistics about translated image execution. The TIE displays these statistics whenever a program exits as long as one or more translated images were activated during program execution. For example, if a program's main image is native but calls a translated RTL, the TIE displays the run-time statistics when the program exits.

These statistics, which pertain to all images activated during program execution, describe the TIE resources used and all interactions between native and translated images. These statistics are particularly useful for characterizing translated image performance. For example, the statistics show how many VAX instructions required interpreting and how many complex VAX instructions required emulating.

To display the statistics, define the logical name TIE$DISPLAY_STATISTICS to be either 1 or TRUE or any string other than 0 or FALSE. For example:

```
    $ DEFINE TIE$DISPLAY_STATISTICS TRUE
```

The TIE writes the statistics to SYS$OUTPUT, which normally points to your terminal. However, to save the statistics, along with other program output, to a file, define the logical name SYS$ERROR. For example:

```
    $ DEFINE SYS$ERROR DEV_00::[GROUP.STATS]ERRORS.DAT
```

To turn off the statistics, redefine the logical name as 0 or FALSE. The statistics will also not be displayed if the logical name is not defined. For example:

```
    $ DEFINE TIE$DISPLAY_STATISTICS FALSE
```

or

```
    $ DEASSIGN TIE$DISPLAY_STATISTICS
```

Example 4-2 shows the statistics displayed for SIEVE_TV.EXE.

The following list explains the statistics from Example 4-2 in detail:

1 This table describes all the TIE lookups. Whenever a translated image does a CALL, a JSB, or a JMP that could not be resolved by VEST translation, the TIE must determine the target code by looking up the destination in an internal table. Depending on the results of the lookup, control passes either to VAX code that must be interpreted, to translated VAX code, or to native code. As an optimization, the TIE retains information for frequently used lookups in a lookup cache. A statement following the lookup table notes the percentage of lookups found in the cache.

2 "VAX code located outside translated images" refers to VAX code that an image creates at run time and which must be interpreted. Such code occurs very rarely.

3 A Fault-on-Execute condition converted to a lookup occurs when translated or native code attempts to branch to VAX code, which usually happens with RET, RSB, or computed branches. Depending on the lookup, the TIE either interprets the VAX code or finds the equivalent translated code.

4 The TIE may need to use the VAX-Instruction Atomicity Controller when an image was translated with the VEST qualifier /PRESERVE=INSTRUCTION_ATOMICITY. The controller ensures that translated code equivalent to a complete VAX instruction finishes without interruption should an asynchronous system trap (AST) or other event occur.

5 This list describes the type and number of complex VAX instructions emulated by the TIE.

6 This statement summarizes the total number of VAX instructions interpreted.

7 This line notes the overall CPU time used to run the program.

8 The autojacketing statistics note the number of times calls occurred between translated and native routines. The number of calls from translated to native routines is always the same as the

number of TIE lookups in the CALLx column of the "Went to Native routines:" row in the table at the top of the statistics.

9 Finally, the statistics list all the translated images and then all the native images used while the program ran.

4.2.3 TIE Feedback and .HIF Files

At run time, the TIE can save information about code it has interpreted in a translated image. When you then retranslate the image, VEST uses that information, called feedback, to create a more efficient translated image. Repeatedly running and retranslating an image therefore becomes a simple and effective way to improve run-time performance.

If the TIE interprets code at run time and thereby discovers previously untranslated entry points within the image, it can write descriptions of the entry points to a hand-edited information file or .HIF file, a text file that describes entry points in an image (Section 5.3). If a .HIF file does not already exist for the translated image, the TIE creates one. When you subsequently retranslate the image, VEST reads the image's .HIF file to locate and translate code it was unable to find before. You can repeatedly run and then retranslate an image until the TIE needs to interpret little or no VAX code.

When a program exits, the TIE issues messages to specify each .HIF file it has written to. The TIE suggests retranslation when a translated image requires interpreting more than 10000 instructions per CPU second. For example:

```
%TIE-I-HIFENTRY, There was one HIF entry appended to SYS$SCRATCH:DHRYSTONE.HIF
%TIE-I-RETRANSLATE, The TIE interpreted an average of 16965 instructions per CPU
second. Retranslate images that have HIF files to improve performance.
```

The TIE issues the second message only once per program, no matter how many translated images actually execute. The number of instructions interpreted is the combined total for the entire program.

You can control whether or not the TIE saves feedback information either for a specific image or for all translated images that run while a logical name is defined. Section 4.2.3.1 describes the /FEEDBACK qualifier, which allows you to enable feedback when you translate an image. Section 4.2.3.2 describes the logical names you can define to control TIE feedback and to override the effects of the /FEEDBACK qualifier.

4.2.3.1 VEST /FEEDBACK Qualifier

The VEST /FEEDBACK qualifier is one way to control whether or not the TIE saves .HIF information at run time. The default is /FEEDBACK, which requests that the TIE write entry point information to a .HIF file when the translated image actually executes. If you specify /NOFEEDBACK when translating an image, the TIE does not write to .HIF files at run time. The logical name TIE$FORCE_FEEDBACK allows you to override /NOFEEDBACK at run time.

4.2.3.2 Controlling TIE Feedback-Logical Names

Two logical names pertain to the processing of .HIF files:

TIE$FEEDBACK_DIRECTORY-Defining this logical name specifies a directory for .HIF files. When recording entry points, the TIE writes to a .HIF file in the specified directory. For example:

```
$ DEFINE TIE$FEEDBACK_DIRECTORY DEV_00:[GROUP.HIF]
```

For each translated image, the TIE writes to a different .HIF file within the directory. The .HIF file name matches the name of the translated image the TIE is describing. When a .HIF file already exists, the TIE appends the new entries to the end of the file. To suppress all feedback, define the logical name to the null device as follows:

```
$ DEFINE TIE$FEEDBACK_DIRECTORY NL:
```

If the logical name is not defined, the TIE uses SYS$SCRATCH as the .HIF feedback directory.

TIE$FORCE_FEEDBACK-This logical name's setting overrides the effect of the VEST /NOFEEDBACK qualifier for all translated images. If you define TIE$FORCE_FEEDBACK to either 1 or TRUE or any string other than 0 or FALSE, then the TIE writes information to an .HIF file regardless of the translation setting for all translated images. The following example enables feedback regardless of the translation setting:

```
$ DEFINE TIE$FORCE_FEEDBACK TRUE
```

The forced feedback continues until you redefine TIE$FORCE_FEEDBACK. Specifying /NOFEEDBACK at translation time normally suppresses run-time feedback-that is, the TIE will not write to .HIF files the information it derives from interpreting code. If TIE$FORCE_FEEDBACK is undefined or is defined to be either 0 or FALSE, TIE feedback depends on whether the image was translated with /FEEDBACK or /NOFEEDBACK. Either of the following commands defers to the translation setting for feedback:

```
$ DEFINE TIE$FORCE_FEEDBACK FALSE
```

or

```
$ DEFINE TIE$FEEDBACK_DIRECTORY NL:
```

4.3 Using VEST Flowgraphs

Using the /FLOWGRAPH and /VIEW qualifiers in combination, you can request a flowgraph file and determine its contents. The flowgraph file contains information used by the FLOWGRAPH command to create one or more PostScript formatted flowgraph files. The function of flowgraphs is to help you understand an image's structure and to put into context the errors an image incurs during translation. Also, you can use the /RESTRICT qualifier to confine VEST processing to specific parts of the image. The resulting flowgraph shows only those parts.

This section describes how to request and process a flowgraph. See Section 4.3.1 for a detailed description of flowgraph components.

A flowgraph can be one of the following types:

A call flowgraph that charts the image's calling structure and includes the names of the called routines.

An error flowgraph that charts the routines in the image that incurred VEST error messages.

A complete flowgraph that charts the program flow of the entire image based on the code that VEST has found.

Use the /VIEW qualifier to select which of these three kinds of flowgraphs to include in the flowgraph file and the /VIEW qualifier keywords to select the kind of information to be included within either an error or a complete flowgraph:

The input machine code

The output machine code

The program source code

Information on entry masks, stack depths, and resources used and set

The command sequence in Example 4-3 requests all three flowgraphs, processes them, and queues them to a PostScript printer. In the example, graph 1 is the call graph, graph 2 is the error graph, and graph 3 is the complete graph. Note that graph 2 includes 0 nodes, which indicates that the graph is empty; that is, the image incurred no errors.

Figure 4-1 shows the SIEVE call graph ("GRAPH 1" in the example).

See Appendix A for a complete description of the FLOWGRAPH command. Note that command qualifiers allow you to modify the size of the flowgraphs (the /SCALE_FACTOR qualifier) and to select a portion of the flowgraph (the /STARTING_ADDRESS qualifier).

4.3.1 DHRYSTONE.EXE Flowgraph

Figure 4-2 shows the complete flowgraph of an image called DHRYSTONE.EXE at 0.33 scale. This section refers to this figure to describe the various parts of the flowgraph. The header line 1 has the image name, version, and link date from the image header. Pages are numbered 2 x-y such that page 1-2 is to the right of page 1-1 and page 5-1 is below page 4-1.

In VEST flowgraphs, each subroutine is a disjoint subgraph. There are no connections shown between it and any other subroutine. The DHRYSTONE flowgraph includes 13 large rectangles that represent the main program and 12 subroutines. The rectangles are arranged on the pages in decreasing order of area, with the largest at the upper left. (The main program 3 happens to fall near the center of the page.)

4.3.2 Basic Blocks in Flowgraphs

Figure 4-3 shows the printed form of one subroutine in the DHRYSTONE flowgraph. The graph has 17 basic blocks connected by 20 arcs. A basic block is a sequence of VAX instructions that are all executed together. The blocks are arranged vertically so that, ignoring loops, each block falls below all of its predecessors. The blocks are arranged horizontally so that each block tends to be centered under its predecessors and above its successors. All forward-going arcs are drawn down and to the right. All loop-closing arcs are drawn up and to the left.

VEST flowgraphs consist of basic blocks in six different shapes.

CALLx entry block

A CALLx entry block 1 consists of a call mask. It is hexagonal and contains the name of the block, the name of the procedure if known, and the VAX call mask in hexadecimal. The name of the block consists of the VAX virtual address of the first byte of the block in hexadecimal followed by "_CALL". A CALLx entry block is reached by means of a CALLS or CALLG instruction. The main entry point of an executable image is a CALL entry called from the OpenVMS operating system.

JSB entry block

A JSB entry block is oval and contains the name of the block, and the name of the subroutine if known. The name of the block consists of the VAX virtual address of the first byte of the block in hexadecimal followed by "_JSB". A JSB entry point is reached by means of a JSB, BSBB, or BSBW instruction.

Normal block

A normal block 2 is rectangular and contains the name of the block and all the VAX instructions within it. The name of the block is simply the VAX virtual address of the first byte of the block in hexadecimal. A normal block is reached by means of a branch, case, jump, or return instruction or by falling through from a preceding basic block.

CALLx placeholder block

A CALLx placeholder 3 block is dashed hexagonal and contains the name of the block, and the name of the called procedure if known. The name of the block consists of the VAX virtual address of the first byte of the called procedure in hexadecimal (if known at translate time, else zero), followed by "_" and a unique hexadecimal number, followed by "_" and a single digit. A CALLx placeholder block represents the flow in and out of a procedure and any recorded side effects of that procedure for each call of that procedure. If Func1 is called from five different places, there will be five different placeholder blocks for those calls (plus the real call entry block for the actual code of Func1).

JSB placeholder block

A JSB placeholder block is dashed oval and contains the name of the block, and the name of the subroutine if known. The name of the block consists of the VAX virtual address of the first byte of the subroutine in hexadecimal (if known at translate time, else zero), followed by "_" and a unique hexadecimal number, followed by "_" and a single digit. A JSB placeholder block represents the flow in and out of a subroutine and any recorded side effects of that subroutine for each call (JSB) of that subroutine.

Exit node block

An exit node block is octagonal. It contains a summary of the resources set by the routine and the change in stack depth.

4.3.3 Arcs in Flowgraphs

VEST flowgraphs have two styles of arcs.

Normal arc

Normal arcs 4 are solid and used almost everywhere.

True branch from a two-way conditional arc

True arcs 5 are dashed and are used for the taken path of a two-way branch, for nonfallthrough paths from CASE instructions, and for all but the fallthrough arc of a CASEx instruction.

In Figure 4-3, the first basic block 1 shows that at address 3208 is a VAX called procedure named Func2 with call mask 003C (specifying to save and restore R2, R3, R4, and R5).

The second basic block 2 contains four VAX instructions, starting at address 320A.

This block falls into a loop consisting of the next five blocks. The block at 3218 ends with a call to Func1 3. The placeholder for Func1 shows a normal return connection to the block at 322B.

This block ends with a BNEQ instruction whose true (taken) arc goes to 3235 and whose false (fallthrough) arc goes to 322F. Similarly, the block at 3235 ends in a two-branch whose true arc 6 loops back to 3218.

The rest is straightforward, except for the call at the end of block 3253. It goes indirectly through what turns out to be an OpenVMS VAX image fixup vector, pointing to a shareable image. By parsing the fixup vector and reading an .IIF file for the shareable image VAXCRTL, VEST is able to identify that the call goes to VAXCRTL:STRCMP, and that routine 7 does a normal return. Without the .IIF file, VEST would have to make assumptions about the return point.

4.3.4 Error Flowgraphs

Figure 4-4 is an example of an error graph for a routine within DHRYSTONE.EXE. A wide, dashed pointer connects a shaded hexagonal block describing the error with the basic block in which the error was discovered. The blocks showing the path that reveals the error are displayed larger than blocks that are not in the error path. The example shown highlights a code path that would cause R3 to be read uninitialized.

4.4 Identifying Data Alignment Problems

You can use OMSVA to identify unaligned and skewed data in an OpenVMS VAX image. The qualifier /DST instructs VEST to collect data information from the debugger symbol table (DST) in the image and the command DSTGRAPH creates a PostScript output file that shows how the data is aligned. This capability is useful if you have application source files and can use the data alignment information to help you port the application to an OpenVMS Alpha system.

Alignment on a natural boundary occurs when a data item's address is a multiple of the data item's size in bytes. The mixture of byte-sized, word-sized, and larger data types typically found in data-structure definitions and static data areas in OpenVMS VAX applications can lead to data not being aligned on natural boundaries. Skewed data is not unaligned, but its layout requires more memory accesses than is strictly necessary. Skewed data's performance impact is small compared to the performance impact of unaligned data.

OpenVMS VAX systems use microcode to minimize unaligned data's impact on performance. But on OpenVMS Alpha systems, there is no hardware assistance. Instead, references to unaligned data trigger a fault, which must be handled by the OpenVMS Alpha operating system unaligned fault handler. While the fault is being handled, the instruction pipeline must be stopped. Therefore, the performance cost of an unaligned reference is dramatically higher on an OpenVMS Alpha system.

See the manuals Migrating to an OpenVMS Alpha System: Planning for Migration and Migrating to an OpenVMS Alpha System: Recompiling and Relinking Applications for further information about the importance of data alignment when migrating applications to OpenVMS Alpha.

4.4.1 Using the /DST Qualifier and the DSTGRAPH Command

To begin, you must recompile and relink your OpenVMS VAX program using the /DEBUG or /DEBUG=ALL qualifier, depending on the source language. The resulting image contains a full DST that describes the addressing for each variable. Unless you create the image by compiling and linking with the /DEBUG qualifier, the image will not contain the full DST information VEST needs. The /DST qualifier instructs VEST to format all the memory references in the DST and to write them to a file called image.STI. The extension STI stands for symbol table information.

For example:

```
    $ VEST/DST/NOEXECUTABLE/INTERPRET=ALL_CODE DHRYSTONE.EXE
```

The qualifiers /NOEXECUTABLE and /INTERPRET=ALL_CODE cause VEST to execute quickly because it does not actually translate code. VEST creates the file DHRYSTONE.STI.

The DSTGRAPH command accepts an image.STI file as input and generates a PostScript file called image.PS as output. Using command-line qualifiers, you can control which data structures will be shown and the scale used for the data illustrations. See Appendix A for a detailed description of the DSTGRAPH command and its qualifiers.

4.4.2 Interpreting DSTGRAPH Output

This section describes how to interpret the DSTGRAPH output. Section 4.4.2.1 describes the overall layout of the pictures and Section 4.4.2.2 explains the detailed syntax of the information shown for each module.

4.4.2.1 Overall Layout

The PostScript output shows all the information for each source module in a separate rectangle that has rounded corners and a heavy border. Figure 4-5 shows an example that describes a source module called AXDHRYSTONE_GLOBAL_DEF. The rectangles represent individual modules that have the following characteristics:

> A rectangle contains text and structure diagrams in one or more columns.
>
> A rectangle is never wider than one page, but may span multiple pages vertically. The text and diagrams flow down one column, possibly across many pages, and up to the top of the next column. When a column spans more than one page, the top and bottom overlap slightly with the top and bottom of the contiguous pages.
>
> At the default scale (1.0), a rectangle can include two columns of text and diagrams. At the scale of 0.8, a rectangle can include three columns.
>
> At the scale of 0.75, a page can include up to three rectangles side-by-side. Any scale below 0.75 becomes unreadable.

4.4.2.2 Syntax and Conventions

This section discusses the syntax and conventions used to present the data structure information for each source module, which includes:

A header line describing the source module

A list of the unaligned or skewed data items identified in the module

Diagrams of the identified data structures

Module Header Line

Above the rectangle representing a module is a header:

```
    module_name [language]
```

Where module_name is the name of the source module and language is the source language, which is one of the following DST-supported languages:

VAX Ada	VAX COBOL	MODULA
VAX BASIC	VAX DIBOL	VAX PASCAL
BLISS-32	VAX FORTRAN	PL/I
VAX C	KOALA	RPG
C++	VAX MACRO-32	SCAN

List of Variables

Within the rectangle, from the top, is a list of the unaligned or skewed variables identified for the module. Each line in the list has the following syntax:

```
    status: name_1\name_2\...\name_n [size] @ offset
```

Where

status	**UNaligned or SKewed**
name_1\name_2\...\name_n	A fully qualified variable name. The initial names (name_1\name_2\ ...) are routine or record names. Routines and record names are listed in alphabetical order. Individual variables are listed in numerical offset order. The final name (name_n) is the variable itself. To save space, ditto marks (") are used instead of initial names that match those of the preceding line. Names longer than 38 characters are truncated to 38 by keeping the first 32 and last 4 characters, separated by "..". Names that are all upper-case letters are converted to initial caps to fit more letters of the name into the diagrams. Refer to the list below that describes conventions for representing certain kinds of variables.
[size]	The size of the variable in bytes. The number of whole bytes is in hexadecimal and the number of bits in partial bytes is given as a fraction. For example: [10] equals 10 hexadecimal (16 decimal) bytes [2.7] equals 2 bytes plus 7 bits
@ offset	The offset of the variable from the front of the containing structure. The same notation is used to represent the offset that is used to represent variable

size (see above). Variable offsets within a record are given with respect to the front of the record, which is assumed to be aligned on a boundary determined by the /WIDTH qualifier (quadword boundary by default)

DSTGRAPH uses the following conventions to represent certain kinds of variables:

Variables at absolute addresses are described as being inside a structure with the name .Globals., starting at virtual address 0.

Variables specified in the DST as relative to (FP) are described as being inside a structure with the name .Locals.

Variables specified in the DST as relative to (SP) are described as being inside a structure with the name .Stack.

Variables specified in the DST as Bliss fields are all described as being inside a single module-global structure with the name .Blifld. This can result in somewhat jumbled-looking Bliss data structures.

Data Structure Diagrams

Within each rectangle appear one or more data structure diagrams below the list of the unaligned or skewed variables. The diagrams are in alphabetical order by variable name and each represents one of the identified variables. The following list describes the content and format of each diagram:

The fully qualified name of the entire structure appears immediately above the diagram.

Below the structure name is a series of bars that depict the individual variables within the structure. The length of each bar corresponds to a number of bytes, which is determined by the /WIDTH qualifier to the DSTGRAPH command. The default width is 8 bytes, which corresponds to aligned quadwords.

The shading within each bar indicates the status of the represented variable:

- Unaligned variables are shown with white names on a black background.
- Skewed variables are shown with black letters on a gray background.
- Normal variables are shown with black letters on a white background.

Within a structure, DSTGRAPH positions each variable, according to its order in the DST, starting at the first bar in which it will fit without creating an overlap. Bars are set off by a blank line, without repeating the structure name. If there are overlapping variables in a structure, the structure is drawn as multiple stripes, each containing no overlaps.

If a variable name is too long to fit within a bar segment proportional to its size in bits, the point size of the typeface is first reduced to 83% or 67% of normal. If the name is still too long, it is truncated on the right until it fits. (Cutting the /WIDTH in half allows twice as much room for each name.)

To the far right of each bar that contains the start of a variable is the byte offset of the first byte in hexadecimal. Bytes within a bar are numbered from right to left. At the end of each structure, the total size in decimal bytes is given. Structure sizes larger than 9999 bytes are given in Kbytes (multiples of 1024) or Mbytes (multiples of 1,048,576).

Each variable within a structure occupies from one to three bars, starting and ending at positions proportional to the offsets of the first and last bits of the variable. Were a variable to occupy more than three lines, DSTGRAPH omits the excess lines and notes the elision by putting double tick marks on the vertical part of the box for that variable.

Unused bytes in a bar are shown as three dots inside a dashed box.

5. Using Information Files

This chapter discusses the following topics:

Topic	See
Types of information files	Section 5.1
Image information files (.IIF files)	Section 5.2
Hand-edited information files (.HIF files)	Section 5.3
Information file syntax	Section 5.4

5.1 Types of Information Files

The types of information files include: image information files (.IIF files), hand-edited information files (.HIF files), and symbol information files (.SIF files). Two types, .IIF and .HIF files, are used and created differently but share the same syntax. The third type, .SIF files, which are discussed in Section 6.4, have a different syntax and are used to control how VEST orders entries in a translated image's symbol vector.

5.2 Image Information Files

An image information file is a text file that describes the interface to a shareable image by detailing its entry point properties. When translating a shareable image, VEST creates a corresponding .IIF file. VEST then reads that .IIF file whenever it translates an image that refers to (that is, has image activate fixups to) the corresponding shareable image. The .IIF file describes the properties of the shareable image's exported interface; that is, the precise locations that are described in the image's global symbol table (GST). For example, if an image refers to LIBRTL and MTHRTL, VEST searches for the image information files LIBRTL.IIF and MTHRTL.IIF. The information in these files allows VEST to generate translated VAX code that correctly handles references to the shared libraries. Note that the OpenVMS Alpha operating system includes .IIF files for all the translated run-time libraries.

See Section 2.1 for a description of the directories that VEST searches to find relevant information files. The individual descriptions in an image.IIF file describe entry properties, such as the entry type (CALL, JSB, or BRANCH) and the resources read from or written to as a result of calling the entry (uses and sets properties).

5.3 Hand-Edited Information Files

A hand-edited information file contains information about the input image that VEST is currently translating. VEST reads .HIF files to augment or override what it can discover about an image from the image itself. Information in the file points VEST to additional code, correctly specifies an entry point (as a CALL, JSB, or BRANCH) when the input GST is wrong or ambiguous, and adds or deletes entry points that the GST describes incorrectly.

A .HIF file can be created in one of two ways:

You can manually create an image.HIF file to provide VEST with information it is unable to discover itself.

The TIE creates an image.HIF file, or appends information to an existing one, if interpreting VAX code at run time enables it to identify additional entry points in the image.

If you retranslate image after the TIE has discovered VAX code, VEST uses the image.HIF file entries to find more code and thereby create a translated image that requires less interpretation. You can repeat the process every time the TIE discovers more entry points at run time.

5.4 .IIF and .HIF File Syntax

An .IIF or a .HIF file is an ASCII file consisting of single line records of three types: image records, property records, and comment records.

5.4.1 Image Records

An image record consists of an image name and identification (as they appear in an image header) and the date and time the image was linked. An image record is always indented by either spaces or tabs. For example:

```
Image "TIME", "V1.0", 12-JUN-1992 16:45:26.56
```

An .IIF file can contain multiple image records for images with different identifications ("V1.0") and different link times. The records that immediately follow an image record pertain to that image only.

5.4.2 Property Records

A property record consists of an image offset followed by a property name, optionally followed by a property value. For example:

```
+000006981  sets 2  "R0 AP FP SP PC N Z V C" 3
```

The components of a property record are as follows:

1 The image offset represents the offset in the image to which the property applies. The offset appears in one of the following formats:

```
+integer
symbolic_name
symbolic_name-integer
symbolic_name+integer
```

where integer is a hexadecimal integer value and symbolic_name is a character string defined in a previous record. The + integer form (+0000698, for example) is the most common image offset representation to be found in an information file. If the symbolic_name form is used, it must be defined in a previous record. The last two forms express plus or minus offsets from a symbolic_name and are very rarely used.

2 The property name is one of the legal property names shown in Table 5-1.

3 The property value is an optional value specific to the property. Each value is either a character string or a hexadecimal number preceded by a plus or minus sign. See Table 5-1 for a description of the property values, if any, associated with each property name. When the image name portion of the property value refers to the current image, a period is used to refer to it.

Note that all property names and values are case-sensitive and all symbol names that contain white space or special characters must be enclosed in double quotes.

5.4.3 Comment records

A comment record is any line of information preceded by a semicolon. The comment may start a new line or be included in another record. It can occur anywhere in the file. VEST ignores all comment records.

Example 5-1 is an excerpt from one of the run-time library's .IIF files.

5.4.4 Interface Properties

Table 5-1 lists all legal property names, their associated values, if any, and a brief description.

5.4.5 Specifying Resources as Property Values

The following property names accept a list of resources as a property value:

```
sets            uses

+sets           +uses
```

```
        -sets          -uses
```

The resources that can be included in a property value list are:

The following general purpose VAX registers:

R0 R1 R2 R3 R4 R5 R6 R7 R8 R9 R10 R11

The following special purpose registers:

PC SP FP AT

which represent:

PC-the program counter (R15)

SP-the stack pointer (R14)

FP-the frame pointer (R13)

AP-the argument pointer (R12)

The following VAX instructions to indicate how the call returns:

RET RSB REI

The following processor status longword fields:

N Z V C

The following codes indicating registers used to access memory:

M.pc M.sp M.abs M.unk

A value indicating that the VAX PC has to be used as the return address of a call:

IPC

Example:

```
    "R0 AP FP SP PC N Z V C"
```

For further information about the register resources, refer to the VAX Architecture Reference Manual.

Note that the list of resources must be enclosed in quotes. The resources are case sensitive and must be separated by white space.

5.4.6 Flag Bits Modified in Processor Status Longword

Some OpenVMS VAX Run-time Library Routines (RTLs) modify flag bits in the processor status longword (PSL) based on input parameters to the routine. VEST is unable to detect this behavior when it translates an image unless an .IIF file includes one of the interface properties that describe this behavior, which include:

dv_set_to

fu_set_to

iv_set_to

Most RTLs do not modify PSL flag bits. However, routines that depend on this behavior may turn off the underflow or overflow bits or deliberately generate and handle overflows. Specifying one of these property names allows VEST to use correct trap enable values in subsequent Alpha Alpha instructions. For example, LIB$INT_OVER (in LIBRTL) sets the IV bit to the value of its first parameter, p1. If this property is not used when required, executing the translated code will fail to generate an expected integer overflow trap.

5.5 Library Information Files

A library information file describes how to rename shareable image references in the translated image depending on the referenced image section id. This file aids keeping multiple versions of the same translated library and its .IIF file on the same system. A .LIF file is created manually.

5.6 .LIF File Syntax

A .LIF file is a text file containing records of two types: renaming records and comment records.

5.6.1 Renaming records

A renaming record consists of four fields separated by one or more spaces or tabs: VAX image name, VAX image global section id, match control and the translated image name. For example:

```
    VAXCRTL          04.000004      GE      VAXCRTL_V73           (1)
    VAXCRTL          04.000006      GE      VAXCRTL_SPEC          (2)
    VAXCRTL          04.000008      EQ      VAXCRTL_V74           (3)
```

The components of the renaming record are as follows:

1 "VAX image name" is the name of the image, referenced by the VAX image being translated. VEST converts shareable image references in fixup section and global section names matching this field.

2 "VAX image global section id" consists of two hexadecimal numbers separated by a dot. These are major and minor section identifiers. This field and VAX image name are the key fields used for matching. If there are any duplicate combinations of these fields, it is indeterminate, which one of such combinations is used for matching.

3 "Match control" is an operator, which indicates how the section id is matched. One of two values are accepted in this field: GE and EQ. GE indicates that the specified section id and all greater section ids are matched by this entry if they are not matched by an entry with the greater section id. EQ indicates that only the specified section id is matched by this entry. The order of the entries is not sufficient because VEST rearranges them internally while reading the .LIF file.

4 "Translated image name" determines the prefix of the image reference in the translated image and the name of the .IIF file. The image references are constructed by appending the suffix _TV to this prefix, so it must not exceed 36 characters in length. If the prefix does exceed this length, VEST will truncate it.

In the example above, the entry with the section id 04.000004 matches all references to VAXCRTL with the section id greater or equal to 04.000004. However, the next entry overrides this matching for a single section id starting (04.000006). Finally, the entry with the section id 04.000008 overrides the matching for all section ids starting with 04.000008. The following table depicts how the image references are translated according to the .LIF file consisting of these three records:

Image reference (fixup)	Global section name	Global section id	Translated image reference (fixup)	Translated global section name	Used entry
VAXCRTL	VAXCRTL_002	04.000003	VAXCRTL_TV	VAXCRTL_TV_002	none
VAXCRTL	VAXCRTL_002	04.000004	VAXCRTL_V73_TV	VAXCRTL_V73_TV_002	1
VAXCRTL	VAXCRTL_002	04.000005	VAXCRTL_V73_TV	VAXCRTL_V73_TV_002	1
VAXCRTL	VAXCRTL_002	04.000006	VAXCRTL_SPEC_TV	VAXCRTL_SPEC_TV_002	2
VAXCRTL	VAXCRTL_002	04.000007	VAXCRTL_V73_TV	VAXCRTL_V73_TV_002	1
VAXCRTL	VAXCRTL_002	04.000008	VAXCRTL_V74_TV	VAXCRTL_V74_TV_002	3
VAXCRTL	VAXCRTL_002	04.000009	VAXCRTL_V74_TV	VAXCRTL_V74_TV_002	3
VAXCRTLG	VAXCRTLG_002	04.000004	VAXCRTLG_TV	VAXCRTLG_TV_002	none

When there are no matching records for an image name and a section id, the image reference is converted in the default way. If the .LIF file is not specified on the command line, then all image references are converted in the default way. Finally, if the same image name is specified in the library information file and as a value of /JACKET qualifier (or this qualifier has no values specified), /JACKET takes precedence, and the image reference is not converted.

5.6.2 Comment records

A comment record is any line of information preceded by a semicolon. The comment may start a new line or be included in another record. It can occur anywhere in the file. VEST ignores all comment records.

6. Translating and Replacing OpenVMS VAX Shareable Images

This chapter discusses the following topics:

Topic	See
Interoperability requirements	Section 6.1
Procedures for translating and replacing OpenVMS VAX shareable images	Section 6.2
Procedures for creating a jacket image, needed in rare cases when a native shareable image cannot completely replace the original OpenVMS VAX shareable image	Section 6.3
Description of symbol information files (.SIF files), which control how entries are ordered in a translated image's symbol vector and which symbols are included in the global symbol table (GST)	Section 6.4
Description of the /JACKET qualifier, used when you create a jacket image	Section 6.5

6.1 Interoperability Requirements

The OpenVMS Alpha system allows translated and native images to interoperate by sending and receiving calls to and from one another. The system jackets calls between native and translated images; that is, it performs the necessary conversions between the VAX and the Alpha calling standards. When you create native images that call or otherwise 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 VAX image, you need to maintain compatibility with that image.

The information in this chapter builds on a discussion of interoperability in the manual Migrating to an OpenVMS Alpha System: Recompiling and Relinking Applications. That manual uses a C program called MYMATH to illustrate a VAX shareable image and a C program called MYMAIN to illustrate a main image that calls MYMATH. Using these example programs, the manual describes how to create native images that make calls to and receive calls from translated images and highlights 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 vector in a native shareable image and the transfer vector in the superseded VAX shareable image

The procedures described in this chapter use MYMATH and MYMAIN as examples of creating native and translated shareable images that interoperate and that preserve compatibility with previous versions of the images.

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 the OpenVMS Alpha system 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 properly jacket calls between translated and native images. When you specify the /NONATIVE_ONLY qualifier on the LINK command line, 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.

See Migrating to an OpenVMS Alpha System: Recompiling and Relinking Applications, the appropriate compiler documentation, and the OpenVMS Linker Utility Manual for further information about these qualifiers.

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 OpenVMS Linker Utility Manual describes how to maintain upward compatibility by using transfer vectors for OpenVMS VAX images and symbol vectors for OpenVMS Alpha images. Maintaining upward compatibility is also advisable when creating either (1) a translated shareable image, (2) a native image that replaces an OpenVMS VAX shareable image, or (3) a native image that replaces a translated shareable image.

Just as you create consistent transfer 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 Alpha systems. When you translate a shareable image, you can use a symbol information file (.SIF file) to control how VEST orders the symbol vector in the translated image. 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 disrupting the original order. You can use whatever mechanism is convenient to create a symbol vector that keeps the original order. As an example, Section 6.2.3 discusses using a program to convert information in a .SIF file to entries in a symbol vector.

If you do not ensure that the transfer and symbol vectors maintain the same entry order, you run the risk of breaking calling images. VEST 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. And if you have to create a jacket image because not all routines can be reproduced in native mode, you may have to rebuild the jacket image to accommodate the new entry order. This process is complicated and not recommended. Use the procedures shown in this chapter instead.

For further information about transfer vectors, symbol vectors, and compatibility, refer to the 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 VAX 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 OMSVA 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 VEST subdirectory of SYS$EXAMPLES. The command files must be executed in the following order:

1. First execute BUILD_MYMATH_VAX.COM on an OpenVMS VAX system. The VAX C compiler is required.

2. Then execute BUILD_MYMATH_AXP.COM on an OpenVMS Alpha system. The DEC C compiler is required.

The procedure descriptions all follow the same format:

A figure illustrates what kind of images are 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 simplified to illustrate the fundamental steps to creating interoperable images and do not consider all cases. Your application may need to consider other factors and/or include different or additional steps.

6.2.1 Building the Original OpenVMS VAX Shareable Image

This procedure for building a shareable image on an OpenVMS VAX system illustrates the use of a transfer vector.

Create and compile the transfer vector

```
$ EDIT MYVEC.MAR
  .
  .
  .
.PSECT     $CODE$, RD, NOWRT, EXE
.TRANSFER myadd
.MASK       myadd
JMP         L^myadd+2
.TRANSFER mysub
.MASK       mysub
JMP         L^mysub+2
.TRANSFER mydiv
.MASK       mydiv
JMP         L^mydiv+2
.TRANSFER mymul
.MASK       mymul
JMP         L^mymul+2
.END
$ MACRO MYVEC/OBJ=VAX_MYVEC.OBJ
```

Make a transfer vector by creating and compiling a VAX MACRO file. Refer to the OpenVMS Linker Utility Manual and VAX MACRO and Instruction Set Reference Manual for further information.

```
Create and build the OpenVMS VAX shareable image
$ CC MYMATH/OBJ=VAX_MYMATH.OBJ
$ LINK/SHAREABLE=VAX_MYMATH,SYS$INPUT:/OPTIONS
VAX_MYVEC.OBJ
VAX_MYMATH.OBJ
SYS$SHARE:VAXCRTL/SHAREABLE
GSMATCH=LEQUAL,2,0
[EXIT]
```

Compile MYMATH, name the object file VAX_MYMATH, and link it with the transfer vector object file (MYVEC.OBJ) to create the shareable image.

Create the OpenVMS VAX main image

```
$ CC MYMAIN/OBJ=VAX_MYMAIN.OBJ
$ LINK VAX_MYMAIN.OBJ,SYS$INPUT/OPTIONS
VAX_MYMATH/SHAREABLE
SYS$SHARE:VAXCRTL/SHAREABLE
[EXIT]
```

Compile MYMAIN, name the object file VAX_MYMAIN, and link it to the shareable image VAX_MYMATH.

Define logical name and run main image

```
$ DEFINE VAX_MYMATH YOUR$DISK:[YOUR_DIR]VAX_MYMATH.EXE
$ RUN/NODEBUG VAX_MYMAIN
```

Define the logical name VAX_MYMATH so that it points to the location of VAX_MYMATH.EXE.

6.2.2 Creating the Translated Shareable Image

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

Creating a translated shareable image and a translated main image that calls it (Section 6.2.2.1)

Creating a translated image and a native main image that calls it (Section 6.2.2.2)

For a replacement shareable image, using a .SIF file is recommended to control how VEST orders the entries in the translated image's symbol vector. Refer to Section 6.4 for a detailed description of .SIF files.

6.2.2.1 Translated Main Image

This procedure illustrates translating both the shareable and the main images.

Translate main and shareable images

```
$ VEST VAX_MYMATH
$ VEST VAX_MYMAIN
```

Translate the shareable image VAX_MYMATH.EXE. VEST automatically produces the file VAX_MYMATH.IIF. When VEST translates the calling image VAX_MYMAIN, it uses the .IIF file to correctly translate references to VAX_MYMATH_TV.EXE.

Define logical name and run main image

```
$ DEFINE VAX_MYMATH_TV YOUR$DISK:[YOUR_DIR]VAX_MYMATH_TV.EXE
$ RUN VAX_MYMAIN_TV
```

Define the logical name VAX_MYMATH_TV so that it points to the location of VAX_MYMATH_TV.EXE. When you run VAX_MYMAIN_TV, it successfully calls VAX_MYMATH_TV.

6.2.2.2 Native Main Image

This procedure illustrates translating the OpenVMS VAX shareable image and creating a native main image that calls it.

Translate OpenVMS VAX shareable image

```
$ VEST VAX_MYMATH
```

Create native main image

```
$ CC/TIE MYMAIN/OBJ=AXP_MYMAIN
$ LINK/NONATIVE_ONLY AXP_MYMAIN, SYS$INPUT:/OPTIONS
VAX_MYMATH_TV.EXE/SHAREABLE
[EXIT}
```

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

Define logical name and run main image

```
$ DEFINE VAX_MYMATH_TV YOUR$DISK:[YOUR_DIR]VAX_MYMATH_TV.EXE
$ RUN AXP_MYMAIN
```

Define the logical name MYMATH_TV so that it points to the location of MYMATH_TV.EXE. When you run MYMAIN, it successfully calls MYMATH_TV.

6.2.3 Building a Replacement Shareable Image

The following procedure describes a process for creating a native shareable image that replaces an OpenVMS VAX image. It also demonstrates building a native main image and a translated main image. In an OpenVMS Alpha system, both native and translated images can call a native replacement image.

Create .SIF and .IIF files

```
    $ VEST VAX_MYMATH.EXE/SIF/NOEXECUTABLE
    $ DIRECTORY VAX_MYMATH.*IF
```
Directory YOUR$DISK:[YOUR_DIR]

```
    VAX_MYMATH.IIF;1
    VAX_MYMATH.SIF;1
```
The first step when creating a native replacement image is to consider compatibility, either with the original OpenVMS VAX shareable image or with a translated version of the image. In the latter case, a .SIF file may be available that specifies the correct order for symbol vector entries. If not, you can retranslate the shareable image with the /SIF qualifier to create one.

It may be convenient to use some kind of automated procedure to create a linker options file that sets up the symbol vector entries correctly. The OMSVA kit includes an example C program called SIF2OPT that reads a .SIF file and creates the corresponding SYMBOL_VECTOR= entries in a linker options file. Note that in some cases, you may need to leave the first SYMBOL_VECTOR= entry SPARE for compatibility with the translated version of the shareable image.

You need the .IIF file created in this step for translating OpenVMS VAX main images that call the native replacement image. The entries in the .IIF file must correspond to the order of the entries in the native image symbol vector. Translating the OpenVMS VAX shareable image using a .SIF file that reflects the symbol vector order in the native replacement image guarantees creating an appropriate .IIF file.

Build native shareable image

```
    $ CC/TIE MYMATH/OBJ=AXP_MYMATH
    $ LINK/SHAREABLE/NONATIVE_ONLY AXP_MYMATH, SYS$INPUT:/OPTIONS
    SYMBOL_VECTOR=(SPARE_PROCEDURE,-
                myadd=PROCEDURE,-
                mysub=PROCEDURE,-
                SPARE_PROCEDURE,-
                mymul=PROCEDURE)
    GSMATCH=LEQUAL,2,0
    [EXIT]
```
Compile the shareable image with the /TIE qualifier and link it with the /NONATIVE_ONLY qualifier. Include a linker options file that orders the entries according to the order in the original OpenVMS VAX shareable image and that leaves usv offset 0 spare.

Build native main image

```
    $ CC/TIE MYMAIN/OBJ=AXP_MYMAIN
    $ LINK/NONATIVE_ONLY AXP_MYMAIN,SYS$INPUT:/OPTIONS
    AXP_MYMATH/SHAREABLE
    [EXIT]
```
Compile and link the main image.

Define logical name and run native main image

```
    $ DEFINE AXP_MYMATH YOUR$DISK:[YOUR_DIR]AXP_MYMATH.EXE
    $ RUN/NODEBUG AXP_MYMAIN
```
Define the shareable image logical name so that it points to the correct location and run the native main image.

Translate and run OpenVMS VAX main image

```
    $ VEST VAX_MYMAIN
    $ DEFINE VAX_MYMATH_TV YOUR$DISK:[YOUR_DIR]AXP_MYMATH.EXE
    $ RUN/NODEBUG VAX_MYMAIN_TV
```
Translate the OpenVMS VAX main images using an .IIF file that reflects the native replacement image's symbol vector. Define the logical name (in this case, VAX_MYMAIN_TV) to the new native shareable image and run the translated image. Note that to enable the translated image to call the native replacement image, you must define the translated image name to refer to the native image name, as shown in this example.

Note that you must ensure that the idents of VAX_MYMATH_TV and AXP_MYMATH allow image activation.

6.3 Procedures for Building a Jacket Image

A jacket image is a specially constructed translated version of an OpenVMS shareable image. The jacket image performs tasks like the following:

- Implements one or more translated routines that, for one reason or another, must be available in translated form
- Redirects translated calls to the appropriate native routines and returns any results to the calling image
- Converts between calling standards when nonstandard calls are made or when a JSB branch is used to go between images

Figure 6-1 illustrates the interconnection of the jacket image, the native image that performs most routines, a translated main image, and a native main image.

To show how a jacket image works and how it relates to other images, the procedures that follow assume that the routine mydiv cannot be replicated in the native version of MYMATH. Some of the real reasons for creating a jacket image include:

- The original shareable image includes a branchentry or jsb entry. The Alpha AXP call standard supports call entries only.
- A routine provided on OpenVMS VAX is not available on OpenVMS Alpha so the jacket image includes a routine signaling an error when an image calls the obsolete routine.
- Two shareable images, a translated shareable image and a native shareable image serving a similar purpose, share system resources that need to be coordinated by a single image.
- An OpenVMS VAX image does not use the defined OpenVMS VAX calling standard.

Remember that a jacket image is not required in most cases. Do not create one unless it is really necessary.

The OMSVA kit includes the source and command files for building the jacket image and native replacement image illustrated in this section. After installing the kit, the source and command files are located in the VEST subdirectory of SYS$EXAMPLES. The command files must be executed in the following order:

1. First execute BUILD_MYMATH_JACKET_VAX.COM on an OpenVMS VAX system. The VAX C compiler is required.
2. Then execute BUILD_MYMATH_JACKET_AXP.COM on an OpenVMS Alpha system. The DEC C compiler is required.

The procedures for building a jacket image are divided into the following sections:

- Preparing the jacket image sources (Section 6.3.1)
- Preparing the companion native shareable image (Section 6.3.2)
- Building the jacket image (Section 6.3.3)

6.3.1 Preparing the Jacket Images Sources

The jacket image you build on an OpenVMS VAX system serves no useful purpose until it is translated. The image you construct will only run in translated form on an OpenVMS Alpha system and will provide an interface to the native replacement image. When you compile and link the jacket image sources, you are doing so as if you were compiling and linking on OpenVMS Alpha. Why? Because the jacket image, before you translate it, must already include correct references to the routines that will be provided by the native replacement image.

Fooling the jacket image into including the correct references is the job of the stub image. The stub image consists solely of a transfer vector with entries that match the corresponding entries in the native replacement image. The stub's transfer vector omits entries for routines that the jacket image itself will perform, but reserves 16 bytes within the vector to preserve correct spacing. (Each entry in an OpenVMS Alpha symbol vector is 16 bytes long.) When you build the actual native replacement image, you ensure that its symbol vector exactly matches the entries and spacing given in the stub's transfer vector. Eventually, you translate the jacket image, using the VEST qualifier /JACKET provided for this special case. At run time, the jacket image can forward calls to the appropriate native routine at the correct address, which was established when you linked with the stub image.

The procedure for preparing the jacket image sources include the following steps:

Create a jacket image transfer vector

Create a jacket image source file

Prepare a stub transfer vector

Compile and link the stub

Compile the jacket image source module and link it with the stub image

Create a jacket image transfer vector

```
$ EDIT MYVEC_JACKET.MAR
 .
 .
.PSECT $CODE$, RD, NOWRT, EXE
.MASK myadd_jkt
JMP L^myadd_jkt+2
.MASK mysub_jkt
JMP L^mysub_jkt+2
.TRANSFER mydiv
.MASK mydiv
JMP L^mydiv+2
.MASK mymul_jkt
JMP L^mymul_jkt+2
.END
```

Create a transfer vector called MYVEC_JACKET.MAR. Include a .TRANSFER directive only for routines that the jacket image itself will perform. Omit the .TRANSFER directive for all the routines that the native replacement image will perform. These are the routines that will be jacketed.

Native images use the exported symbols; translated images call the entries in the jacket image as described in the .IIF file. By omitting the .TRANSFER directive, you ensure that the universal symbol for the routine will not be included in the jacket image's global symbol table (GST). (You can also use a .SIF file to direct VEST not to include a symbol in the GST; see Section 6.4) This is necessary to prevent a name conflict , which happens if the same universal symbol appears in different GSTs. When an image includes a universal symbol in its GST, it exports that symbol. Either the jacket image or the native replacement image must export each symbol, but not both. Also, each symbol must be exported by the image that actually performs the routine the symbol references.

Create a jacket image source file

```
$ EDIT MYJACKET.C
 .
 .
int myadd_jkt( value_1, value_2)
{
 return myadd( value_1, value_2);
}
int mysub_jkt( value_1, value_2)
{
 return mysub( value_1, value_2);
}
int mydiv( value_1, value_2 )
```

```
     int value_1;
     int value_2;
    {
     int result;
     result = value_1 / value_2;
     return( result );
    }
    int mymul_jkt( value_1, value_2, )
    {
     return mymul( value_1, value_2);
    }
    }
```

From the original source code, remove the routines that will be jacketed, that is, that the native replacement image will perform. In that code's stead, insert jacket routines that either pass control to the corresponding routine in the native image or perform some function required by your particular application. Leave in place code for any routine that must remain translated and that the jacket image itself will perform.

Prepare the stub VAX MACRO source file

```
    $ EDIT MYMATH_STUB.MAR
     .
     .
     .
    .MACRO looks_like name
    .TRANSFER          name
    .ENTRY name, ^M<R2,R3,R4,R5,R6,R7,R8,R9,R10,R11>
    MOVL       #1, R0
    MOVL       #0, R1
    RET
    .BLKB 7            ; Fill this stub out to be 16 bytes so
               ; it's the same size as a symbol vector
               ; entry.
    .ENDM
    .MACRO dont_need  name
    .BLKB 8
    .BLKB 8
    .ENDM
    .PSECT $CODE$, RD, NOWRT, EXE
    dont_need  spare
    looks_like myadd
    looks_like mysub
    dont_need  mydiv
    looks_like mymul
    .END
```

The stub source is a VAX MACRO file consisting of a transfer vector with entries that exactly match the corresponding symbol vector entries in the native symbol vector. In this example, a file called MYMATH.MAR contains two macros, looks_like and dont_need:

looks_like sets up an entry for a routine to be jacketed. Because it includes a .TRANSFER directive, the stub image, and, eventually the native image, will export the symbol for the routine.

dont_need sets up an entry for a routine remaining translated. It reserves 16 bytes in the transfer vector to preserve the ordering of the routines in the corresponding native image symbol vector.

The subsequent code in MYMATH.MAR calls either looks_like or dont_need for each routine.

Compile and link the stub

```
    $ MACRO MYMATH_STUB
    $ LINK/SHAREABLE=AXP_MYMATH SYS$INPUT:/OPTIONS
    MYMATH_STUB.OBJ
    NAME=AXP_MYMATH
    GSMATCH=LEQUAL,2,0
    [EXIT]
```

This example uses the NAME= option to name the stub

AXP_MYMATH, the same name as the native replacement shareable image.

Compile the jacket image source module and link it with the stub image

```
    $ MACRO MYVEC_JACKET
    $ CC MYJACKET
```

```
$ LINK/SHAREABLE=MYJACKET SYS$INPUT:/OPTIONS
NAME=VAX_MYMATH
MYVEC_JACKET.OBJ
MYJACKET.OBJ
AXP_MYMATH/SHAREABLE
SYS$SHARE:VAXCRTL/SHAREABLE
GSMATCH=LEQUAL,2,0
[EXIT]
```

Compile the jacket's transfer vector and the jacket source itself. Then link the jacket image transfer vector (MYVEC_JACKET.OBJ), the jacket image object file (MYJACKET.OBJ), the stub image object file, (MYMATH /SHAREABLE), and the VAX C Run-Time Library (SYS$SHARE:VAXCRTL). The example uses the NAME= option to name the jacket image VAX_MYMATH, the same name as the translated OpenVMS VAX shareable image.

Compile and link the stub and the jacket image on an OpenVMS VAX system. At this point in the process, you need to work on an OpenVMS Alpha system to translate, compile, link, and run the various components that go in to setting up the jacket image, the native replacement image, and the main images that call them. The next step is to prepare the native shareable image sources.

6.3.2 Preparing the Native Shareable Image

As you prepare the native shareable image that will perform the jacketed routines, you need to ensure that its symbol vector corresponds to the transfer vector created in the stub image. The procedure described in this section takes this need into account and includes the following steps:

Generate .SIF and .IIF files

Use the .SIF file to control the symbol vector order

Rename or copy the .IIF file to a .HIF file

Determine the symbol vector ordering and build the native image

Generate .SIF and .IIF files

```
$ VEST VAX_MYMATH.EXE/SIF/NOEXECUTABLE
$ DIRECTORY VAX_MYMATH.*IF
```

Directory YOUR$DISK:[YOUR_DIR]

VAX_MYMATH.IIF;1

VAX_MYMATH.SIF;1

Translate the original OpenVMS VAX image to generate a .SIF file and an .IIF file. Use the .SIF file as described in the next step to control the symbol vector order in both the native replacement image and the jacket image.

Use the .SIF file to control the symbol vector order

The .SIF file enables you to guarantee the symbol vector order:

Either use VEST to create the .SIF file or create one yourself. Then edit the file so that it exactly matches the transfer vector created for the stub image. If necessary, ensure that location 0 in the vector is SPARE for compatibility with the translated jacket image.

Use the .SIF file as a basis for creating the SYMBOL_VECTOR values in the linker options file for the native replacement image. An option is to create an automated process that sets up the symbol vector based on the .SIF file. The C program SIF2OPT from the OMSVA kit, which the installation procedure copies to the SYS$EXAMPLES directory, is an example of such a procedure-it reads a .SIF file and creates a corresponding symbol vector in a linker options file.

Save the .SIF file and use it as input when translating the jacket image (Section 6.3.3). When VEST creates the translated jacket image, it sets up the symbol vector according to the entries in the .SIF file.

Build the native image

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

Compile and link the native replacement image.

6.3.3 Translating and Using the Jacket Image

The final steps to creating the jacket image are as follows:

Translate the jacket images

Translate a main image

Define a logical name and run the translated main image

Create a native main image

Define a logical name and run the native main image

Translate the stub and jacket images

```
$ VEST AXP_MYMATH
$ VEST/JACKET=AXP_MYMATH MYJACKET
$ RENAME MYJACKET.IIF VAX_MYMATH.IIF
```

Translate the stub image. Then translate the jacket image with the /JACKET qualifier specifying the name of the native replacement image. Make sure that the .SIF file created in Section 6.3.2 is in one of the VEST include directories. Rename the jacket image .IIF file (MYJACKET.IIF) so that it appears to be the .IIF file of the original shareable image (VAX_MYMATH.IIF). VEST then uses this .IIF file when translating main images that depend on the jacket image.

Translate a main image

```
$ VEST VAX_MYMAIN
```

Translate a main image, making sure to use the .IIF file created and renamed in the previous step.

Define logical name and run translated main image

```
$ DEFINE AXP_MYMATH YOUR$DISK:[YOUR_DIR]AXP_MYMATH.EXE
$ DEFINE VAX_MYMATH_TV YOUR$DISK:[YOUR_DIR]MYJACKET_TV.EXE
$ RUN/NODEBUG VAX_MYMAIN_TV
```

Define a logical name to assign the name of the original translated shareable image to the location and name of the jacket image. Then run the main image.

Create a native main image

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

Compile and link a native main image. In most cases, the native replacement image implements all routines that a native image would call and that were provided by the original shareable image. However, it is possible that a native image may need to call the jacket image for some routine not available in native form. In this case, you must also link the native main image with the jacket image, as follows:

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

Define logical name and run native main image

```
$ DEFINE AXP_MYMATH YOUR$DISK:[YOUR_DIR]AXP_MYMATH.EXE
$ RUN MYMAIN_TV
```

Define the logical name MYMATH so that it points to the location and name of the native replacement image. If you also link the native main image to the jacket image, you need to define another logical name as follows:

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

6.4 Symbol Information File (.SIF file)

The .SIF file allows you to control the ordering of entries in the symbol vector of the translated image. Using a .SIF file, you can:

Create a translated image with its symbol vector ordered identically to the transfer vector of the original image.

Create a translated image with its symbol vector ordered identically to the symbol vector of a native version of the image.

Create a translated shareable image with the symbol vector ordered identically to a previously translated version of the image.

When the symbol vector ordering is consistent from version to version, replacing one version with another is transparent to the images that call it. When the symbol vector ordering is not consistent, images that call it may no longer work.

When creating a jacket image, use a .SIF file to control which symbols are exported in the GST, as well as to ensure the symbol vector ordering. (Section 6.3.1 describes another way to control which symbols are exported - by omitting a .TRANSFER directive for an entry in the jacket stub transfer vector.) Because a jacket image and its companion native replacement image coexist on an OpenVMS Alpha system, a name conflict occurs if the GSTs in both images expect the same symbol names and you link against both images. To avoid the name conflict, you must ensure that only one GST exports a given symbol name.

Section 6.4.1 defines the .SIF file syntax and Section 6.4.2 explains how to instruct VEST to suppress entering a symbol name in the GST.

6.4.1 .SIF File Syntax

Each line of text in a .SIF file is a directive that provides VEST with information on how to deal with a specific symbol in the GST and symbol vector of a shareable image being translated. (This contrasts with the information in an .IIF or .HIF file, which attaches properties to image offsets.) The following example is the directive for a symbol named FOO:

sym_name	sym_value	sv_flag	gst_flag	usv	sym_type	sym_flags
FOO	000AB123	+S	+G	00000200	00	0E

Each directive is a single line. Spaces or tabs separate the individual fields, which are defined in Table 6-1.

The following example is a directive for the symbol STRCMP:

```
STRCMP      00000284      +S      +G      000004F0       00      4E
```

The directive states that the STRCMP symbol's offset within the OpenVMS VAX image is 00000284 and that VEST should add the symbol to the symbol vector (+S) and to the GST (+G). The following description of STRCMP, which reflects the directive shown above, is extracted from output generated by the DCL command ANALYZE/IMAGE after the image was translated using a .SIF file:

```
Universal Symbol Specification (EGSD$C_SYMG)
```

```
data type: DSC$K_DTYPE_Z (0) 1
symbol flags:2
    (0) EGSY$V_WEAK       0
    (1) EGSY$V_DEF        1
    (2) EGSY$V_UNI        1
    (3) EGSY$V_REL        1
    (4) EGSY$V_COMM       0
    (5) EGSY$V_VECEP       0
    (6) EGSY$V_NORM       1
psect: 0
value: 1264 (%X'000004F0')3
symbol vector entry (procedure)
    %X'00000000 00030B8C'
    %X'00000000 00030B8C'
symbol: "STRCMP" 4
1 Corresponds to sym_type , the symbol type
2 Corresponds to the low 7 bits of the hex value of sym_flags
3 Corresponds to usv, the symbol vector offset within the translated image
4 Corresponds to sym_name
```

6.4.2 Creating and Using a .SIF File

Creating a .SIF file is part of the process of creating a translated shareable image or a jacket image to ensure consistent ordering of symbol vectors from one version of the image to another. You can either generate a .SIF file yourself or use the VEST /SIF qualifier. The following VEST command builds a .SIF file:

```
VEST/SIF image .EXE
```

VEST writes the file, called image.SIF to your current directory. The defaults that VEST uses as it writes each directive are as follows for a shareable or jacket image:

Entry	Shareable Image	Jacket Image
CALL entry	+S +G	+S -G
JSB entry	-S -G	-S -G
Relocatable data	+S +G	+S -G

After you have edited the image.SIF file and are ready to use it for a translation, copy it to one of the directories that VEST searches for information files (see Section 2.1). If VEST finds the file, it follows its directives for creating the GST and symbol vector in image_TV.EXE. Note that you use the /SIF qualifier to create a .SIF file and not to read a .SIF file.

For Any Shareable Image -If the symbol vector of a translated shareable image must conform to the symbol vector of a native version, first create a .SIF file and then edit each directive as necessary. For example, edit the usv field of a directive so that it reflects the symbol's position in the native image's symbol vector. Retranslate the original OpenVMS VAX image with the edited .SIF file. If you edited the .SIF file correctly, the ordering of the symbol vectors within both the translated and the native shareable images will correspond exactly.

Alternatively, you can modify the contents of the SYMBOL_VECTOR directive in the options file for the native image to match the entries in a .SIF file and then relink the native image. The C program called SIF2OPT, included in the VEST subdirectory of SYS$EXAMPLES, is a procedure that reads a .SIF file and creates corresponding SYMBOL_VECTOR= entries for a linker options file.

For a Jacket Image -When you know which symbols to export in a translated jacket image, that is, to include in the GST, you can edit the .SIF accordingly. (If you have not already edited it to reflect the symbol vector ordering in the native image, you need to do that first.) Edit the directives as necessary to tell VEST, which symbols to include in (+G) or delete from (-G) the GST. Translate the specially prepared jacket image with the edited .SIF file. If you edited the .SIF file correctly, the GST in the translated jacket image will only export symbols not exported by the native replacement image.

6.5 /JACKET Qualifier

The /JACKET qualifier instructs VEST not to change references to all or some images used by the image being translated. Normally VEST looks up any references to an external image in that image's .IIF file (see Section 5.2). It then replaces the references with the corresponding correct references for the native or translated version of the image, as specified by the usv_offset property in the .IIF file.

The /JACKET qualifier allows you to specify one or more image names or none at all:

> /JACKET-Do not change any references to any shareable images.
>
> /JACKET= image-Do not change references to image, but handle references to any other images in the normal manner.
>
> /JACKET=(image1,image2...)-Do not change references to any of the list of images, but handle references to other images in the normal manner.

For example:

```
    $ VEST/JACKET=(SHARE1,SHARE2) SHARE_JACKET.EXE
```

The special effects of the /JACKET switch are as follows:

> The offsets associated with the fixup entries in the input image are used as corresponding symbol vector offsets in fixup entries in the translated image.
>
> The shareable image list in the fixup section of the translated jacket image still refers to the original shareable image names. In other words, if SHARE_JACKET.EXE refers to LIBRTL.EXE, SHARE_JACKET_TV.EXE still refers to LIBRTL.EXE, not LIBRTL_TV.EXE, which otherwise is the default.

Part III Reference Information

The appendices in Part III contain the following information:

Topic	See
A detailed description of the DSTGRAPH, FLOWGRAPH, VEST, and VEST/DEPENDENCY command lines and qualifiers	Appendix A
An alphabetical listing of all VEST, DSTGRAPH, and FLOWGRAPH messages with explanations and recommended user actions	Appendix B
Debugging problems with translations	Appendix C
Coding practices and other restrictions that affect the translatability of OpenVMS VAX images	Appendix D
An alphabetical listing of all VAX instructions with explanations for how VEST and TIE handle each one	Appendix E

A. Command Summaries

VEST

The VEST utility translates OpenVMS VAX executable and shareable images into functionally equivalent OpenVMS Alpha images. VEST also allows you to analyze OpenVMS VAX images to assess both their translatability and their performance as translated images on an OpenVMS Alpha system.

Format

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

Qualifiers	Defaults
/AUDIT	/NOAUDIT
/DEBUG	Defaults to the state of the input image
/DEPENDENCY	See the command description of VEST/DEPENDENCY
/DST	/NODST
/EXECUTABLE	/EXECUTABLE
/FEEDBACK	/FEEDBACK
/FLOAT	/FLOAT=D53_FLOAT
/FLOWGRAPH	/NOFLOWGRAPH
/IIF	/IIF (ignored if image is not shareable)
/INCLUDE_DIRECTORY	/NOINCLUDE_DIRECTORY
/INTERPRET	/INTERPRET=NO_CODE
/JACKET	/NOJACKET
/LIST	/LIST
/LIF	/NOLIF
/OPTIMIZE	/OPTIMIZE=(ALIGNMENT,SCAN,SCHEDULE)
/PRESERVE	/PRESERVE=NONE
/RESTRICT	/NORESTRICT
/SHOW	/SHOW=MESSAGES
/SIF	/NOSIF
/TRACEBACK	Defaults to the state of the input image
/VIEW	/VIEW=(ERROR=SOURCE_CODE
/WARNINGS	/WARNINGS=NONE

/AUDIT

Instructs VEST to analyze the input image and to summarize its migration characteristics.

Default:	/NOAUDIT
Format:	/AUDIT
Qualifier Values:	None.

Description

The summary specifies:

Whether the image can be recompiled

Whether the image is translatable

Whether the performance of the translated image will be SLOW or OK

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. Refer to these other messages to understand how VEST arrived at its characterization of the image. Example A-1 is an example showing the format of the summary.

The summary for each image is three text lines, each beginning with <SUM>. The first and second lines serve as the header; the third line provides the actual summary:

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

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

Comp	YES or NO to indicate whether the image can be recompiled and rebuilt.
Tran	YES or NO to indicate whether the image is translatable.
Perf	OK or SLOW to indicate whether the image includes code that would adversely affect its performance as a translated image. SLOW indicates that VEST issued at least one performance-related message while analyzing the image.
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 may call.

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

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

When you specify the /AUDIT qualifier, VEST stops parsing code in the image after 16,000 VAX instructions. Using /AUDIT forces the following VEST qualifiers:

/NOEXECUTABLE

/NOIIF

/SHOW=NOMACHINE_CODE

Example

```
$ VEST/AUDIT DHRYSTONE
```

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

```
 VEST V1.1 built at May 7 1993 13:37:35 starting at May 17 1993 15:48:16
```

```
   with command line:
VEST/AUDIT DHRYSTONE
Image "DHRYSTONE_SHR", "V1.0", 13-DEC-1989 10:17:07.95
! Message summary by category:
!
! 2 messages in SOURCE_ANALYSIS category:
!  2       INFO NONSTDCALLU - Non-standard call uses !AZ
!
! 5 messages in VERBOSE category:
!  1       INFO    READING - Reading file !AZ
!  1       INFO   NOHIF - HIF file !AZ not found
!  1       INFO   PASS1 - Starting analysis pass 1
!  1       INFO   PASS2 - Starting analysis pass 2
!  1       INFO ENDPASS2 - Ending analysis pass 2 -- beginning code
   generation and output
<SUM> Image name                                Comp Tran Perf Languages
<SUM> --------------------------------------- ---- ---- ---- ----------------
<SUM> VST_00:[STORM.TEST]DHRYSTONE.EXE; YES YES OK C
```

/DEBUG

Controls whether the translated image invokes the OpenVMS Debugger.

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: /DEBUG

Qualifier Values: None.

Description

Depending on the compiler and linker options used, an image can contain three levels of debugging related information:

Level 1	No DST	The image was linked or compiled with the /NOTRACEBACK qualifier.
Level 2	Traceback DST	The image was linked and compiled with the /TRACEBACK qualifier, which is the default
Level 3	Full DST	The image was compiled and linked with both the /DEBUG and /TRACEBACK qualifiers. (A compiler may require /DEBUG=ALL; see the relevant compiler documentation.)

By using the /[NO]DEBUG and /[NO]TRACEBACK qualifiers, you can change the level as follows:

Use /NODEBUG to create a translated image that does not invoke the debugger.

Use the /DEBUG qualifier to create a translated image that invokes the debugger. /DEBUG forces /TRACEBACK. A debug version of the translated image is useful if you are developing a translated image by modifying and recompiling source code on an OpenVMS VAX system, translating the image, and then debugging it on an OpenVMS Alpha system. Debugging a translated image is different from debugging an OpenVMS VAX image; a translated image does not support symbolic debugging, either by symbol name or source line.

See the description of the /TRACEBACK qualifier for further information.

Example

```
$ VEST/NODEBUG SCALES
```

Whether or not the SCALES.EXE image was linked with the /DEBUG qualifier, the translated image SCALES_TV.EXE will not start up the debugger at run time.

```
$ VEST/DEBUG/TRACEBACK STORM_TRACK
```

Whether or not STORM_TRACK.EXE was linked with the /DEBUG qualifier, the translated image STORM_TRACK_TV.EXE will start up the debugger at run time. Note that you must also include the /TRACEBACK qualifier in the VEST command line if the OpenVMS VAX image was linked with the /NOTRACEBACK qualifier.

/DST

Instructs VEST to collect information about data in the input image and to write that information to a file for subsequent processing by the DSTGRAPH command.

Default:	/NODST
Format:	/DST

Description

Use the /DST qualifier together with the DSTGRAPH command to identify any unaligned or skewed data present in the input image. Unaligned data is data that is not aligned on a natural boundary. In other words, the data's address is not a multiple of the data item's size in bytes. Unaligned data adversely affects the translated image's performance on OpenVMS Alpha systems. Unlike OpenVMS VAX, OpenVMS Alpha systems have no hardware assistance to minimize the impact of unaligned data on performance. Skewed data is not unaligned, but its layout requires more memory accesses than are strictly necessary.

Before using the /DST qualifier, ensure that the input image contains a full debugger symbol table (DST). A full DST is present if the image was compiled and linked with the /DEBUG or /DEBUG=ALL qualifier, depending on the source language. The /DST qualifier instructs VEST to format all the memory references in the input image's DST and to write the formatted information to a file called image.STI, where STI stands for symbol table information.

After creating the image.STI file, use the DSTGRAPH command to generate a PostScript file that illustrates any unaligned or skewed data in the image.

/EXECUTABLE

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

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

Defines the file specification for the translated image.

Description

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

If you do not provide a file specification, VEST writes the translated image to the current directory and names it by appending "_TV" to the input image's file name. For example, if the name of the OpenVMS VAX image is PROGRAM.EXE, the default name of the translated image is PROGRAM_TV.EXE.

Note

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

/FEEDBACK

Controls whether or not the TIE writes information to an .HIF file about entry points discovered when interpreting code in the translated image at run time.

Default:	/FEEDBACK
Format:	/FEEDBACK
Qualifier Values:	None.

Description

When /FEEDBACK is enabled, the TIE writes information to an .HIF file about the entry points it finds when interpreting code. If you specify /NOFEEDBACK when translating an image, the TIE does not write to an .HIF file at run time. The logical name TIE$FORCE_FEEDBACK allows you to override /NOFEEDBACK at run time.

/FLOAT

Specifies the arithmetic precision for D-floating operations in the translated image.

Default:	/FLOAT=D53_FLOAT
Format:	/FLOAT =float-type
Qualifier Values:	float-type

Specifies the precision, which can be one of the following:

D53_FLOAT	Implement 53-bit precision D-floating arithmetic. VEST converts operands to G-floating format. At run time, G-floating hardware on the OpenVMS Alpha system performs the arithmetic. Because D53_FLOAT uses hardware rather than software emulation, it has performance advantages over D56_FLOAT. It does, however, lose some precision. The three bits of precision lost are the three least significant bits in the fraction
D56_FLOAT	Implement full 56-bit precision for D-floating point operations in the translated image using software emulation. D56_FLOAT provides 3 bits more precision than D53_FLOAT, but it loses in performance.

/FLOWGRAPH

Creates a flowgraph file with contents based on selections made by the /VIEW qualifier and optionally specifies a file name.

Default:	/NOFLOWGRAPH
Format:	/FLOWGRAPH [=filespec]
Qualifier Values:	filespec

Identifies the file specification for the flowgraph file.

Description

The flowgraph file contains information used by the FLOWGRAPH command to create one or more PostScript formatted flowgraph files. A flowgraph can be one of the following types:

A call flowgraph that charts the image's calling structure and includes the names of the called routines.

An error flowgraph that charts the routines in the image that incurred VEST error messages.

A complete flowgraph that charts the program flow of the entire image based on the code that VEST has found.

Use the /VIEW qualifier to select which of these three kinds of flowgraphs to include in the flowgraph file and the /VIEW qualifier keywords to select the kind of information to be included within either an error or a complete flowgraph.

If you do not include a file specification, VEST writes the flowgraph file to the current directory and defaults to the input image's file name and the extension GRAPH. For example, if the input image is DATA_TEST.EXE, VEST names the flowgraph file DATA_TEST.GRAPH.

If you specify the /FLOWGRAPH qualifier and not /VIEW as well, VEST creates a file containing an error flowgraph, with the name of the input image and the extension GRAPH.

Use the FLOWGRAPH command to create a PostScript file from the ASCII flowgraph file. Section 4.3 describes the FLOWGRAPH command and how to print the PostScript files.

/IIF

Enables the generation of an image information file (an .IIF file) when the input image is shareable and optionally specifies a file name.

Default: /IIF if the input image is shareable. Ignored if the input image is an executable image.

Format: /IIF [=filespec]

Qualifier Values: filespec

Identifies a file specification for the .IIF file.

Description

The .IIF file describes the properties of a shareable image's exported interface , that is, the precise locations that are described in the image's global symbol table (GST). See Section 5.2 for details.

If you do not include a file specification, VEST writes the .IIF file to the current directory and defaults to the input image's file name and the extension .IIF. For example, if the input image is SHAREABLE_LIB.EXE, VEST names the .IIF file SHAREABLE_LIB.IIF.

/INCLUDE_DIRECTORY

Specifies one or more directories in which VEST searches for .IIF, .HIF, and .SIF files.

Default: /NOINCLUDE_DIRECTORY

Format: /INCLUDE_DIRECTORY =(dir-spec[,...])

Qualifier Values: dir-spec

Identifies a directory that VEST searches for .IIF, .HIF, and .SIF files.

Description

VEST searches for relevant .IIF, .HIF, and .SIF files in the following locations and in the following order:

The current default directory

The directory or directories specified as values to the /INCLUDE_DIRECTORY qualifier in the VEST command line

The directory or directories, if any, defined by the VEST$INCLUDE logical name

/INTERPRET

Controls the choice between translation or interpretation for all or specific parts of the input image.

Default: /INTERPRET=NO_CODE

Format: /INTERPRET [=keyword]

Qualifier Values: keyword

The acceptable values for keyword are as follows:

ALL_CODE	Interpret the entire input image.
NO_CODE	Do not interpret any code VEST is able to find; that is, translate all code VEST finds in the input image, including code in writeable image sections
WRITEABLE_CODE	Interpret code in writeable image sections; that is, translate all code found in non-writeable image sections

Description

VEST tries to find, parse, and translate as much VAX code as possible to minimize the need for interpreting VAX code at run time. By default, VEST translates code that it finds in a writeable image section and issues a message (RWTRANSDEF) to warn that the code could be modified at run time. If the code is modified at run time, the translated code becomes incorrect. If you know that the code in a writeable image section is not modified at run time, use the /INTERPRET=NO_CODE qualifier so that VEST does not issue a warning-level message. If you know that the code is modified, use the /INTERPRET=WRITEABLE_CODE qualifier to force the code to be interpreted at run time.

If you specify /INTERPRET without a keyword, the qualifier defaults to /INTERPRET=ALL_CODE. If you specify /NOINTERPRET, the qualifier is equivalent to /INTERPRET=NO_CODE.

VEST does not generate an .IIF file either when you specify /INTERPRET, /INTERPRET=ALL_CODE, or /INTERPRET=WRITEABLE_CODE. Under these circumstances, the .IIF file could be incorrect since VEST did not analyze untranslated code.

/JACKET

Builds a jacket image from a specially constructed OpenVMS VAX shareable image.

Default: /NOJACKET

Format: /JACKET [=(image-name[,...]])

Qualifier Values: image-name

Identifies the name of an image as specified by "NAME=" in the linker options file.

Description

The /JACKET qualifier is used only if you are creating a jacket image for a shareable image as described in Chapter 6. The /JACKET qualifier instructs VEST not to change references to all or some of the shareable images referred to by the input image. Normally VEST looks up any external references to a shareable image in that image's IIF file (see Section 5.2). It then replaces the references with the corresponding correct references for the native or translated version of the image, as specified by usv_offset property records in the IIF file.

The /JACKET qualifier allows you to specify one or more image names or none at all:

/JACKET-Do not change any references to any shareable images.

/JACKET= image-Do not change references to image, but handle references to any other images in the normal manner.

/JACKET=(image1,image2...)-Do not change references to the list of images, but handle references to any other images in the normal manner.

Example

```
$ VEST/JACKET VAX_SHARE
```

VEST creates the image VAX_SHARE_TV.EXE with no changes made to any external image references.

```
$ VEST/JACKET=(NATIVE1,NATIVE2) PARTJACKET
```

VEST creates the image PARTJACKET_TV.EXE with no change to references to the images NATIVE1.EXE and NATIVE2.EXE. All other external image references are converted as usual.

/LIF

Renames the image references according to the specified library information file.

Default: /NOLIF

Format; /LIF=filespec

Qualifier Values: filespec

Identifies a file specification for the library information file.

Description:

The /LIF command controls how the image references are renamed depending on global section ids.

/LIST

Requests a list file with contents based on selections made by the /SHOW qualifier and optionally specifies a file name.

Default:	/LIST
Format:	/LIST [=filespec]
Qualifier Values:	filespec

Identifies a file specification for the list file.

Description

The /SHOW command controls the contents of the list file.

If you do not specify a file specification, VEST writes the list file to the current directory, names it by appending "_TV 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_TV.LIS.

Example

```
$ VEST DHRYSTONE
```

This example shows the listing file DHRYSTONE_TV.LIS:

```
 VEST V1.1-25 built at Apr 19 1993 13:38:36 starting at May 17 1993 13:43:44
 with command line:
 VEST DHRYSTONE
 Image "DHRYSTONE_SHR", "V1.0", 13-DEC-1989 10:17:07.95
 ! Message summary by category:
 !
 ! 2 messages in SOURCE_ANALYSIS category:
 !  2       INFO NONSTDCALLU - Non-standard call uses !AZ
 !
 ! 5 messages in VERBOSE category:
 !  1       INFO    READING - Reading file !AZ
 !  1       INFO   NOHIF - HIF file !AZ not found
 !  1       INFO   PASS1 - Starting analysis pass 1
 !  1       INFO   PASS2 - Starting analysis pass 2
 !  1       INFO ENDPASS2 - Ending analysis pass 2 -- beginning code generation and
output
```

/OPTIMIZE

Enables and/or disables various types of performance optimizations in the translated image.

Default:	/OPTIMIZE=(ALIGNMENT,SCAN,SCHEDULE)
Format:	/OPTIMIZE =(keyword[,...])
Qualifier Values:	keyword

Specifies a type of performance optimization. The acceptable values for keyword are as follows:

ALL	Enable all the optimization choices to generate the most efficient possible Alpha AXP code in the translated image.
ALIGNMENT[= option]	Provide hints to VEST about the alignment of data in memory. These hints enable VEST to generate efficient code sequences to access data. The keyword option has one of the following values: LONGWORD Enables VEST to make optimistic alignment assumptions about data that is up to 4 bytes in size QUADWORD Enables VEST to make optimistic alignment assumptions about data that is up to 8 bytes in size. Specifying ALIGNMENT=QUADWORD may improve the performance of programs using double precision floating point arithmetic on naturally aligned data.ALIGNMENT=QUADWORD is the default. Use /OPTIMIZE=NOALIGNMENT if excessive alignment faults are degrading translated image performance.

NONE	Disable all optimizations
SCAN	Enable a linear scan (pass 2) during which VEST attempts to find more code in the input image that was not found in the initial analysis (pass 1).
SCHEDULE	Enables VEST to rearrange the generated code so that it executes more efficiently. This keyword is disabled if /PRESERVE=INSTRUCTION_ATOMICITY is specified. Specifying /OPTIMIZE=NOSCHEDULE enables reporting exceptions with accurate VAX PCs when used in conjunction with the FLOAT_EXCEPTIONS and INTEGER_EXCEPTIONS keywords with the /PRESERVE qualifier. All keyword values except ALL and NONE are negatable (NOSCHEDULE, for example).

Description

Select as many optimizations as you safely can to create a translated image with the greatest possible performance on OpenVMS Alpha systems.

If you specify /NOOPTIMIZE without a keyword, the qualifier is equivalent to /OPTIMIZE=NONE. If you specify /OPTIMIZE, the qualifier is equivalent to /OPTIMIZE=ALL.

/PRESERVE

Preserves specific aspects of VAX behaviors exactly.

Default:	/PRESERVE=NONE
Format:	/PRESERVE =(keyword[,...])
Qualifier Values:	keyword

Identifies a VAX behavior to be preserved in the translated image. The acceptable values for keyword are as follows:

ABNORMAL_RETURN_BEHAVIOR	Force VEST to interpret code when a routine modifies the return address. You may want to use this switch if the code was written in MACRO-32 and VEST has issued a WRITECF4 error message. By default, the switch is disabled and VEST assumes that the code returns normally even though it has modified the return PC. When the switch is enabled, VEST does not assume that the code returns normally, which occurs most often in PL/I code.
ALL	Enable all keywords to preserve complete VAX behavior

CONDITION_CODES	Materialize the VAX condition codes for all calls to and returns from JSB entries. Most compiler-generated code does not require this keyword. Normally, VEST analyzes the VAX code, determines when the condition codes are needed and generates code to materialize them. However, you may need to use the CONDITION_CODES keyword in some cases. For example, enabling the keyword becomes necessary when VEST has not detected a call to or a return from a routine and the VAX condition codes are used on entering the routine or immediately after returning from the routine.
FLOAT_EXCEPTIONS	Generate code to provide precise VAX floating point arithmetic exception behavior. To obtain accurate reporting of VAX PCs for such exceptions, use this keyword along with /OPTIMIZE=NOSCHEDULE.
INSTRUCTION_ATOMICITY	Generate code to provide precise VAX instruction atomicity. Enabling this keyword ensures that no partial update of VAX state will be visible to another thread of execution. Specifying this keyword forces the setting /OPTIMIZE=NOSCHEDULE. If you attempt to specify /PRESERVE=INSTRUCTION_ATOMICITY in the same command line as /OPTIMIZE=SCHEDULE, VEST issues a warning and gives precedence to the preserve option.
INTEGER_EXCEPTIONS	Generate code to provide precise VAX integer arithmetic exception behavior. To obtain accurate reporting of VAX PCs for such exceptions, use this keyword along with /OPTIMIZE=NOSCHEDULE.
MEMORY_ATOMICITY	Generate code to provide precise VAX atomic memory access. This keyword is necessary only when there is shared data in memory. Shared data can be either explicit or involuntary. Data is shared explicitly when there are at least two threads of execution sharing the same piece of data. Data is shared involuntarily when there are at least two threads of execution, each accessing data that is not explicitly shared by the threads, but is contained in the same 8-byte, naturally aligned location in memory. The MEMORY_ATOMICITY keyword ensures that updates to explicitly shared data appear to be atomic and updates to involuntarily shared data appear not to interfere with each other.
NONE	Enable none of the /PRESERVE keywords.
READ_WRITE_ORDERING	Generate code to preserve precise VAX ordering of memory accesses.
SAFETY	Generate safe code that includes no assumptions about call targets or resource usage or any other assumptions that can cause VEST to generate erroneous translated code based on faulty analysis.

SAFETY	Turns off all code optimizations. This setting generates safe code that includes no assumptions about call targets or resource usage or any other assumptions that can cause VEST to generate erroneous translated code based on faulty analysis.

All keyword values except ALL and NONE are negatable

(NOCONDITION_CODES, for example).

Description

The /PRESERVE qualifier ensures precise VAX behaviors for the selected features at the expense of performance.

If you specify /PRESERVE without a keyword, the qualifier is equivalent to /PRESERVE=ALL. If you specify /NOPRESERVE, the qualifier is equivalent to /PRESERVE=NONE. All keyword values except ALL and NONE are negatable.

/RESTRICT

Specifies code processing restrictions.

Default:	/NORESTRICT
Format:	/RESTRICT =(keyword[,...])
Qualifier Values:	keyword

Specifies the type of restriction. The acceptable values for keyword are as follows:

DEPTH= level	Restricts the call depth traced to the number of levels specified by level For example, DEPTH=3 restricts code processing to three levels of call depth.
ENTRY=(" string "[,...])	Limits the entry points processed to those that include string in the corresponding symbolic name. Each entry point address is converted to a symbolic name - this symbolic name appears on the second line within the appropriate box of an error or complete flowgraph. If a symbolic name contains string , VEST processes the corresponding entry point. VEST ignores any symbolic name that does not contain string . VEST prints the symbolic name for every entry point processed in the list file.

Note

The quotes in "string " are necessary to preserve the exact upper and lower case settings. If you omit the quotes, DCL converts all text to upper case

Description

The /RESTRICT qualifier confines VEST processing to specific entry points and specific call depths in its search for code. If you specify " string" with the ENTRY keyword, VEST enforces the following qualifier negations:

/NOEXECUTABLE

/NOIIF

/OPTIMIZE=NOSCAN

If you specify a null string with the setting /RESTRICT=(ENTRY=""), VEST selects all the entry points in the image and prints their names in the list file.

To select the routines from a specific module for analysis, use the variable string to specify the module name, that is, /RESTRICT=(ENTRY="module-name"). You can select a single routine to analyze by using both the ENTRY and DEPTH keywords; for example, /RESTRICT=(ENTRY="node_a",DEPTH=1).

You cannot use /RESTRICT to add an entry point. Use an .HIF file for this purpose.

/SHOW

Controls the information to be included in the list file.

Default:	/SHOW=MESSAGES
Format:	/SHOW =(keyword[,...])
Qualifier Values:	keyword

Specifies the type of information to be included in the list file. The acceptable values for keyword are as follows:

ALL	Display everything
INPUT_MACHINE_CODE	Display source machine code.
MACHINE_CODE	Display both source machine and target machine code.
MESSAGES	Display messages.
NONE	Display the image identification line and summary only.
SOURCE	Include source code if available.
STATISTICS	Display VEST statistics that describe characteristics of the input and output images, such as the number of blocks, images, and external references found, the number of code bytes found, and the number of VAX instructions translated

Description

If you specify /SHOW without a keyword, the qualifier is equivalent to /SHOW=ALL. If you specify /NOSHOW, the qualifier is equivalent to /SHOW=NONE.

Example

```
$ VEST/LIST=SIEVE_5_22_TV.LIS/SHOW=(MESSAGES,STATISTICS) SIEVE
```

This example generates the following list file:

```
VEST V1.1-25 built at Apr 19 1993 13:38:36 starting at May 17 1993 17:12:36
with command line:
VEST/LIST=SIEVE_5_22_TV.LIS/SHOW=(MESSAGES,STATISTICS) SIEVE
 Image "SIEVE", "V1.0", 14-OCT-1991 14:20:19.13
! VEST Statistics:
!
! OpenVMS VAX Image : VST_00:[FREAN.STORM.TEST]SIEVE.EXE;.
!                  : 5 block(s), 8 image sections(s), 3 external reference(s) to 1
image(s).
! Code found       : 206 bytes -- 206 in pass 1 (100%) + 0 in pass 2.
! Completed at     : May 17 1993 17:12:39.
! Completion status : %VEST-I-TRANSOK, Translation completed successfully.
!
! OpenVMS Alpha Image : VST_00:[FREAN.STORM.TEST]SIEVE_TV.EXE;.
!                   : 51 block(s), 7 image sections(s), 531 external reference(s) to 2
image(s).
!                   : 45 VAX instructions translated into 260 Alpha AXP instructions.
! Expansion         : Instructions: 5.7x Disk blocks: 10.1x.
! Message summary by category:
!
! 5 messages in VERBOSE category:
!   1       INFO    READING - Reading file !AZ
!   1       INFO   NOHIF - HIF file !AZ not found
!   1       INFO   PASS1 - Starting analysis pass 1
!   1       INFO   PASS2 - Starting analysis pass 2
!   1       INFO ENDPASS2 - Ending analysis pass 2 -- beginning code generation and
output
$ VEST/LIST/SHOW=(MACHINE_CODE,SOURCE) COBTST_D
```

This example requests a listing file that includes both the VAX and Alpha AXP machine code as well as the original source code. This type of listing is useful when you are debugging translated images. The following excerpt from the listing shows how VEST lays out the various types of code so you can see how they interrelate:

```
DCAUS\DCAUS+0 [COBOL] 1
.
.
.
DCAUS\DCAUS\61
\61 MOVE L TO NULL-TIME. 2
\66 PERFORM TIMRB.
                                        000008BF:       MOVL 84(R11),CC(R11) 3
                                        000008C4:       MOVAB B^000008CB,(R11)
                                        000008C8:       BRW     00000B7E
 * 000102D0:       LDL    R19,FF84(R11) 4
 000102D4:  LDA    R20,88CB(R15)
 * 000102D8:       STL    R20,0(R11)
 000102DC:  STL    R19,FFCC(R11)
 * 000102E0:       BR     10DD0
.
.
.
```

The list file lays out the source code, VAX code and Alpha AXP code as follows:

1 A line identifying the source code module, routine, line number, and, optionally, the source code language. The source code language appears when the line corresponds to an entry point in the image. This line is equivalent to the "At:" line in error messages that identifies a location in the image. VEST derives the location information from the image's debugger symbol table (DST). Depending on the compiler, the DST may not have the information VEST needs to pick up the corresponding line of source code at the beginning of a subroutine.

2 One or more lines of source code beginning at the previously identified location and corresponding to a basic block of machine code. Each line is equivalent to a "Source:" line in error messages. To conserve space, VEST compresses white space, truncates the source line to 63 characters and omits lines that do not actually generate VAX code. The exactness of the correlation between the line numbers and the actual source depends on the image's DST, which in turn depends on the compiler that created it.

3 The VAX code corresponding to a basic block. Each line begins with the VAX offset within the original image.

4 The Alpha AXP code corresponding to the VAX code displayed above and to the right. Each line begins with the Alpha AXP offset within the translated image. An asterisk indicates Alpha AXP code equivalent to the beginning of a single translated VAX instruction.

Note

The asterisk may not be meaningful. By default, scheduling causes the Alpha AXP instructions corresponding to VAX instructions to be intermixed in many cases. However, if you translate an image specifying /PRESERVE=INSTRUCTION_ATOMICITY, the placement of the asterisks is meaningful

/SIF

Creates a symbol information file (.SIF file) for the input image and optionally specifies a file name.

Default:	/NOSIF
Format:	/SIF [=filespec]
Qualifier Values:	filespec

Specifies the .SIF file to be created. If you specify /SIF without a filespec, VEST creates a file named image.SIF in the current directory.

Description

You create and use .SIF files when you are working with shareable images that will be translated repeatedly or must interoperate with native versions of the same images. The .SIF file that VEST creates describes the global symbol table (GST) and symbol vector entries in the translated shareable image. The individual descriptions are called directives . You create and use a .SIF file whenever you need to preserve and control the ordering or contents of the GST and symbol vector. If you include the .SIF file in the search path and then retranslate the same image, VEST uses the .SIF directives to determine the ordering and contents of the new translated image's GST and symbol vector. In the absence of a .SIF file, VEST creates the symbol vector entries in an undefined order.

You can also specify the /SIF qualifier when a .SIF file for the image already exists in the search path. In this case, the directives in the existing file determine how VEST defines the corresponding symbols. The default VEST behavior applies to whatever symbols are not described in the existing file. VEST then creates a new .SIF file that includes the directives from the existing file plus directives for the remaining symbols.

See Section 6.4 for further information about creating and using .SIF files.

Example

```
$ VEST/SIF MYSHR
```

If MYSHR.EXE is a shareable image, the command creates a file called MYSHR.SIF, a file that fully describes the contents and layout of the symbol vector and GST of the translated image MYSHR_TV.EXE. This .SIF file, whether left unchanged or modified by hand, becomes an input file to future translations of MYSHR.EXE, allowing full control of the shareable image external interface.

```
$ VEST/SIF=SIFDIR: MYSHR
```

VEST writes the file MYSHR.SIF to the directory SIFDIR: rather than to the current directory.

/TRACEBACK

Controls whether the traceback handler is activated when an error occurs.

Default: /TRACEBACK if the input image was linked with the /TRACEBACK qualifier; /NOTRACEBACK if the input image was not linked with the /TRACEBACK qualifier.

Format:	/TRACEBACK

Description

In a translated image, the traceback information consists of hexadecimal rather than symbolic values.

By default, VEST creates a translated image with the same /TRACEBACK and /DEBUG settings enabled as those for the input image. By using the /[NO]DEBUG and /[NO]TRACEBACK qualifiers, you can change the level. See the description of the /DEBUG qualifier for further information.

If the input image invokes the debugger and includes traceback information but you want the translated image to do neither, then you must specify both /NOTRACEBACK and /NODEBUG. If you specify /NOTRACEBACK only, VEST ignores the qualifier when the input image was linked with /DEBUG.

Example

```
$ VEST/NOTRACEBACK STORM_GRAPHIC
```

The /NOTRACEBACK qualifier is useful to VEST when used in the following way. If the image STORM_GRAPHIC.EXE is normally linked with the /NOTRACEBACK qualifier, VEST may be able to find more code at translation if the image is relinked with the /TRACEBACK qualifier and then translated with /NOTRACEBACK. This improves the translation without affecting the run-time behavior of the image.

If the image terminates with an unhandled exception, no traceback information will be displayed on the terminal.

```
$ VEST/TRACEBACK PLANT_DATA
```

If PLANT_DATA_TV.EXE terminates with an unhandled exception, the terminal displays information showing the active call chain at the time of the exception. This behavior does not depend on whether or not the image was linked with /TRACEBACK. The display does not include information normally derived from a debugger symbol table (DST).

/VIEW

Specifies the types of flowgraphs to place in a graph file.

Default:	/VIEW=(ERROR=SOURCE_CODE)
Format:	/VIEW =(keyword[,...])
Qualifier Values:	keyword

Specifies which types of flowgraph to include in the flowgraph file. The acceptable values for keyword are as follows:

ALL	Include all kinds of graphs
CALL	Include a call graph in the flowgraph file.

COMPLETE[= (option[,...])]	Include a complete flowgraph that includes information defined by option as follows: ALL Include all possible option information INPUT_MACHINE_CODE Include input machine code MACHINE_CODE Include input and output machine code NONE Include none of the option the option SOURCE_CODE Include source code, if available VERBOSE Include information on entry masks, masks, stack depths, and resources used and set
ERROR[=(option[,...])]	Include an error flowgraph that includes information defined by option. The acceptable values for option are the same as those defined for the COMPLETE keyword.
NONE	Do not include any flowgraphs

Description

See the description of the /FLOWGRAPH qualifier for definitions of the three kinds of flowgraphs.

/VIEW forces the /FLOWGRAPH qualifier unless you also specify =NONE. If you specify /VIEW without a keyword, the qualifier is equivalent to /VIEW=ALL. If you specify /NOVIEW, the qualifier is equivalent to /VIEW=NONE.

Specifying /VIEW=CALL or /VIEW=COMPLETE turns off the default setting /VIEW=ERROR=SOURCE_CODE unless you explicitly specify this ERROR keyword as well.

/WARNINGS

Selects messages to be displayed that are otherwise disabled by default or suppresses the display of messages.

Default:	/WARNINGS=NONE
Format:	/WARNINGS =keyword[,...])
Qualifier Values:	keyword

Specifies a category of message. The acceptable values for keyword are as follows:

ALL[=FULL]	Enable all message categories controllable by the /WARNINGS qualifier. Add =FULL to override the suppression threshold for these messages.
MESSAGE_ID=([NO]message [,...])	Display or disable one or more specified messages. Specify message to display a message and override the suppression threshold level for each. Specify NOmessage to disable a specific message (NOVAXDFLOAT, for example). Use a wildcard ( ) * to specify all messages that begin with one or more preceding letters (for example, READCF). *

PERFORMANCE[=FULL]	Display all messages related to performance. Add =FULL to override the suppression threshold for all messages of this type.
SOURCE_ANALYSIS[=FULL]	Display all messages related to source analysis. Add =FULL to override the suppression threshold for all messages of this type.
SYNCHRONIZATION[=FULL]	Display all messages related to synchronization. Add =FULL to override the suppression threshold for all messages of this type.
VERBOSE[=FULL]	Display all verbose messages. Add =FULL to override the suppression threshold for all messages of this type.

All keyword values except All and NONE are negatable

(NOPERFORMANCE, for example).

Description

/WARNINGS is equivalent to /WARNINGS=ALL.

/NOWARNINGS is equivalent to /WARNINGS=NONE.

Use the /WARNINGS qualifier to request messages you want VEST to display in addition to the standard messages VEST displays by default. You can control the additional message displays in the following ways:

You can select among the following categories of informational messages:

- Performance messages note code in the input image that can cause the translated image to run more slowly.

- Source analysis messages point out unconventional code patterns in the input image that can indicate latent bugs or nonportable features in the source.

- Synchronization messages note possible uses of features requiring some form of synchronization, such as asynchronous system traps (ASTs), asynchronous I/O, or multiprocessor execution. Messages in this category may mean that you need to use the /PRESERVE=INSTRUCTION_ATOMICITY or /PRESERVE=MEMORY_ATOMICITY qualifier.

- Verbose messages report on VEST progress during translation.

Table A-1 lists the applicable VEST messages by category.

You can choose to lift the VEST suppression threshold on all messages within a category (for example, SOURCE_ANALYSIS=FULL) or on individual messages (MESSAGE_ID=(VAXDLOAT). By default, VEST limits the number of times an individual message is actually displayed and then summarizes the total number of suppressed messages in the list file. The count equals the number of times VEST would have displayed the messages if the suppression threshold were lifted.

You can disable individual messages with the /WARNINGS=MESSAGE_ID=NO message setting. For example, /WARNINGS=MESSAGE_ID=NOWRITEJSBRET disables all occurrences of the WRITEJSBRET message.

If you have disabled specific WARNING-level messages and VEST issues no other WARNING messages, as well as no ERROR or FATAL messages, VEST returns the status %VEST-I-TRANSOK. In other words, VEST ignores disabled WARNING messages when it determines the exit status that is returned and that is displayed in the statistics section of the list file.

Use the VEST qualifier /SHOW=STATISTICS to include the counts of suppressed messages in the list file.

Examples

1.

```
$ VEST/WARNINGS=(SOURCE_ANALYSIS=FULL,VERBOSE)
```

This example requests the source analysis and verbose messages and lifts the suppression threshold on the source analysis messages only.

2.

```
$ VEST/WARNINGS=(VERBOSE,MESSAGE_ID=(STKUNAL,READCF*))
```

This example requests the verbose messages, STKUNAL messages, and all messages whose identifier begins with READCF. The /MESSAGE_ID qualifier also lifts the suppression threshold on STKUNAL and READCF messages.

3.

```
$ VEST/WARNINGS=(SOURCE_ANALYSIS,MESSAGE_ID=NOVAXPACKED)
```

This example requests the source analysis messages with the exception of the VAXPACKED message.

VEST/DEPENDENCY

The VEST/DEPENDENCY command analyzes one or more input images to identify external references to shareable images. If a referenced shareable image in turn refers to another shareable image, VEST/DEPENDENCY searches that image for references to any shareable images, and so on. Using this information, you can determine the correct order in which to translate a set of images. By translating images in the correct order, you ensure that VEST has access to information it needs about shareable images as it translates each image.

By default, VEST/DEPENDENCY creates a VAX/DEC Module Management System (MMS) description file. Use the command file VEST_MMS_DRIVER.COM to execute this file as described in the description of the /DEPENDENCY /MMS_DESCRIPTION qualifier.

Format VEST/DEPENDENCY[/qualifier,...] image[.EXE] list[.DAT]/FILE_LIST

Qualifiers	Defaults
/FILE_LIST	None. Modifies list[.DAT] parameter
/FLOWGRAPH[= filespec]	/[NO]FLOWGRAPH
/LIST[= filespec]	/[NO]LIST
/MMS_DESCRIPTION[= filespec]	/MMS_DESCRIPTION
/VIEW_EQUIVALENCE_NAMES	/VIEW_EQUIVALENCE_NAMES=AUTOMATIC

/FILE_LIST

Modifies the VEST/DEPENDENCY parameter, identifying it as the name of a file containing a list of images to be examined.

Default: None.

Format: /FILE_LIST

Description

Use the /FILE_LIST qualifier when you want to examine more than one image at a time.

The /FILE_LIST qualifier is position dependent. It must modify the command parameter, which specifies a text file listing file specifications. The file specification modified by /FILE_LIST has a default extension of .DAT. The text file specifies images to be examined, where each specification starts on a new line in the first position. These image specifications have a default extension of .EXE. One way to create a list of files is to use the DIRECTORY command as shown in the following example:

```
$ DIR/NOHEAD/NOTRAIL/COLUMN=1/OUTPUT= file *.EXE/EXCLUDE=*_TV.EXE
```

/FLOWGRAPH

Enables the creation of a dependency graph file to be processed by the Flowgraph utility and optionally specifies a file name.

Default:	/NOFLOWGRAPH
Format:	/FLOWGRAPH [=filespec]
Qualifier Values:	filespec

Identifies the file specification for the flowgraph file.

Description

When you specify /FLOWGRAPH, VEST/DEPENDENCY creates a flowgraph file. The default name for the file is input.GRAPH, where input is the name of the input image file or the input list file if the /FILE_LIST qualifier was used. Use the Flowgraph program to convert the file into a PostScript representation of the image dependencies. The PostScript graph displays the dependencies as a tree.

/LIST

Requests a list file and optionally specifies a file name.

Default:	/NOLIST
Format:	/LIST [=filespec]
Qualifier Values:	filespec

Identifies a file specification for the list file.

Description

The list file contains error and status messages. If you do not include a file specification, VEST/DEPENDENCY writes the list file to the current directory and defaults to the input image's file name with the suffix _TV and the extension LIS. The following is an example file called SIEVE_TV.LIS:

```
92-06-22 16:59:13 (VEST T1.0-24 built Jun 19 1992 11:56:18)
VEST/DEPEND/FLOWGRAPH/LIST SCRATCH_DEPEND/FILE_LIST
```

/MMS_DESCRIPTION

Requests an MMS description file and optionally specifies a file name.

Default:	/MMS_DESCRIPTION
Format:	/MMS_DESCRIPTION [=filespec]
Qualifier Values:	filespec

Identifies the file specification for the MMS input file.

Description

> The /MMS_DESCRIPTION qualifier enables VEST /DEPENDENCY to create a VAX/DEC Module Management System (MMS) description file. The file directs MMS to issue VEST commands that translate interdependent images in the correct order. Use the VEST_MMS_DRIVER.COM command file to execute the MMS description file; do not process it directly with MMS. The command file defines the logical name VEST$FULL_INCLUDE, required by the MMS description file to execute correctly. Images that have already been translated will not be retranslated.
>
> If you do not include a file specification, VEST/DEPENDENCY writes the MMS file to the current directory and defaults to the input image's file name and the extension MMS.

/VIEW_EQUIVALENCE_NAMES

Specifies whether the image dependency graph includes a full equivalence name for each image name.

Default:	/VIEW_EQUIVALENCE_NAMES=AUTOMATIC
Format:	/VIEW_EQUIVALENCE_NAMES =keyword
Qualifier Values:	keyword

The keyword must be one of the following values:

ALWAYS	Causes equivalence names to always be shown.
AUTOMATIC	Causes an equivalence name to be shown only when it differs from SYS$SHARE:image.EXE.
NEVER	Causes equivalence names never to be shown.

Description

> An equivalence name is the full file specification for an image to which a logical name has been assigned. The graph shows the logical name when an equivalence name is not shown.

DSTGRAPH

The DSTGRAPH command reads an image.STI file as input and creates a PostScript file showing diagrams of selected records and modules that highlight unaligned and skewed data. Before using the DSTGRAPH command, first recompile and relink the source files using the /DEBUG or /DEBUG=ALL qualifier so that the OpenVMS VAX image includes a full Debug Symbol Table (DST). Then use the VEST qualifier /DST to create the image.STI file. (See the description of the VEST command for further details).

The DSTGRAPH command qualifiers allow you to tailor the information included in the PostScript file:

Use the /UNALIGNED, /SKEW, or /ALL qualifiers to select the type of data to be shown.

Use the /SELECT qualifier to restrict processing to specific data items.

Use the /SCALE_FACTOR to control the size of the rectangles and diagrams.

Use the /WIDTH qualifier to control how much data in bytes is represented in each bar of the diagrams.

If possible, use the information DSTGRAPH provides you about unaligned data to modify data structures in your application's source files as you port the application to OpenVMS Alpha. See the manuals Migrating to an OpenVMS Alpha System: Planning for Migration and Migrating to an OpenVMS Alpha System: Recompiling and Relinking Applications for further information.

Format DSTGRAPH[/qualifier,...] image[.STI]

Qualifiers	Default
/ALL	Not enabled
/SCALE_FACTOR=[factor]	/SCALE_FACTOR=1.0
/SELECT=" string"	Not enabled
/SKEW	Not enabled
/UNALIGNED	/UNALIGNED
/WIDTH=[bytes]	/WIDTH=8

/ALL

Flags unaligned and skewed data while depicting all modules and records.

Default: Not enabled

Format: /ALL

Description

The /ALL qualifier causes DSTGRAPH to depict information about all the data in the image and also to highlight data that is unaligned or skewed.

The /ALL qualifier is not negatable.

/SCALE_FACTOR

Specifies a decimal number that controls the size of the rectangles and data diagrams in the PostScript output file.

Default: /SCALE_FACTOR=1.0

Format: /SCALE_FACTOR =factor

Qualifier Values: factor

Specifies a decimal number between 0.7 and 2.25.

Description

The /SCALE_FACTOR qualifier allows you to expand or contract the size of the rectangles and data diagrams. The default scale of 1.0 specifies 6 to 8 point size type for most data items and two columns per page. A scale of 0.75 results in three columns per page.

The /SCALE_FACTOR qualifier is not negatable.

/SELECT

Requests that only data items whose names contain string be subject to processing by the /UNALIGNED, /SKEW, or /ALL qualifiers.

Default: Not enabled

Format: /SELECT ="string"

Qualifier Values: "string"

Defines a string representing all or part of a data item name. The quotes around string are required to preserve case settings.

Description

> The /SELECT qualifier allows you to confine DSTGRAPH to considering specific data items.
>
> The /SELECT qualifier is not negatable.

/SKEW

Specifies that DSTGRAPH include only modules or records that include skewed and unaligned data in the PostScript output file.

Default: Not enabled

Format: /SKEW

Description

> Use /SKEW to include both unaligned and skewed data only. A data item, such as a character string or array, is skewed if it occupies more bars than necessary in its DSTGRAPH diagram. The /WIDTH qualifier defines the length in bytes that DSTGRAPH uses to determine whether or not data is skewed. The default width is a quadword. (See the description of the /WIDTH qualifier.) For example, an 84-byte array is skewed if it starts at address 6 and therefore occupies 12 aligned quadwords. The array would be aligned better if it were to start at address 0 and therefore occupy 11 aligned quadwords. Skewed data forces the OpenVMS Alpha system to perform more memory accesses than necessary.
>
> The /SKEW qualifier is not negatable.

/UNALIGNED

Specifies that DSTGRAPH include only the modules or records that include unaligned data items in the PostScript output file.

Default: /UNALIGNED

Format: /UNALIGNED

Description

> Use the /UNALIGNED qualifier to include unaligned data only in the PostScript file. A data item is unaligned if its address is not a multiple of its own size in bytes.
>
> The /UNALIGNED qualifier is not negatable.

/WIDTH

Specifies the width in bytes of a single bar of data in the PostScript output file.

Default: /WIDTH=8

Format: /WIDTH [=bytes]

Qualifier Values: bytes

An integer that must be a power of two and that represents a number of bytes.

Description

The /WIDTH qualifier determines how many bytes of data are depicted in each bar of the diagrams. The default width of 8 bytes is equivalent to the natural word size of an Alpha system. Which width you choose depends on the type of skew you are interested in. (See the description of the /SKEW qualifier.) Specifying /WIDTH=32 draws pictures and detects skew corresponding to DECchip 21064 32-byte cache blocks. Specifying /WIDTH=512 draws pictures corresponding to a VAX page.

The /WIDTH qualifier is not negatable.

FLOWGRAPH

The FLOWGRAPH command creates one or more PostScript formatted flowgraphs that represent all or part of an image's program flow or that show an image's dependency on shareable images. A .GRAPH file created in one of two ways is the single command parameter:

A .GRAPH file created by the VEST/FLOWGRAPH/VIEW command. Keywords to the /VIEW qualifier determine the .GRAPH file contents and the types of flowgraphs created by the FLOWGRAPH command. Three types are possible: a complete flowgraph, a call flowgraph, and an error flowgraph. See the description of the VEST command for details.

A .GRAPH file created by the VEST/DEPENDENCY /FLOWGRAPH command from which the FLOWGRAPH command creates a dependency graph. See the description of the VEST/DEPENDENCY command for details.

Format: FLOWGRAPH[/qualifier,...] image[.GRAPH]

Qualifiers	Defaults
/OUTPUT[= filespec]	/OUTPUT
/SCALE_FACTOR=[factor]	/SCALE_FACTOR=1.0
/STARTING_ADDRESS=[address]	None

/OUTPUT

Enables the creation of a PostScript output file and optionally includes a file specification.

Default: /OUTPUT

Format: /OUTPUT [=filespec]

Qualifier Values: filespec

Defines the file specification for the PostScript output file.

Description

By default, FLOWGRAPH creates a PostScript output file that you can either print or display online using Display PostScript. The types and content of the flowgraphs created depend on the command used to generate the input .GRAPH file.

If you do not provide a file specification, FLOWGRAPH writes the output file to the current directory with the same name as the input file and .PS as the extension.

/SCALE_FACTOR

Specifies a decimal number that controls the size of the one or more flowgraphs in the output file.

Default: /SCALE_FACTOR=1.0

Format: /SCALE_FACTOR [=factor]

Qualifier Values: factor

Specifies a decimal number.

Description

> The default scale factor of 1.0 specifies 6-point text within each flowgraph box and positions the boxes on a quarter inch grid. Scale factors smaller than 1.0 shrink the text and boxes, allowing more of the flowgraph to appear on a single page. Scales greater than 1.0 expand the text and boxes so that less of the graph fits on a single page.
>
> The default scale typically prints up to 8 blocks across a page and up to 20 blocks down a page. Try the default scale first. FLOWGRAPH tells you how many pages the resulting graph occupies. If the number is too large, try a smaller scale (the number of pages decreases as the square of the scale). Scales in the range 0.667 through 1.0 are readable. Scales down to 0.5 are harder to read. Scales on the order of 0.1 can be useful to get an overall picture of all the code found. For scales below 0.25, FLOWGRAPH omits the text inside boxes, thus cutting the size of the PostScript file almost in half. Scales greater than 1.0 are useful for making slides, or perhaps for Display PostScript on a low-resolution screen.

/STARTING_ADDRESS

Requests that a complete or error flowgraph document only the basic blocks starting with the specified address.

Default: None

Format: /STARTING_ADDRESS [=address]

Qualifier Values: address

Specifies a hexadecimal number representing the starting address of a basic block.

Description

> This qualifier allows you to narrowly restrict a flowgraph to the basic blocks that start with the address you specify.

B. Error and Status Messages

This appendix discusses the following topics:

Topic	See
Interpreting OMSVA messages	Section B.1
Alphabetical listings of VEST, DSTGRAPH and FLOWGRAPH messages	Section B.2

B.1 Interpreting VEST Messages

VEST error and status messages identify many different conditions within an image. For example, VEST can identify code that affects a translated image's run-time performance or code that prevents successful translation. Each message has one of the following severity levels:

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

The type and number of error messages that VEST includes in its listing file depend on the /WARNINGS qualifier setting. In addition to standard messages that VEST displays by default, you can control the display of performance messages, source analysis messages, synchronization messages, and verbose messages. Furthermore, each message has an associated threshold that determines the maximum number of times VEST displays it for any given translation. See the description of /WARNINGS in Appendix A for further information.

Section B.2 provides explanations for each message and, when feasible, suggests a user action. A VEST message as it is displayed or written to the log file consists of the facility name (VEST), 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:

```
%VEST- I -FLAGASYNC, Image calls system service SYS$GETDVI which may perform
asynchronous memory access
```

Additionally, whenever a message pertains to a specific location in the image, VEST displays two or three additional lines of information. The extent of the information provided depends on the level of debugging and traceback information present in the input image. VEST uses information in the DST of an OpenVMS VAX image to trace an error encountered at a specific address in the image back to the line of source corresponding to that address.

Section B.1.1 describes the possible levels of debugging and traceback information available in an input image. Section B.1.2 describes the syntax of the location information.

B.1.1 Levels of Debugging and Traceback Information

An image can contain three levels of debugging-related information:

Level	Type of DST	Description
Level 1	No DST	The image was linked or compiled with the /NOTRACEBACK qualifier

Level 2	Traceback DST	The image was linked and compiled with the /TRACEBACK qualifier, which is the default.
Level 3	Full DST	The image was compiled and linked with both the /DEBUG and /TRACEBACK qualifiers. (A compiler may require DEBUG=ALL; see the relevant compiler documentation.)

The extent of the information VEST is able to display depends on which level the input image corresponds to. Level 2 and level 3 messages allow you to trace a possible error condition in the input image directly back to its corresponding source code. These messages include a line that is similar in format to that used by the Performance and Coverage Analyzer (PCA) utility to point to source code from an image. This capability is useful for preparing an OpenVMS VAX applications either for image translation or source code porting. The following example messages show the three possible variations:

Level 1:

```
%VEST-I-FLAGASYNC, Image calls system service SYS$GETDVI which may
perform asynchronous memory access
   At:     00007CA0 (00007CA0)
   Input: CALLS S^#08,@#7FFEE410
```

The level 1 message shows that the FLAGASYNC condition occurs at address 00007CA0 and the VAX instruction at that address is shown in the "Input:" line. No DST information is available to trace the location back to source code.

Level 2:

```
%VEST-I-FLAGMP, Image calls system service SYS$WAITFR which may indicate
multiprocessor operation
   At:     SHELL$CLI_NAME\shell$cli_name\911 [C] (00005963)
   Input: CALLS S^#01,@#7FFEE078
```

The level 2 message uses information available in a traceback DST to describe the module name, routine, and line in the source code (SHELL$CLI_NAME\shell$cli_name\911), the source code language (C), and the image address (00005963).

Level 3:

```
%VEST-I-VAXPACKED, VAX packed decimal string opcode MULP -- will be emulated
   At:     DCAUS\DCAUS\114 [COBOL] (00000A4C)
   Source: COMPUTE RLS-P17 = RLS-P9 * RLS-P7.
   Input: MULP S^#07,A4(R11),S^#09,A8(R11),S^#11,B0(R11)
```

The level 3 message takes advantage of a full DST to add the actual line of source code (Source: COMPUTE RLS-P17 = RLS-P9 RLS-P7.) In all three messages, the * final "Input:" line describes the VAX instruction found at the image address.

B.1.2 Location Information Syntax

As the examples in Section B.1.1 illustrate, the location information syntax for level 1 messages differs from the syntax for level 2 and level 3 messages. The following description explains each syntax in detail:

Level 1 location information syntax:

At: address (address)

Input: vax_instruction

The variables are defined as follows:

address	The address in the image that prompted VEST to issue the message (00007CA0, for example)

vax_instruction	The VAX instruction at that address (CALLS S^#01,@#7FFEE078, for example)

Level 2 and level 3 syntax:

```
At: module\routine\line [language] (address)
[Source: source_line]
Input: vax_instruction
```

The variables are defined as follows:

module	**The name of the source program module (DCAUS, for example).**
routine	The name of the source routine within the module (shell$cli_name, for example) or the PSECT name if the source language is VAX MACRO-32
line	The line number within the routine or PSECT (114, for example).
language	The source language (VAX COBOL, for example).
address	The address in the image that prompted VEST to issue the message (00007CA0, for example).
source_line	Level 3 messages only. The corresponding source code line, truncated to 63 characters and including compressed white space.
vax_instruction	The VAX instruction at that address (CALLS S^#01,@#7FFEE078, for example).

B.2 The Messages

This appendix describes OMSVA error and status messages:

Section B.2.1 describes VEST messages.

Section B.2.2 describes DSTGRAPH messages.

Section B.2.3 describes FLOWGRAPH messages.

B.2.1 VEST Messages

BADCASEFALL, Invalid fallthrough for CASE instruction-may need to use HIF: '+address' caselimit 'limit'

Facility: VEST, VAX Environment Software Translator utility

Explanation: VEST has detected a CASEx instruction with a suspect default case.

User Action: Make an .HIF file entry for the image being translated at 'address' with the caselimit property set to 'limit'. For example:

```
+00000698 caselimit 5
```

BADCASELIMIT, CASE instruction has a limit greater than 64K - following assumed and may be used in HIF: 'address' caselimit 'limit'

Facility: VEST, VAX Environment Software Translator utility

Explanation: VEST has detected a CASEx instruction with a limit value greater than or equal to 64K. The message suggests a more realistic value for 'limit'.

User Action: Make an .HIF file entry for the image being translated at 'address' with the caselimit property set to 'limit'. For example:

```
+00000698 caselimit 5
```

BADCASETARG, Invalid target for CASE instruction with non-zero lower bound-may need to use HIF: 'address' caselimit 'limit'

Facility: VEST, VAX Environment Software Translator utility

Explanation: VEST has detected a CASEx instruction with a non-zero lower bound and one of the entries in the displacement table contains an invalid target. The invalid target may indicate an incorrect caselimit.

User Action: If the caselimit is incorrect, make a .HIF file entry for the image being translated at 'address' with the caselimit property set to 'limit'. For example:

+00000698 caselimit 5

BADEXE, Image is not translatable because it is not an OpenVMS VAX V4.0 or later image ['text_string']

Facility: VEST, VAX Environment Software Translator utility

Explanation: VEST detected that the input image is not translatable. If present, 'text_string' explains why it is not translatable.

User Action: Check to see that you have specified the correct image.

CALLENTRY, Untyped GST entry assumed to be a callentry-can override via HIF

Facility: VEST, VAX Environment Software Translator utility

Explanation: The global symbol table (GST) in the input image contains many addresses marked as symbols, some of which may actually be CALL entrypoints. VEST tests each symbol address to see if it is a CALL entrypoint. If it is, VEST adds the address as a CALL entrypoint to its list of addresses to be translated.

User Action: None required. However, you can use an .HIF entry to override the VEST designation of the specified offset as a CALL entrypoint and make it a JSB entrypoint instead. For example:

+00002FFC jsbentry

CMASKOVERLAP, Call mask at 'address' overlaps previously found code so it won't be translated-use HIF to override: '+address' caselimit 'limit'

Facility: VEST, VAX Environment Software Translator utility

Explanation: VEST has encountered a routine at 'address' whose first two bytes (the call mask) overlap a previously parsed VAX instruction. One of the two parsings is incorrect. If the routine is executed at run time, VAX code is interpreted in the region of overlap.

User Action: None required.

If you want to investigate the code that incurred this error, produce an error flowgraph of the image to show the conflicting blocks of code. If the overlapping bytes at 'address' are in fact the target of a call, use an .HIF property ('address' callentry) to direct VEST to parse the overlapping bytes correctly. If the overlapping bytes are in fact VAX code, then the calling instruction is not really a VAX instruction; use an .HIF property ('diff_address' dataentry) to direct VEST away from parsing the calling instruction, or use an .HIF property ('diff_address' caselimit) if the path came from a CASE instruction 'diff_address' with no fallthrough path or an incorrectly specified table size. ('diff_address' is an address other than 'address', which you must determine from examining the error graph for the image.)

If the overlap was encountered in the second VEST pass to identify code, an alternative action is to specify /OPTIMIZE=NOSCAN, which bypasses the second pass.

DEPENDERROR, Dependency analysis unsuccessful

Facility: VEST, VAX Environment Software Translator utility

Explanation: VEST/DEPENDENCY encountered one or more ERROR level error messages. This message is the status code returned to DCL.

User Action: Review the ERROR level messages VEST /DEPENDENCY issued to identify the problem.

DEPENDFATAL, Translation was impossible

Facility: VEST, VAX Environment Software Translator utility

Explanation: VEST/DEPENDENCY encountered one or more FATAL level error messages. This message is the status code returned to DCL. msg_text>(User Action) Review the FATAL level messages VEST/DEPENDENCY issued to identify the problem.

DEPENDOK, Dependency analysis completed successfully

Facility: VEST, VAX Environment Software Translator utility

Explanation: VEST/DEPENDENCY successfully completed its analysis. This message is the status code returned to DCL.

User Action: None.

DEPENDWARN, Dependency analysis completed with warnings - review them before using output files.

Facility: VEST, VAX Environment Software Translator utility

Explanation: VEST/DEPENDENCY issued at least one WARNING level error message. This message is the status code returned to DCL.

User Action: Examine the WARNING level error messages before using an MMS description file or dependency graph that VEST/DEPENDENCY has created.

ENDPASS2, Ending analysis pass 2 - beginning code generation and output

Facility: VEST, VAX Environment Software Translator utility

Explanation: VEST issues this message at the completion of its second analysis of the input image.

User Action: None.

EXPRCOM, Missing expected ',' in expression 'string' - expression ignored

Facility: VEST, VAX Environment Software Translator utility

Explanation: While reading an .IIF or .HIF file, VEST detected a missing comma (,) from the line indicated.

User Action: Edit the .IIF or .HIF file specified in the previous READIF message to add the missing comma. Because VEST generates .IIF files automatically, this message is more likely to pertain to an .HIF file that has been edited by hand. Retranslate the image if necessary.

EXPRERR, Unknown token type in expression 'string'-expression ignored

Facility: VEST, VAX Environment Software Translator utility

Explanation: VEST has encountered an internal error.

User Action: Submit a Software Performance Report (SPR) describing the error message.

EXPRNYI, Unimplemented token 'string1' in expression 'express' - expression ignored

Facility: VEST, VAX Environment Software Translator utility

Explanation: While reading an .IIF or .HIF file, VEST encountered an invalid property string value or token. Because VEST generates .IIF files automatically, this message is more likely to pertain to an .HIF file that has been edited by hand.

User Action: Either ignore the message or replace the token with a supported string value.

EXPRRPAR, Missing expected ')' in expression 'express' - expression ignored

Facility: VEST, VAX Environment Software Translator utility

Explanation: While reading an .IIF or .HIF file, VEST failed to encounter an expected closing parenthesis in the line indicated. Because VEST generates .IIF files automatically, this message is more likely to pertain to an .HIF file that has been edited by hand.

User Action: Edit the line in the file specified in the previous READING message to add the closing parenthesis and retranslate the image if necessary.

EXPRTOK, Invalid token 'string' in expression 'express' - expression ignored

Facility: VEST, VAX Environment Software Translator utility

Explanation: While reading an .IIF or .HIF file, VEST encountered an invalid property value string or token, 'string', which it could not match to any supported property values for information files. Because VEST generates .IIF files automatically, this message is more likely to pertain to an .HIF file that has been edited by hand.

User Action: If necessary, replace the invalid token with one that is supported and retranslate the image.

FLAGAST, Image calls system service 'sys_serv' which may use AST routines

Explanation: VEST has encountered a call to a system service whose normal use involves asynchronous system trap (AST) threads that can interrupt a main program at arbitrary points. This message indicates that the image may contain accesses to variables that are shared between a main thread and an AST thread. Whenever two threads of execution share memory on nonaligned boundaries, a synchronization problem may occur.

User Action: Check the OpenVMS VAX image and the program sources, if possible, to check for AST routines that share data. If a system call specifies a shared area that is not an integral multiple of aligned quadwords, Compaq recommends that you rewrite the original OpenVMS VAX program to align quadwords wherever possible, then retranslate.

If you cannot rewrite the original program and need to ensure the exact OpenVMS VAX behavior, retranslate the image with the qualifier /PRESERVE=MEMORY_ATOMICITY. Because this qualifier adversely affects the performance of the translated image, do not use it unless absolutely necessary.

FLAGASYNC, Image calls system service 'sys_serv' which may perform asynchronous memory access

Facility: VEST, VAX Environment Software Translator utility

Explanation: VEST has encountered a call to a system service whose normal use involves asynchronous read or write of a user data buffer or control-block area. This message indicates that the image may contain accesses to variables that are shared by the user program and an asynchronous system update (which appears to be done by another processor). Whenever two threads of execution share memory on nonaligned boundaries, a synchronization problem may occur.

User Action: Check the OpenVMS VAX image and the program sources, if possible, to check for AST routines that share data. If a system call specifies a shared area that is not an integral multiple of aligned quadwords, Compaq recommends that you rewrite the original OpenVMS VAX program to align quadwords wherever possible, then retranslate.

If you cannot rewrite the original program and need to ensure the exact VAX behavior, retranslate the image with the qualifier /PRESERVE=MEMORY_ATOMICITY. Because this qualifier adversely affects the performance of the translated image, do not use it unless absolutely necessary.

FLAGIO, Image calls system service 'sys_serv' which may perform asynchronous I/O

Facility: VEST, VAX Environment Software Translator utility

Explanation: VEST has encountered a call to a system service whose normal use involves asynchronous I/O read or write of a user data buffer or control-block area. This message indicates that the image may contain accesses to variables that are shared by the user program and an I/O controller (which appears to be another processor). Whenever two threads of execution share memory on nonaligned boundaries, a synchronization problem may occur.

User Action: Check the OpenVMS VAX image and the program sources, if possible, to check for AST routines that share data. If a system call specifies a shared area that is not an integral multiple of aligned quadwords, Compaq recommends that you rewrite the original OpenVMS VAX program to align quadwords wherever possible, then retranslate.

If you cannot rewrite the original program and need to ensure the exact OpenVMS VAX behavior, retranslate the image with the qualifier /PRESERVE=MEMORY_ATOMICITY. Because this qualifier adversely affects the performance of the translated image, do not use it unless absolutely necessary.

FLAGMP, Image calls system service 'sys_serv' which may indicate multiprocessor operation

Facility: VEST, VAX Environment Software Translator utility

Explanation: VEST has encountered a call to a system service whose normal use involves multiple processes that can run simultaneously on multiple processors.

User Action: This message indicates that the image may well contain accesses to variables that are shared by multiple processes. Whenever two threads of execution share memory on nonaligned boundaries, a synchronization problem may occur. Check the OpenVMS VAX image and the program sources, if possible, to check for AST routines that share data. If a system call specifies a shared area that is not an integral multiple of aligned quadwords, Compaq recommends that you rewrite the original program to align quadwords wherever possible, then retranslate.

If you cannot rewrite the original program and need to ensure the exact OpenVMS VAX behavior, retranslate the image with the qualifier /PRESERVE=MEMORY_ATOMICITY. Because this qualifier adversely affects the performance of the translated image, do not use it unless absolutely necessary.

HIFEMPTY, No usable definitions found in HIF file 'hif_file'

Facility: VEST, VAX Environment Software Translator utility

Explanation: The .HIF file specified does not contain any usable definitions. For example, the image idents in the file do not match the image ident of the image being translated.

User Action: Make sure that the appropriate .HIF file is available and retranslate the image.

IDENTMISMATCH, Image ident 'ident1' doesn't match ident 'ident2' in file 'file-name'-entries ignored

Facility: VEST, VAX Environment Software Translator utility

Explanation: The image's IDENT field does not match the .IIF IDENT field in the IMAGE record.

User Action: Rerun VEST with the /IIF qualifier to generate a new .IIF file with a correct IMAGE record.

IIFEMPTY, No usable entries found in IIF file 'iif_file'

Facility: VEST, VAX Environment Software Translator utility

Explanation: The information file specified does not contain any usable definitions.

User Action: Make sure that the appropriate information file is available and retranslate the image.

IIFINVALID, Invalid qualifier combination (/IIF and /INTERPRET[={ALL_CODE, WRITEABLE_CODE}])

Facility: VEST, VAX Environment Software Translator utility

Explanation: The VEST command line specified either the /INTERPRET, the /INTERPRET=ALL, or the /INTERPRET=WRITE qualifier in addition to the /IIF qualifier. Because VEST performs no analysis of interpreted code, using one of the /INTERPRET settings listed and the /IIF qualifier results in an invalid .IIF file. In the case of /INTERPRET=WRITEABLE_CODE, the .IIF file is invalid for all code in writable PSECTS, typically MACRO-32 code.

User Action: If you need a valid .IIF file for the input image, reissue the VEST command without the incompatible /INTERPRET setting.

ILLOPC, Illegal opcode 'instruction'-will be interpreted and will fault if executed

Facility: VEST, VAX Environment Software Translator utility

Explanation: VEST decoded the illegal instruction ('instruction') specified in the message. VEST may have been decoding a part of the image that does not actually include code. Another possibility is that the image includes the illegal instruction within a part of the image that is never executed.

User Action: Use a flowgraph to investigate the location where VEST detected the illegal instruction. You may be able to ignore the message. If you discover that the "instruction" is not really code, use a dataentry property record in an .HIF file to tell VEST that the location contains data.

ILLOPSPEC, Illegal operand specifier in instruction 'instruction' - will be interpreted and will fault if executed

Facility: VEST, VAX Environment Software Translator utility

Explanation: VEST decoded an illegal instruction. VEST may have been decoding a part of the image that does not actually include code. Another possibility is that the image includes the illegal instruction within a part of the image that is never executed.

User Action: Use a flowgraph to investigate the location where VEST detected the illegal instruction. You may be able to ignore the message. If you discover that the "instruction" is not really code, use a dataentry property record in an .HIF file to tell VEST that the location contains data.

INSTRMID, Branch target at 'address' overlaps previously found code so it won't be translated-use HIF to override

Facility: VEST, VAX Environment Software Translator utility

Explanation: VEST has encountered an instruction that falls into or branches to a target 'address' whose first bytes overlap a previously parsed VAX instruction. One of the two parsings is incorrect. If the instruction is executed at run time, VAX code is interpreted in the region of overlap.

User Action: Produce an error flowgraph of the image to show the conflicting blocks of code. If the instruction is in fact correct and branches to 'address', use an .HIF property ('address'-1 dataentry) to direct VEST away from the wrong parsing. If the instruction is in fact correct and falls into 'address', use an .HIF property ('diff_address' dataentry) to direct VEST away from a wrong parsing of some instruction that appears to lead to 'address', or use an .HIF property ('case_instruction' caselimit) if the path to 'address' came from a CASE instruction 'case_instruction' with no fallthrough path or a badly specified table size.

If the overlapping bytes 'address' are in fact VAX code, then the instruction at 'location' is not really a VAX instruction. Use an .HIF property ('location' dataentry) to direct VEST away from parsing 'location', or use an .HIF property ('case_instruction' caselimit) if the path to 'location' came from a CASE instruction 'case_instruction' with no fallthrough path or a badly specified table size.

For both instances of 'case_instruction' described above, examine an error graph for the image to determine its value.

If the overlap was encountered in the second VEST pass to identify code, an alternative action is to specify /OPTIMIZE=NOSCAN to bypass that second pass.

INTERNE, Internal consistency error - please submit an SPR with the following information: 'text_string'

Facility: VEST, VAX Environment Software Translator utility

Explanation: VEST has encountered an ERROR-level internal error.

User Action: Submit a Software Performance Report (SPR) describing the error message.

INTERNF, Fatal internal consistency error - please submit an SPR with the following information: 'text_string'

Facility: VEST, VAX Environment Software Translator utility

Explanation: VEST has encountered a FATAL-level internal error.

User Action: Submit a Software Performance Report (SPR) describing the error message.

INTERNW, Internal consistency warning-please submit an SPR with the following information: 'text_string'

Facility: VEST, VAX Environment Software Translator utility

Explanation: VEST has encountered an WARNING-level internal error.

User Action: Submit a Software Performance Report (SPR) describing the error message. The translated image should run correctly, but use it with caution.

ISDBASED, Image section 'isd_number' is based-not translatable

Facility: VEST, VAX Environment Software Translator utility

Explanation: The image to be translated is a based image-that is, the image has been linked so that all code and data are tied to fixed addresses. VEST does not support translating based images.

User Action: Relink the OpenVMS VAX image without the /BASE link option.

ISDCONFLICT, Image section 'isd_number' has a copy/share conflict with other sections on the same 64KB page-not translatable

Facility: VEST, VAX Environment Software Translator utility

Explanation: By default, the OpenVMS Alpha linker creates image sections on 64K-byte boundaries, the maximum page size supported by Alpha AXP systems. In contrast, OpenVMS VAX images assume a page size of 512 bytes. For most images, VEST automatically resizes and combines image sections as necessary to realign them on 64K-byte boundaries. However, if an attempt by VEST to align image sections on 64K-byte boundaries causes an Alpha AXP page to contain both shared and unshared image sections, VEST cannot translate the image.

User Action: Relink the image using /BPAGE=16 as a qualifier to the LINK command ($ LINK/BPAGE=16) on an OpenVMS VAX Version 5.5 or later system.

ISDPROTECT, Image section 'isd_number' is protected-not translatable

Facility: VEST, VAX Environment Software Translator utility

Explanation: This message indicates that the image contains one or more user-written system services and is therefore not translatable. VEST issues this message whenever it encounters an image section descriptor (ISD) that has the protect flag on and is not a message section.

User Action: To migrate this program, either recompile the original sources with a native compiler or rework them and use the MACRO-32 compiler.

ISDVECTOR, Image section 'isd_number' contains privileged change-mode vectors-not translatable

Facility: VEST, VAX Environment Software Translator utility

Explanation: This message indicates that the image contains one or more user-written system services and is therefore not translatable. VEST issues this message whenever it encounters an image section descriptor (ISD) that has the vector flag on and is not a message section.

User Action: To migrate this program, either recompile the original sources with a native compiler or rework them and use the MACRO-32 compiler.

LINKMISMATCH, Image link time 'date_time1' doesn't match link time 'date_time2'

Facility: VEST, VAX Environment Software Translator utility

Explanation: The image's link time does not match the .IIF link time in the IMAGE record.

User Action: Retranslate the shareable image to create a new .IIF file with a matching link time.

LNKSYS, Image is linked against OpenVMS VAX and references symbols in it - not translatable

Facility: VEST, VAX Environment Software Translator utility

Explanation: The input image was linked against a specific version of OpenVMS VAX and references specific symbols in that version. The image is therefore not translatable.

User Action: If possible, fix the sources to remove the references and then recompile and relink to create a translatable image.

LNKSYSOK, Image is linked against OpenVMS VAX but references no symbols in it-translation possible

Facility: VEST, VAX Environment Software Translator utility

Explanation: The input image was linked against a specific version of OpenVMS VAX but is translatable because it makes no references to symbols in that version.

User Action: None.

MSGSUPR, Reached limit for 'type' messages-all future occurrences will be suppressed

Facility: VEST, VAX Environment Software Translator utility

Explanation: VEST has encountered numerous errors of the same type and has suppressed any further display of messages flagging those types of errors.

User Action: None required. Use the VEST qualifier /WARNINGS=MESSAGE_ID= type to remove the threshold for the specified message if you want to be notified every time it occurs.

NAMEMISMATCH, Image name 'name1' doesn't match name 'name2' in file 'filename'-entries ignored

Facility: VEST, VAX Environment Software Translator utility

Explanation: The image name identification ('name2') within the file 'filename' does not match 'name1'. VEST ignores the definitions in 'filename'.

User Action: If 'filename' is an .IIF file, retranslate the shareable library to obtain a valid .IIF file. If 'filename' is an .HIF file, run the translated image on an OpenVMS Alpha system. If the TIE interprets code within the image, it writes entry point information to an .HIF file. Use that .HIF file to retranslate the original image.

NOHIF, HIF file 'hif_file' not found

Facility: VEST, VAX Environment Software Translator utility

Explanation: VEST was unable to find the file 'hif_file'. When VEST tries to open an information file, it searches the following directories in the order given:

· The current default directory

· The directory or directories specified as values to the /INCLUDE_DIRECTORY qualifier, if present, in the VEST command line

· The directory or directories, if any, defined by VEST$INCLUDE

User Action: To make an .HIF file for the input image available, copy it to one of the directories described above. Note, however, that an .HIF file is not required for image translation.

NOIIF, Error opening interface file 'filename'-will assume default interface

Facility: VEST, VAX Environment Software Translator utility

Explanation: VEST was unable to open the specified information file.

User Action: Make the specified file available to VEST by including it in one of the following locations:

· The current default directory

· The directory or directories specified as values to the /INCLUDE_DIRECTORY qualifier, if present, in the VEST command line

· The directory or directories, if any, defined by the VEST$INCLUDE logical name

When VEST tries to open an information file, it searches the directories listed in the order given.

NONSTDCALLS, Non-standard call sets 'resource'

Facility: VEST, VAX Environment Software Translator utility

Explanation: A call sets the specified resource. This behavior does not conform to the VAX Calling Standard for that call. Nevertheless, VEST reproduces the exact behavior in the translated image.

User Action: None.

NONSTDCALLU, Non-standard call uses 'resource'

Facility: VEST, VAX Environment Software Translator utility

Explanation: A call uses the specified resource. This behavior does not conform to the VAX Calling Standard for that call. Nevertheless, VEST reproduces the exact behavior in the translated image.

User Action: None.

NOUSVOFS, No symbol vector offset (usv_offset) specified for cross image reference to offset 'address' in 'name'.IIF

Facility: VEST, VAX Environment Software Translator utility

Explanation: The file 'name'.IIF contains no symbol vector entry for the offset 'address' in the image 'name'. VEST uses the offset in the VAX code within the image 'name' instead, which will cause a slight performance degradation at run time. This message may indicate that you are using an .IIF file that is out of date or otherwise does not match the shareable image actually linked to the translated image.

User Action: Check the file 'name'.IIF to determine why it does not include a symbol vector entry for 'offset'. It may be necessary to delete existing .IIF and .SIF files and then recreate them in a two-step process:

· Translate the shareable image using the /SIF qualifier to create a .SIF file

· Retranslate the shareable image to create a .IIF file.

NOVM, No more virtual memory available to create data structures

Facility: VEST, VAX Environment Software Translator utility

Explanation: VEST is unable to allocate dynamic memory for data structures. Any VEST output files are invalid.

User Action: Increase the page file quota and/or virtual page count. If you have already reached the maximum quotas, try breaking the image up into two or more smaller images.

OPENIN, Error opening 'filename' as input

Facility: VEST, VAX Environment Software Translator utility

Explanation: VEST could not open the image file specified in the command line.

User Action: Reenter the command line with a valid file specification for the image to be translated.

OPENOUT, Error opening 'filename' as output

Facility: VEST, VAX Environment Software Translator utility

Explanation: VEST was unable to open the identified file 'filename'.

User Action: Check to determine why VEST could not open the file. (For example, the /LIST qualifier specified a directory to which you do not have write access or specified a file version that was open and write-locked.) Reenter the VEST command specifying an output file that VEST can open.

PASS1, Starting analysis pass 1

Facility: VEST, VAX Environment Software Translator utility

Explanation: VEST is starting the parsing of VAX instructions from specified entry points.

VEST Action: None.

PASS2, Starting analysis pass 2

Facility: VEST, VAX Environment Software Translator utility

Explanation: VEST is starting to scan the input image for VAX instructions that may have been missed in pass 1.

User Action: Specify /OPTIMIZE=NOSCAN to bypass pass 2.

PRIVOPC, Privileged opcode 'opcode' will be interpreted and will fault if executed

Facility: VEST, VAX Environment Software Translator utility

Explanation: VEST is designed to translate only user- mode OpenVMS VAX images, but encountered a VAX instruction 'opcode' that is defined only to work in kernel mode. If the instruction is actually encountered at run time, the VEST-generated code will do exactly what the user-mode VAX code will do: take a privileged instruction fault.

User Action: If the instruction is not a real instruction, use an .HIF property ('location' dataentry) to direct VEST away from parsing the instruction.

If the instruction was encountered in the second VEST pass to identify code, either ignore this message or specify /OPTIMIZE=NOSCAN to bypass the second pass. If the instruction is real and INTENDED to fault at run time (some languages use HALT, BPT, or BUGCHK for this purpose), then no user action is needed; treat the message as informational. If the instruction is real and INTENDED to run in kernel mode, then the image cannot be translated. Use the MACRO-32 compiler or rewrite the source code.

PRIVSS, Image calls system service ‘service’. Execution of privileged code in a translated context is not supported.

Facility: VEST, VAX Environment Software Translator utility

Explanation: The image makes a call to the change mode service noted in the PRIVSS message. VEST can translate the image, but if the translated image calls the service and then attempts to execute privileged code, a fatal error occurs.

User Action: The user action depends on the image being translated.

READCF0, Routine reads the condition handler address from its call frame

Facility: VEST, VAX Environment Software Translator utility

Explanation: VEST encountered code that reads the VAX call frame. The equivalent translated code will depend on the emulated OpenVMS VAX environment at run time.

User Action: None.

READCF1, Routine reads the mask/PSW longword from its call frame

Facility: VEST, VAX Environment Software Translator utility

Explanation: VEST encountered a routine that reads the VAX call frame. The equivalent translated code will depend on the emulated OpenVMS VAX environment at run time.

User Action: None.

READCF2, Routine reads the saved AP from its call frame

Facility: VEST, VAX Environment Software Translator utility

Explanation: VEST encountered a routine that reads the VAX call frame. The equivalent translated code will depend on the emulated OpenVMS VAX environment at run time.

User Action: None.

READCF3, Routine reads the saved FP from its call frame

Facility: VEST, VAX Environment Software Translator utility

Explanation: VEST encountered a routine that reads the VAX call frame. The equivalent translated code will depend on the emulated OpenVMS VAX environment at run time.

User Action: None.

READCF4, Routine reads the return PC from its call frame

Facility: VEST, VAX Environment Software Translator utility

Explanation: VEST encountered a routine that reads the VAX call frame. The equivalent translated code will depend on the emulated OpenVMS VAX environment at run time. Because this routine reads the return PC, VEST explicitly constructs the VAX PC to ensure the same behavior as the original code. Reconstructing the PC has a slight effect on performance at run time because of the extra Alpha AXP coding it requires.

User Action: None.

READERR, Error reading 'filename'

Facility: VEST, VAX Environment Software Translator utility

Explanation: There was an error reading the file specified in the VEST command line. The file was the wrong format or there was a protection violation.

User Action: Enter the name of an image with the correct format and protection.

READIMAGE, Reading file 'filename'

Facility: VEST, VAX Environment Software Translator utility

Explanation: VEST issues this message to identify the image VEST is currently reading to determine the dependencies that image has on other images.

User Action: None.

READING, Reading file 'filename'

Facility: VEST, VAX Environment Software Translator utility

Explanation: VEST issues this message to identify the .IIF or .HIF file it is reading.

User Action: None.

READJSBRET, JSB routine reads its return PC from the stack

Facility: VEST, VAX Environment Software Translator utility

Explanation: VEST encountered a routine that reads the VAX call frame. The equivalent translated code will depend on the emulated OpenVMS VAX environment at run time. Because this routine reads the return PC, VEST explicitly constructs the VAX PC to ensure the same behavior as the original code. Reconstructing the PC has a slight effect on performance at run time because of the extra Alpha AXP coding it requires.

User Action: None.

REJHIF, Rejected HIF entry 'property' ('address') which conflicts with the image or a prior HIF entry

Facility: VEST, VAX Environment Software Translator utility

Explanation: In an .HIF file identified in a previous READIF message, VEST has encountered an entry that conflicts in some way with a previous entry. VEST ignores the conflicting entry.

User Action: Check the entry indicated and edit the .HIF file to correct the inconsistency if that proves to be necessary. You may choose to retranslate the image depending on the nature of the error you uncover.

RSTREJDST, Rejected DST entry point 'property' ('address')

Facility: VEST, VAX Environment Software Translator utility

Explanation: When the VEST command line includes the /RESTRICT qualifier, VEST issues this message for each .HIF entry that describes an excluded DST entry point. The /RESTRICT qualifier requests VEST to explicitly analyze specified parts of an image.

User Action: None.

RSTREJGST, Rejected GST entry point 'property' ('address')

Facility: VEST, VAX Environment Software Translator utility

Explanation: When the VEST command line includes the /RESTRICT qualifier, VEST issues this message for each .HIF entry that describes an excluded GST entry point. The /RESTRICT qualifier requests VEST to explicitly analyze specified parts of an image.

User Action: None.

RSTREJHIF, Rejected HIF entry point 'property' ('address')

Facility: VEST, VAX Environment Software Translator utility

Explanation: When the VEST command line includes the /RESTRICT qualifier, VEST issues this message for each .HIF entry that describes an excluded entry point. The /RESTRICT qualifier requests VEST to explicitly analyze specified parts of an image.

User Action: None.

RSTREJTAA, Rejected transfer address array entry point 'property' ('address')

Facility: VEST, VAX Environment Software Translator utility

Explanation: When the VEST command line includes the /RESTRICT qualifier, VEST issues this message for each .HIF entry that describes an excluded vector entry point. The /RESTRICT qualifier requests VEST to explicitly analyze specified parts of an image.

User Action: None.

RSTSELDST, Selected DST entry point 'property' ('address')

Facility: VEST, VAX Environment Software Translator utility

Explanation: When the VEST command line includes the /RESTRICT qualifier, VEST issues this message for each .HIF entry that describes a DST entry point within the part of the image selected to be analyzed. The /RESTRICT qualifier requests VEST to explicitly analyze specified parts of an image.

User Action: None.

RSTSELGST, Selected GST entry point 'property' ('address')

Facility: VEST, VAX Environment Software Translator utility

Explanation: When the VEST command line includes the /RESTRICT qualifier, VEST issues this message for each .HIF entry that describes a GST entry point within the part of the image selected to be analyzed. The /RESTRICT qualifier requests VEST to explicitly analyze specified parts of an image.

User Action: None.

RSTSELHIF, Selected HIF entry point 'property' ('address')

Facility: VEST, VAX Environment Software Translator utility

Explanation: When the VEST command line includes the /RESTRICT qualifier, VEST issues this message for each .HIF entry that describes an entry point within the part of the image selected to be analyzed. The /RESTRICT qualifier requests VEST to explicitly analyze specified parts of an image.

User Action: None.

RSTSELSCAN, Selected scan entry point 'property' ('address')

Facility: VEST, VAX Environment Software Translator utility

Explanation: When the VEST command line includes the /RESTRICT qualifier, VEST issues this message for each .HIF entry that describes a scan entry point within the part of the image selected to be analyzed. The /RESTRICT qualifier requests VEST to explicitly analyze specified parts of an image.

User Action: None.

RSTSELTAA, Selected transfer address array entry point 'property' ('address')

Facility: VEST, VAX Environment Software Translator utility

Explanation: When the VEST command line includes the /RESTRICT qualifier, VEST issues this message for each .HIF entry that describes a vector entry point within the part of the image selected to be analyzed. The /RESTRICT qualifier requests VEST to explicitly analyze specified parts of an image.

User Action: None.

RWINTERP, Code in writeable image section will be interpreted

Facility: VEST, VAX Environment Software Translator utility

Explanation: By default, VEST interprets code in writeable image sections (/INTERPRET=WRITEABLE_CODE).

User Action: None if you know that the image sections might be written. If the image sections are not written to, use the /INTERPRET=NOCODE qualifier to force VEST to translate rather than interpret code in writeable image sections.

RWTRANS, Code in writeable image section is being translated

Facility: VEST, VAX Environment Software Translator utility

Explanation: VEST has encountered code in a writeable image section, which it will translate (/INTERPRET=NOCODE).

User Action: To force VEST to interpret rather than translate such code, use the /INTERPRET=WRITEABLE_CODE qualifier.

RWTRANSDEF, Code in writeable image section is being translated by default-use /INTERPRET to override

Explanation: VEST found code in a writeable image section. By default, VEST translates such code. This is a WARNING message.

User Action: Determine whether the code should be interpreted or translated. To force code in writeable image sections to be interpreted at run time, retranslate the image with the qualifier /INTERPRET=WRITEABLE_CODE. If you want VEST to translate code in writeable image sections, retranslate the image with the qualifier /INTERPRET=NO_CODE.

SHAREABLE, Input is a sharable image-writing IIF file 'filename'

Facility: VEST, VAX Environment Software Translator utility

Explanation: This message specifies the name of the .IIF file VEST created for a shareable image.

User Action: Make the file 'filename' available to VEST whenever it translates an image that refers to the version of the shareable image just translated.

STKMISMATCH, Stack pointer adjustments don't match on all incoming paths-may indicate source bug

Facility: VEST, VAX Environment Software Translator utility

Explanation: VEST analysis indicates that the stack may be unbalanced at the specified location. The stack is unbalanced if different paths to the same code incur a different incremental change to the stack. One or more of the paths may never actually occur at run time. However, the imbalance may indicate source code problems.

User Action: If you believe the stack imbalance may indicate a problem in the source code, make a flowgraph of the affected part of the image. The flowgraph may help you isolate the problem. Otherwise you can ignore this message.

STKUNAL, Stack pointer is not longword aligned after changing by 'number' bytes-accessing stack will be slow

Facility: VEST, VAX Environment Software Translator utility

Explanation: VEST has detected a block of code that changes the stack from longword aligned to unaligned. This causes performance degradation on OpenVMS VAX systems, and will cause severe performance degradation of translated code.

User Action: Compaq recommends that the user rewrite the original OpenVMS VAX program to keep the stack longword aligned, then retranslate.

SUSPCASELIMIT, CASE instruction has a non-zero bound-may need to use HIF: 'address' caselimit 'limit'

Facility: VEST, VAX Environment Software Translator utility

Explanation: VEST has detected a CASE instruction that may not be a real CASE instruction.

User Action: Make an .HIF file entry for the image being translated at 'address' with the caselimit set to 'limit'.

SYNTAX, Syntax error in file 'filename': column 'column' in record 'record_text'

Facility: VEST, VAX Environment Software Translator utility

Explanation: VEST detected a syntax error at the file location identified in the message. Most likely, a hand edit to the file introduced a format error.

User Action: Identify the error in the information file indicated and correct it.

TRACEBACKON, /NOTRACEBACK was overridden because image was linked /DEBUG-add /NODEBUG to force /NOTRACEBACK

Facility: VEST, VAX Environment Software Translator utility

Explanation: If an image includes full debugging information in the debug symbol table (DST), you must include both the /NOTRACEBACK and the /NODEBUG qualifier in the VEST command line to create a translated image that omits debugging and traceback information. This message occurs if the VEST command line includes the /NOTRACEBACK qualifier but does not include the /NODEBUG qualifier.

User Action: Retranslate the image specifying both /NOTRACEBACK and /NODEBUG in the VEST command line if you want the translated image to omit traceback and debugging information.

TRANSERROR, Translation unsuccessful-no output image created

Facility: VEST, VAX Environment Software Translator utility

Explanation: VEST encountered one or more ERROR level error messages. VEST did not create a translated image. This message is the status code returned to DCL when VEST completes its run.

User Action: Review the ERROR level messages VEST issued to identify the problem.

TRANSFATAL, Translation was impossible

Facility: VEST, VAX Environment Software Translator utility

Explanation: VEST encountered one or more FATAL level error messages. VEST did not create a translated image. This message is the status code returned to DCL when VEST completes its run.

User Action: Review the FATAL level messages VEST issued to identify the problem.

TRANSOK, Translation completed successfully

Facility: VEST, VAX Environment Software Translator utility

Explanation: VEST successfully translated the image and issued INFO level error messages only. This message is the status code returned to DCL when VEST completes its run.

User Action: None.

TRANSWARN, Translation completed with warnings- review them before using the output image

Facility: VEST, VAX Environment Software Translator utility

Explanation: VEST has issued at least one WARNING level error message. A WARNING level message indicates that the translated image may have an error, possibly due to a problem in the original source code. This message is the status code returned to DCL when VEST completes its run.

User Action: Examine the WARNING level messages VEST has issued before using the translated image.

UNDSYM, Syntax error in file 'filename': undefined symbol 'symbol'

Facility: VEST, VAX Environment Software Translator utility

Explanation: A file offset has been expressed as a symbol, but the symbol is undefined.

User Action: Either edit the file to define the symbol before using it or use an absolute offset.

UNSUPABSREF, Reference to unsupported absolute address 'address' ('space' + 'offset')

Facility: VEST, VAX Environment Software Translator utility

Explanation: The image references an absolute address outside the supported range. The value 'space' is either P0_SPACE, P1_SPACE, or S0_SPACE, each of which represents a region of memory that an image cannot reference. In some cases, 'address' could be an OpenVMS VAX system service that is unsupported the OpenVMS Alpha operating system.

User Action: If possible, rewrite the program to avoid this unsupported reference.

VARCASELIMIT, CASE instruction has a variable limit operand- will be interpreted

Facility: VEST, VAX Environment Software Translator utility

Explanation: VEST has encountered a VAX CASE instruction with an unknown table size. The instruction is almost surely not a real VAX instruction. If 'location' is executed at run time, VAX code, including the CASE instruction and its target, will be interpreted.

User Action: If the instruction is not a real instruction, use an .HIF 'location' dataentry property to direct VEST not to parse the instruction.

If the instruction was encountered in the second VEST pass of identifying code, an alternative action is to specify /OPTIMIZE=NOSCAN to bypass the second pass.

VAXDFLOAT, VAX D_floating opcode 'opcode'-will be emulated (D56) or implemented via G_floating (D53)

Facility: VEST, VAX Environment Software Translator utility

Explanation: VEST has encountered a VAX D_floating instruction. D_floating is supported in one of two ways: by means of the G_floating hardware /FLOAT=D53_FLOAT or by means of software emulation (/FLOAT=D56_FLOAT). D53 is fast but loses 3 bits of precision. D56 is slightly slow but gives exact VAX results. Both are format compatible with existing VAX data structures and data files.

User Action: If this message occurs, Compaq recommends that you switch the original OpenVMS VAX program to use the preferred G_floating format, then retranslate. This is usually a matter of recompiling with /G_FLOAT specified. If changing the original OpenVMS VAX program to G_floating is not an option, then the user should choose between D53 and D56.

If the instruction is not a real instruction, use an .HIF property ('location' dataentry) property to direct VEST not to parse the instruction.

VAXHFLOAT, VAX H_floating opcode 'opcode'-will be emulated

Facility: VEST, VAX Environment Software Translator utility

Explanation: VEST has encountered a VAX H_floating instruction. H_floating is supported only by means of software emulation, just as on VAX 3000-, 4000-, and 6000-series machines. This is slightly slow but gives exact VAX results. It is format compatible with existing VAX data structures and data files.

User Action: If this message occurs, Compaq recommends that the user rewrite the original OpenVMS VAX program to use the preferred G_floating format, then retranslate. If changing the original OpenVMS VAX program to G_floating is not an option, then no user action is needed. If this message does not occur, the user need not be concerned with H_floating issues.

If the instruction is not a real instruction, use an .HIF property ('location' dataentry) to direct VEST away from parsing the instruction.

VAXPACKED, VAX packed decimal string opcode 'opcode'-will be emulated

Facility: VEST, VAX Environment Software Translator utility

Explanation: VEST has encountered a VAX packed- decimal instruction. Packed decimal is supported only via software emulation, just as on VAX 3000-, 4000-, and 6000-series machines. This is slightly slow but gives exact OpenVMS VAX results. It is format compatible with existing VAX data structures and data files.

User Action: If this message occurs, Compaq recommends that the user rewrite the original OpenVMS VAX program to use binary variables wherever possible, then retranslate. In VAX COBOL, this means declaring variables COMPUTATIONAL instead of DISPLAY or COMPUTATIONAL-3, and recompiling with the /INSTRUCTION_SET=NODECIMAL switch. If changing the original OpenVMS VAX program to use binary is not an option, then no user action is needed. If this message does not occur, the user need not be concerned with packed decimal issues.

If the instruction is not a real instruction, use an .HIF property ('location' dataentry) to direct VEST away from parsing the instruction.

VECTOROPC, Vector instruction opcode 'opcode'-will be interpreted and will fault if executed

Facility: VEST, VAX Environment Software Translator utility

Explanation: VEST has encountered a OpenVMS VAX vector instruction. Translation analysis continues, but no output image is written.

User Action: Vector system services are not supported at run time, so translating vectorized programs could only result in programs that do not execute. Instead, translate an existing nonvectorized form of the image, or recompile the program without specifying vectorization.

If the instruction is not a real instruction, use an .HIF property ('location' dataentry) to direct VEST away from parsing the instruction.

WRITECF0, Routine writes the condition handler address in its call frame

Facility: VEST, VAX Environment Software Translator utility

Explanation: VEST encountered code that writes to the VAX call frame. The equivalent translated code will depend on the emulated OpenVMS VAX environment at run time.

User Action: None.

WRITECF1, Routine writes the mask/PSW longword in its call frame

Facility: VEST, VAX Environment Software Translator utility

Explanation: VEST encountered code that writes to the VAX call frame. The equivalent translated code will depend on the emulated OpenVMS VAX environment at run time.

User Action: None.

WRITECF2, Routine writes the saved AP in its call frame

Facility: VEST, VAX Environment Software Translator utility

Explanation: VEST encountered code that writes to the VAX call frame. The equivalent translated code will depend on the emulated OpenVMS VAX environment at run time.

User Action: None.

WRITECF3, Routine writes the saved FP in its call frame

Facility: VEST, VAX Environment Software Translator utility

Explanation: VEST encountered code that writes to the VAX call frame. The equivalent translated code will depend on the emulated OpenVMS VAX environment at run time.

User Action: None.

WRITECF4, Routine writes the return PC in its call frame

Facility: VEST, VAX Environment Software Translator utility

Explanation: VEST encountered code that writes to the saved PC in the VAX call frame. The equivalent translated code will depend on the emulated OpenVMS VAX environment at run time. In this case, the code may be writing a VAX address into the return PC. VEST must generate code to map the VAX address back to the equivalent Alpha AXP address.

User Action: None.

WRITEERROR, Error writing file 'filename'

Facility: VEST, VAX Environment Software Translator utility

Explanation: While VEST was writing to a file, it encountered a write failure of some kind, for example, the disk became full or the disk quota was exceeded.

User Action: Check to determine the condition that caused the write failure. Fix the problem, if possible, and retranslate the image.

WRITEJSBRET, JSB routine writes the return PC on the stack

Facility: VEST, VAX Environment Software Translator utility

Explanation: VEST encountered code that writes to the VAX call frame. The equivalent translated code will depend on the emulated OpenVMS VAX environment at run time. In this case, the code may be writing a VAX address into the return PC. VEST must generate code to map the VAX address back to the equivalent Alpha AXP address.

User Action: None.

B.2.2 DSTGRAPH Messages

DSTCLSFAIL, Cannot close output file 'file'

Facility: DSTGRAPH, DSTgraph command

Explanation: DSTGRAPH was unable to close the output file.

User Action: Submit a Software Performance Report (SPR) describing the circumstances.

DSTCRTFAIL, Cannot create output file 'file'

Facility: DSTGRAPH, DSTgraph command

Explanation: DSTGRAPH was unable to open the output file.

User Action: Check to determine why DSTGRAPH could not open the file. (For example, the /OUTPUT qualifier specified a directory to which you do not have write access.) Reenter the DSTGRAPH command line specifying an output file that DSTGRAPH can open.

DSTINTERR, Internal Dstgraph error - submit SPR

Facility: DSTGRAPH, DSTgraph command

Explanation: DSTGRAPH has encountered a fatal internal error.

User Action: Submit a Software Performance Report (SPR) describing the circumstances.

DSTWRTFAIL, Cannot write to output file 'file'

Facility: DSTGRAPH, DSTgraph command

Explanation: While DSTGRAPH was writing to the output file, it encountered a write failure of some kind; for example, the disk became full or the disk quota was exceeded.

User Action: Check to determine the condition that caused the write failure. Fix the problem, if possible, and reenter the command.

INVDG, Dstgraph file 'file' is invalid or corrupted

Facility: DSTGRAPH, DSTgraph command

Explanation: The specified file either does not contain DSTGRAPH information or the DSTGRAPH file is corrupted.

User Action: Either specify a valid DSTGRAPH file or rerun VEST to create a new DSTGRAPH file.

INVSCALE, Invalid scale factor

Facility: DSTGRAPH, DSTgraph command

Explanation: The /SCALE_FACTOR qualifier specified an invalid number. The scale must be a positive decimal number.

User Action: Reenter the DSTGRAPH command and specify a valid scale factor.

INVWIDTH, Invalid width

Facility: DSTGRAPH, DSTgraph command

Explanation: The /WIDTH qualifier specified an invalid number. The number specified must be an integer that is a power of two and that represents a number of bytes.

User Action: Reenter the DSTGRAPH command and specify a valid integer.

NODG, Dstgraph file 'file' not found

Facility: DSTGRAPH, DSTgraph command

Explanation: DSTGRAPH could not open the file specified in the command line.

User Action: Reenter the command line with a valid file specification.

B.2.3 FLOWGRAPH Messages

CLSFAIL, Cannot close output file 'file'

Facility: FLOWGRAPH, Flowgraph command

Explanation: FLOWGRAPH was unable to close the output file.

User Action: Submit a Software Performance Report (SPR) describing the circumstances.

CRTFAIL, Cannot create output file 'file'

Facility: FLOWGRAPH, Flowgraph command

Explanation: FLOWGRAPH was unable to open the output file.

User Action: Check to determine why VEST could not open the file. (For example, the /OUTPUT qualifier specified a directory to which you do not have write access.) Reenter the VEST command line specifying an output file that FLOWGRAPH can open.

INTERR, Internal Flowgraph error - submit SPR

Facility: FLOWGRAPH, Flowgraph command

Explanation: FLOWGRAPH has encountered a fatal internal error.

User Action: Submit a Software Performance Report (SPR) describing the circumstances.

INVFG, Flowgraph file 'file' is invalid or corrupted

Facility: FLOWGRAPH, Flowgraph command

Explanation: The specified file either does not contain a flowgraph or the flowgraph file is corrupted.

User Action: Either specify a valid flowgraph file or rerun VEST to create a new flowgraph file.

`INVSCALE`, Invalid scale factor

Facility: FLOWGRAPH, Flowgraph command

Explanation: The /SCALE_FACTOR qualifier specified an invalid number. The scale must be a positive decimal number.

User Action: Reenter the FLOWGRAPH command and specify a valid scale factor.

`NOADDR`, Cannot find starting address

Facility: FLOWGRAPH, Flowgraph command

Explanation: The /STARTING_ADDRESS qualifier specified an address that FLOWGRAPH could not find in the .GRAPH file.

User Action: Check the .GRAPH file to determine a correct starting address to specify in the /STARTING_ADDRESS qualifier.

`NOFG`, Flowgraph file 'file' not found

Facility: FLOWGRAPH, Flowgraph command

Explanation: FLOWGRAPH could not open the flowgraph file specified in the command line.

User Action: Reenter the command line with a valid file specification.

`WRTFAIL`, Cannot write to output file 'file'

Facility: FLOWGRAPH, Flowgraph command

Explanation: While FLOWGRAPH was writing to the output file, it encountered a write failure of some kind; for example, the disk became full or the disk quota was exceeded.

User Action: Check to determine the condition that caused the write failure. Fix the problem, if possible, and reenter the command.

C. Debugging Translations

This appendix includes tables that describe translation problems and suggest solutions for debugging them. The tables provide information about the following kinds of problems:

For information about the following type of problem	**See**
Image runs slowly	Table C-1
Image executes improperly or returns incorrect results	Table C-2
Image crashes or exits with fatal messages or access violations	Table C-3
Exceptions never terminate	Table C-4

To prepare for debugging, gather resources to help you as follows:

Have the following documentation available for reference:

- Migrating to an OpenVMS Alpha System: Planning for Migration
- Migrating to an OpenVMS Alpha System: Recompiling and Relinking Applications

Have available all VEST output files for the application being translated.

If you're not familiar with the application being translated, find someone who is.

Know the answer to the following questions:

- Did the image execute correctly on OpenVMS VAX V5.4-3? If the answer is no, the image is not translatable.
- How were the images compiled and linked?
- What programming languages were used?
- What were the VEST messages reported during translation?

Use the VEST qualifier /WARNINGS=VERBOSE to get as much information from VEST as possible.

D. Translation and Performance Restrictions

This appendix discusses the following topics:

Topic	See
How to identify images with translation or performance restrictions	Section D.1
Images that cannot be translated	Section D.2
Images that translate with WARNING-level error messages	Section D.3
Images with compatibility problems not detectable during translation	Section D.4
Images with code that adversely affects translated image performance	Section D.5

D.1 Identifying Restrictions and Performance Issues

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

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

D.2 Untranslatable Images

Images described in Table D-1 incur FATAL or ERROR messages.

D.3 Images Translatable with Warnings

Images described in Table D-2 incur WARNING messages. Compaq recommends that you examine each WARNING message carefully. You may determine that the code flagged is not likely to interfere with the image running translated. In other cases, that may not be true and you may need to take steps to ensure that the translated version of the image executes properly on an OpenVMS Alpha system. You may need to make changes to the source code, if available, and then recompile and relink the image before translating. Or you may need to relink the image with different options. Or you may be able to eliminate the warning by using specific VEST switches when you translate the image.

Note that VEST can also issue WARNING messages for reasons other than those identified in Table D-2.

D.4 Images with Undetectable Translation Problems

The image features described in Table D-3 will not incur error messages but represent compatibility problems not detectable during translation.

Two of the three features relate to the increase in page size from OpenVMS VAX to OpenVMS Alpha systems. Page size can affect the amount of virtual memory allocated by memory management routines and system services. It is also the basis on which protection is assigned to code and data in memory. The OpenVMS VAX operating system allocates memory in multiples of 512 bytes, whereas OpenVMS Alpha pages range in size from 8KB to 64KB, depending on the specific hardware platform.

When VEST translates an image, it integrates multiple small VAX pages into a single 64KB Alpha AXP page. The 64KB page assumes the most permissive protection of all the VAX pages combined within it. In most cases, combining multiple VAX pages within a single Alpha AXP page does not affect the translated image's function. However, if your code has built-in dependencies on 512-byte granularity, the translated image may not run properly. For example, suppose your image depends on trapping attempts to write to a read- only section that has been combined into a read-write page in the translated image. When the translated image runs on an OpenVMS Alpha system, the trap will not occur. See the documentation noted in Table D-3 for information about handling such page dependencies.

D.5 Translatable Images with Performance Issues

Images described in Table D-4 are translatable but include code that affects translated image performance (for example, requires interpretation at run time). In addition to the pointers to performance problems that VEST offers, you can use TIE run-time statistics to discover exactly how the image uses resources that slow performance.

E. VAX Instructions

This appendix lists all the VAX instructions and describes how VEST or the Translated Image Environment (TIE) handles each one at translation or run time. The possible actions are as follows:

Translated by VEST

VEST translates the instruction into equivalent Alpha AXP code.

Executed in translated TIE complex instruction routine

Executed in native TIE complex instruction routine

Complex VAX instructions are too large and complex for VEST to generate in-line. Instead, the TIE either calls the native image TIE$SHARE, which executes a native routine that emulates the complex instruction, or calls the translated image TIE$EMULAT_TV, which runs a translated routine to execute the instruction. The speed of complex instruction routines depends on how complicated the instruction actually is and how much data is being operated upon.

Privileged. VEST generates a call to the interpreter. Faults if executed.

VEST does not translate privileged instructions. If the instruction occurs in code that executes at run time, a fault is generated.

Vector. VEST generates a call to the interpreter. Faults if executed.

VEST does not translate vector instructions. If the instruction occurs in code that executes at run time, a fault is generated.

The following table lists all the VAX instructions in alphabetical order and the associated action for each one:

Instruction	Action
ACBB	Translated by VEST
ACBD	Executed in translated TIE complex instruction routine
ACBF	Executed in translated TIE complex instruction routine
ACBG	Executed in translated TIE complex instruction routine
ACBH	Executed in translated TIE complex instruction routine
ACBL	Translated by VEST
ACBW	Translated by VEST
ADAWI	Translated by VEST
ADDB2	Translated by VEST
ADDB3	Translated by VEST
ADDD2	Executed in native TIE complex instruction routine.
ADDD3	Executed in native TIE complex instruction routine.
ADDF2	Executed in translated TIE complex instruction routine
ADDF3	Executed in translated TIE complex instruction routine
ADDG2	Translated by VEST
ADDG3	Translated by VEST
ADDH2	Executed in native TIE complex instruction routine.

Instruction	Action
ADDH3	Executed in native TIE complex instruction routine.
ADDL2	Translated by VEST
ADDL3	Translated by VEST
ADDP4	Executed in translated TIE complex instruction routine
ADDP6	Executed in translated TIE complex instruction routine
ADDW2	Translated by VEST
ADDW3	Translated by VEST
ADWC	Translated by VEST
AOBLEQ	Translated by VEST
AOBLSS	Translated by VEST
ASHL	Translated by VEST.
ASHP	Executed in translated TIE complex instruction routine
ASHQ	Translated by VEST.
BBC	Translated by VEST.
BBCC	Translated by VEST.
BBCCI	Translated by VEST.
BBCS	Translated by VEST.
BBS	Translated by VEST.
BBSC	Translated by VEST.
BBSS	Translated by VEST.
BBSSI	Translated by VEST.
BEQL	Translated by VEST.
BGEQU	Translated by VEST.
BGTR	Translated by VEST.
BGTRU	Translated by VEST.
BICB2	Translated by VEST.
BICB3	Translated by VEST.
BICL2	Translated by VEST.
BICL3	Translated by VEST.
BICPSW	Translated by VEST.
BICW2	Translated by VEST.
BICW3	Translated by VEST.
BISB2	Translated by VEST.
BISB3	Translated by VEST.
BISL2	Translated by VEST.
BISL3	Translated by VEST.

Instruction	Action
BISPSW	Translated by VEST.
BISW2	Translated by VEST.
BISW3	Translated by VEST.
BITB	Translated by VEST.
BITL	Translated by VEST.
BITW	Translated by VEST.
BLBC	Translated by VEST.
BLBS	Translated by VEST
BLEQ	Translated by VEST
BLEQU	Translated by VEST
BLSS	Translated by VEST
BLSSU	Translated by VEST
BNEQ	Translated by VEST
BPT	Translated by VEST
BRB	Translated by VEST
BRW	Translated by VEST
BSBB	Translated by VEST
BSBW	Translated by VEST
BUGL	Privileged VEST generates a call to the interpreter. Faults if executed
BUGW	Privileged VEST generates a call to the interpreter. Faults if executed
BVC	Translated by VEST
BVS	Translated by VEST
CALLG	Translated by VEST
CALLS	Translated by VEST
CASEB	Translated by VEST
CASEL	Translated by VEST
CASEW	Translated by VEST
CHME	Translated by VEST
CHMK	Translated by VEST
CHMS	Translated by VEST
CHMU	Translated by VEST
CLRB	Translated by VEST
CLRL	Translated by VEST
CLRO	Executed in translated TIE complex instruction routine
CLRQ	Translated by VEST
CLRW	Translated by VEST

Instruction	Action
CMPB	Translated by VEST
CMPC3	Executed in native TIE complex instruction routine
CMPC5	Executed in native TIE complex instruction routine
CMPD	Executed in native TIE complex instruction routine
CMPF	Executed in translated TIE complex instruction routine
CMPG	Translated by VEST
CMPH	Executed in native TIE complex instruction routine
CMPL	Translated by VEST
CMPP3	Executed in translated TIE complex instruction routine
CMPP4	Executed in translated TIE complex instruction routine
CMPV	Translated by VEST
CMPW	Translated by VEST
CMPZV	Translated by VEST
CRC	Executed in translated TIE complex instruction routine
CVTBD	Executed in native TIE complex instruction routine
CVTBF	Executed in translated TIE complex instruction routine
CVTBG	Translated by VEST
CVTBH	Executed in native TIE complex instruction routine
CVTBL	Translated by VEST
CVTBW	Translated by VEST
CVTDB	Executed in native TIE complex instruction routine
CVTDF	Executed in native TIE complex instruction routine
CVTDH	Executed in native TIE complex instruction routine
CVTDL	Executed in native TIE complex instruction routine
CVTDW	Executed in native TIE complex instruction routine
CVTFB	Executed in translated TIE complex instruction routine
CVTFD	Executed in native TIE complex instruction routine
CVTFG	Executed in translated TIE complex instruction routine
CVTFH	Executed in native TIE complex instruction routine
CVTFL	Executed in translated TIE complex instruction routine
CVTFW	Executed in translated TIE complex instruction routine
CVTGB	Translated by VEST
CVTGF	Translated by VEST
CVTGH	Executed in native TIE complex instruction routine
CVTGL	Translated by VEST
CVTGW	Translated by VEST

Instruction	Action
CVTHB	Executed in native TIE complex instruction routine
CVTHD	Executed in native TIE complex instruction routine
CVTHF	Executed in native TIE complex instruction routine
CVTHG	Executed in native TIE complex instruction routine
CVTHL	Executed in native TIE complex instruction routine
CVTHW	Executed in native TIE complex instruction routine
CVTLB	Translated by VEST
CVTLD	Executed in native TIE complex instruction routine
CVTLF	Executed in translated TIE complex instruction routine
CVTLG	Translated by VEST
CVTLH	Executed in native TIE complex instruction routine
CVTLP	Executed in translated TIE complex instruction routine
CVTLW	Translated by VEST
CVTPL	Executed in translated TIE complex instruction routine
CVTPS	Executed in translated TIE complex instruction routine
CVTPT	Executed in translated TIE complex instruction routine
CVTRDL	Executed in native TIE complex instruction routine
CVTRFL	Executed in translated TIE complex instruction routine
CVTRGL	Translated by VEST
CVTRHL	Executed in native TIE complex instruction routine
CVTSP	Executed in translated TIE complex instruction routine
CVTTP	Executed in translated TIE complex instruction routine
CVTWB	Translated by VEST
CVTWD	Executed in native TIE complex instruction routine
CVTWF	Executed in translated TIE complex instruction routine
CVTWG	Translated by VEST
CVTWH	Executed in native TIE complex instruction routine
CVTWL	Translated by VEST
DECB	Translated by VEST
DECL	Translated by VEST
DECW	Translated by VEST
DIVB2	Executed in native TIE complex instruction routine
DIVB3	Executed in native TIE complex instruction routine
DIVD2	Executed in native TIE complex instruction routine
DIVD3	Executed in native TIE complex instruction routine
DIVF2	Executed in translated TIE complex instruction routine

Instruction	Action
DIVF3	Executed in translated TIE complex instruction routine
DIVG2	Translated by VEST
DIVG3	Translated by VEST
DIVH2	Executed in native TIE complex instruction routine
DIVH3	Executed in native TIE complex instruction routine
DIVL2	Executed in native TIE complex instruction routine
DIVL3	Executed in native TIE complex instruction routine
DIVP	Executed in translated TIE complex instruction routine
DIVW2	Executed in native TIE complex instruction routine
DIVW3	Executed in native TIE complex instruction routine
EDITPC	Executed in translated TIE complex instruction routine
EDIV	Executed in native TIE complex instruction routine
EMODD	Executed in translated TIE complex instruction routine
EMODF	Executed in translated TIE complex instruction routine
EMODG	Executed in translated TIE complex instruction routine
EMODH	Executed in translated TIE complex instruction routine
EMUL	Translated by VEST
EXTV	Translated by VEST
EXTZV	Translated by VEST
FFC	Translated by VEST
FFS	Translated by VEST
HALT	Privileged VEST generates a call to the interpreter. Faults if executed
INCB	Translated by VEST
INCL	Translated by VEST
INCW	Translated by VEST
INDEX	Translated by VEST
INSQHI	Translated by VEST
INSQTI	Translated by VEST
INSQUE	Translated by VEST
INSV	Translated by VEST
JMP	Translated by VEST
JSB	Translated by VEST
LDPCTX	Privileged VEST generates a call to the interpreter. Faults if executed
LOCC	Executed in native TIE complex instruction routine
MATCHC	Executed in translated TIE complex instruction routine
MCOMB	Translated by VEST

Instruction	Action
MCOML	Translated by VEST
MCOMW	Translated by VEST
MFPR	Privileged VEST generates a call to the interpreter. Faults if executed
MFVP	Vector VEST generates a call to the interpreter. Faults if executed
MNEGB	Translated by VEST
MNEGD	Executed in native TIE complex instruction routine
MNEGF	Executed in translated TIE complex instruction routine
MNEGG	Translated by VEST
MNEGH	Executed in native TIE complex instruction routine
MNEGL	Translated by VEST
MNEGW	Translated by VEST
MOVAB	Translated by VEST
MOVAL	Translated by VEST
MOVAO	Executed in translated TIE complex instruction routine
MOVAQ	Translated by VEST
MOVAW	Translated by VEST
MOVB	Translated by VEST
MOVC3	Executed in native TIE complex instruction routine
MOVC5	Executed in native TIE complex instruction routine
MOVD	Executed in native TIE complex instruction routine
MOVF	Executed in translated TIE complex instruction routine
MOVG	Translated by VEST
MOVH	Executed in native TIE complex instruction routine
MOVL	Translated by VEST
MOVO	Executed in translated TIE complex instruction routine
MOVP	Executed in translated TIE complex instruction routine
MOVPSL	Translated by VEST
MOVQ	Translated by VEST
MOVTC	Executed in translated TIE complex instruction routine
MOVTUC	Executed in translated TIE complex instruction routine
MOVW	Translated by VEST
MOVZBL	Translated by VEST
MOVZBW	Translated by VEST
MOVZWL	Translated by VEST
MTPR	Privileged VEST generates a call to the interpreter. Faults if executed
MULB2	Translated by VEST

Instruction	Action
MULB3	Translated by VEST
MULD2	Executed in native TIE complex instruction routine
MULD3	Executed in native TIE complex instruction routine
MULF2	Executed in translated TIE complex instruction routine
MULF3	Executed in translated TIE complex instruction routine
MULG2	Translated by VEST
MULG3	Translated by VEST
MULH2	Executed in native TIE complex instruction routine
MULH3	Executed in native TIE complex instruction routine
MULL2	Translated by VEST
MULL3	Translated by VEST
MULP	Executed in translated TIE complex instruction routine
MULW2	Translated by VEST
MULW3	Translated by VEST
NOP	Translated by VEST
POLYD	Executed in translated TIE complex instruction routine
POLYF	Executed in translated TIE complex instruction routine
POLYG	Executed in translated TIE complex instruction routine
POLYH	Executed in translated TIE complex instruction routine
POPR	Executed in native TIE complex instruction routine
PROBER	Translated by VEST
PROBEW	Translated by VEST
PUSHAB	Translated by VEST
PUSHAL	Translated by VEST
PUSHAO	Executed in translated TIE complex instruction routine
PUSHAQ	Translated by VEST
PUSHAW	Translated by VEST
PUSHL	Translated by VEST
PUSHR	Executed in native TIE complex instruction routine
REI	Executed in native TIE complex instruction routine
REMQHI	Translated by VEST
REMQTI	Translated by VEST
REMQUE	Translated by VEST
RET	Executed in native TIE complex instruction routine
ROTL	Translated by VEST
RSB	Translated by VEST

Instruction	Action
SBWC	Translated by VEST
SCANC	Executed in native TIE complex instruction routine
SKPC	Executed in native TIE complex instruction routine
SOBGEQ	Translated by VEST
SOBGTR	Translated by VEST
SPANC	Executed in native TIE complex instruction routine
SUBB2	Translated by VEST
SUBB3	Translated by VEST
SUBD2	Executed in native TIE complex instruction routine
SUBD3	Executed in native TIE complex instruction routine
SUBF2	Executed in translated TIE complex instruction routine
SUBF3	Executed in translated TIE complex instruction routine
SUBG2	Translated by VEST
SUBG3	Translated by VEST
SUBH2	Executed in native TIE complex instruction routine
SUBH3	Executed in native TIE complex instruction routine
SUBL2	Translated by VEST
SUBL3	Translated by VEST
SUBP4	Executed in translated TIE complex instruction routine
SUBP6	Executed in translated TIE complex instruction routine
SUBW2	Translated by VEST
SUBW3	Translated by VEST
SVPCTX	Privileged VEST generates a call to the interpreter. Faults if executed
TSTB	Translated by VEST
TSTD	Executed in native TIE complex instruction routine
TSTF	Executed in translated TIE complex instruction routine
TSTG	Translated by VEST
TSTH	Executed in native TIE complex instruction routine
TSTL	Translated by VEST
TSTW	Translated by VEST
VGATHL	Vector VEST generates a call to the interpreter. Faults if executed
VGATHQ	Vector VEST generates a call to the interpreter. Faults if executed
VLDL	Vector VEST generates a call to the interpreter. Faults if executed
VLDQ	Vector VEST generates a call to the interpreter. Faults if executed
VSADDD	Vector VEST generates a call to the interpreter. Faults if executed
VSADDF	Vector VEST generates a call to the interpreter. Faults if executed

Instruction	Action
VSADDG	Vector VEST generates a call to the interpreter. Faults if executed
VSADDL	Vector VEST generates a call to the interpreter. Faults if executed
VSMERGE	Vector VEST generates a call to the interpreter. Faults if executed
VSSUBD	Vector VEST generates a call to the interpreter. Faults if executed
VSSUBF	Vector VEST generates a call to the interpreter. Faults if executed
VSSUBG	Vector VEST generates a call to the interpreter. Faults if executed
VSSUBL	Vector VEST generates a call to the interpreter. Faults if executed
VVADDD	Vector VEST generates a call to the interpreter. Faults if executed
VVADDF	Vector VEST generates a call to the interpreter. Faults if executed
VVADDG	Vector VEST generates a call to the interpreter. Faults if executed
VVADDL	Vector VEST generates a call to the interpreter. Faults if executed
VVMERGE	Vector VEST generates a call to the interpreter. Faults if executed
VVSUBD	Vector VEST generates a call to the interpreter. Faults if executed
VVSUBF	Vector VEST generates a call to the interpreter. Faults if executed
VVSUBG	Vector VEST generates a call to the interpreter. Faults if executed
VVSUBL	Vector VEST generates a call to the interpreter. Faults if executed
XORB2	Translated by VEST
XORB3	Translated by VEST
XORL2	Translated by VEST
XORL3	Translated by VEST
XORW2	Translated by VEST
XORW3	Translated by VEST

Examples

Example 2-1: Translating an Image

```
$ directory/brief 1
Directory VST_00:[VEST.TEST]
DHRYSTONE.EXE;1
Total of 1 file.
$ vest dhrystone 2
$ directory/brief
Directory VST_00:[VEST.TEST]      3
DHRYSTONE.EXE;1    DHRYSTONE_TV.EXE;1 DHRYSTONE_TV.LIS;1
Total of 3 files.
$ type dhrystone_tv.lis 4
VEST V1.1-25 built at Apr 19 1993 13:38:36 starting at
May 17 1993 13:43:44 with command line:
VEST DHRYSTONE
 Image "DHRYSTONE_SHR", "V1.0", 13-DEC-1989 10:17:07.95
! Message summary by category:    5
!
! 2 messages in SOURCE_ANALYSIS category:
!   2       INFO NONSTDCALLU - Non-standard call uses !AZ
!
! 5 messages in VERBOSE category:
!   1       INFO    READING - Reading file !AZ
!   1       INFO   NOHIF - HIF file !AZ not found
!   1       INFO   PASS1 - Starting analysis pass 1
!   1       INFO   PASS2 - Starting analysis pass 2
!   1       INFO ENDPASS2 - Ending analysis pass 2 -- beginning
code generation and output
```

Example 4-1: Audit Information for SIEVE.EXE

```
VEST V1.1-25 built at Apr 19 1993 13:38:36 starting at
May 17 1993 20:55:13 with command line:
VEST/AUDIT SIEVE
 Image "SIEVE", "V1.0", 14-OCT-1991 14:20:19.13
! Message summary by category:
!
! 5 messages in VERBOSE category:
!   1       INFO    READING - Reading file !AZ
!   1       INFO   NOHIF - HIF file !AZ not found
!   1       INFO   PASS1 - Starting analysis pass 1
!   1       INFO   PASS2 - Starting analysis pass 2
!   1       INFO ENDPASS2 - Ending analysis pass 2
                         -- beginning code generation and output
<SUM> Image name                              Comp Tran Perf Languages
<SUM> ------------------------------------ ---- ---- ---- ----------------
<SUM> VST_00:[VEST.TEST]SIEVE.EXE;                     YES YES OK C
```

Example 4-2: Run-Time Statistics for SIEVE_TV.EXE

```
$ define tie$display_statistics true
$ run sieve_tv
Sieve of Eratosthenes
500 iterations
1899 primes found
time taken : 1 seconds
TIE Run-time Statistics:
                   TIE Lookups:          CALLx  JSB     JMP     1
 ============================================================================
    Went to VAX routines:              0       3       3
 Stayed in Translated routines:                48     19       0
    Went to Native routines:                   71      N.A.    N.A.
 ----------------------------------------------------------------------------
                          Total:         119    22      3
 Entries in the lookup cache were found 57% of the time.
 There were no calls to the interpreter for VAX code located outside
2
 translated images.
 There were no Fault-On-Execute conditions converted to Lookups.
3
 The VAX-Instruction Atomicity Controller was never used.                     4
 There were 187 TIE-based ''complex instructions'' executed:                  5
Instruction REI   (02) : 3
Instruction RET   (04) : 101
Instruction MOVC3  (28) : 24
Instruction SPANC  (2B) : 3
Instruction MOVC5  (2C) : 22
Instruction LOCC   (3A) : 29
Instruction EDIV   (7B) : 5
 There were 6 VAX instructions interpreted.           6
 CPU time used:    0 00:00:01.26         7
 Autojacketing Statistics:               8
 There were 2 calls from native to translated routines.
 There were 71 calls from translated to native routines.
 5 translated images were used:          9
SIEVE_TV created by VEST V1.1-25 of Apr 19 1993 13:38:36
VAXCRTL_D56_TV created by VEST V1.1-25 of Apr 19 1993 13:38:36
MTHRTL_D53_TV created by VEST V1.1-25 of Apr 19 1993 13:38:36
LIBRTL_D56_TV created by VEST V1.1-25 of Apr 19 1993 13:38:36
 10 native images were used:
TIE$SHARE
DECC$SHR
DPML$SHR
LIBRTL
LIBOTS
SYS$BASE_IMAGE
SYS$PUBLIC_VECTORS
TIE$MESSAGES
DECC$MSG
SHRIMGMSG
$ deassign tie$display_statistics
```

Example 4-3: Requesting and Processing VEST Flowgraphs

```
$ vest/flowgraph/view=all/noexe sieve
$ flowgraph sieve
GRAPH 1:   4 nodes        7 arcs ACYCLIC 1 page written.
GRAPH 2:   0 nodes        0 arcs ACYCLIC 0 pages written.
GRAPH 3: 27 nodes 32 arcs 2 pages written.
$ print/queu=ps_queue/param=data=postscript sieve.ps
```

Example 5-1: Excerpt from a Run-Time Library .IIF File

```
; IIF Generated by VEST (T1.0-24 Jun 23 1992 13:10:29)
    Image "LIBRTL", "V05-001"
+00000430         usv_offset     +0290
+00000430         sets     "R0 M.sp"
+00000430         uses     "R1 AP FP SP PC RET M.sp M.unk"
+00000430         callentry              "LIB$INT_OVER"
+00000438         usv_offset     +02A0
+00000438         sets     ""
+00000438         uses     "R0 R1 AP FP SP PC RET M.pc M.sp M.unk"
+00000438         callentry              "LIB$LOCC"
+00000420         usv_offset     +0270
.
```

Example A-1: Summary Format

```
<SUM> Image name                                                Comp Tran Perf
Languages
<SUM> ----------------------------.....------------- ---- ---- ---- -----------
<SUM> VST_00:[GROUP.TEST]FLOWGRAPH.EXE;1                        YES YES OK PASCAL
```

Example C-1: Forcing Exception PC Correlation

```
Code sample showing PC correlation
```

```
   .sbttl pli$nonloc_goto - non-local goto processing
;
; check for proper frame
;
20$:       cmpl   r1,stk_l_fp(r0)              ; is the next frame the last?
    beql   40$                         ; if eql then yes
    movab w^norm_signal,-(sp)     ;address normal signal ret addr
    cmpl   stk_l_pc(r0),(sp)+    ;does this frame point there?
    beql   25$                         ;if eql, yes, special case
    movab w^error_signal,-(sp)    ;address error signal ret addr
    cmpl   stk_l_pc(r0),(sp)+    ;does this frame point there?
    beql   25$                         ;if eql, yes, special case
    movab b^canned_return,stk_l_pc(r0); force return in this module
    brb    27$                         ;cont
25$:       movab b^unwind_signal,stk_l_pc(r0); force return for signal frame
27$:       movl   stk_l_fp(r0),r0              ; link to next frame on stack
    probew #3,#stk_l_pc,(r0)           ; frame accessible?
    beql   fatal_error                 ; if eql then insuff frames
    brb    20$                         ; continue search
    .
    .
    .
unwind_signal:
    nop                                ;different than canned_return;
canned_return:
    tstl   stk_l_cnd_hnd(fp)           ; frame have condition handler?
    beql   100$                        ; if eql then no
```
Code sample from PLI Conditional Handler Dispatcher
```
;
; call with r1 as establisher's frame pointer
;
30$:       clrl   (fp)                         ; disable fatal exception vector
    .
    .
    .
    callg (ap),@cnd_l_addr(r3)    ; call handler with our arg list
norm_signal:                                   ; condition handler stack key
    movab w^fatal_exception,(fp) ; setup a condition handler
    .
    .
    .
    rsb                                ; and do so
```

Figures

Figure 1-1: VEST Processing

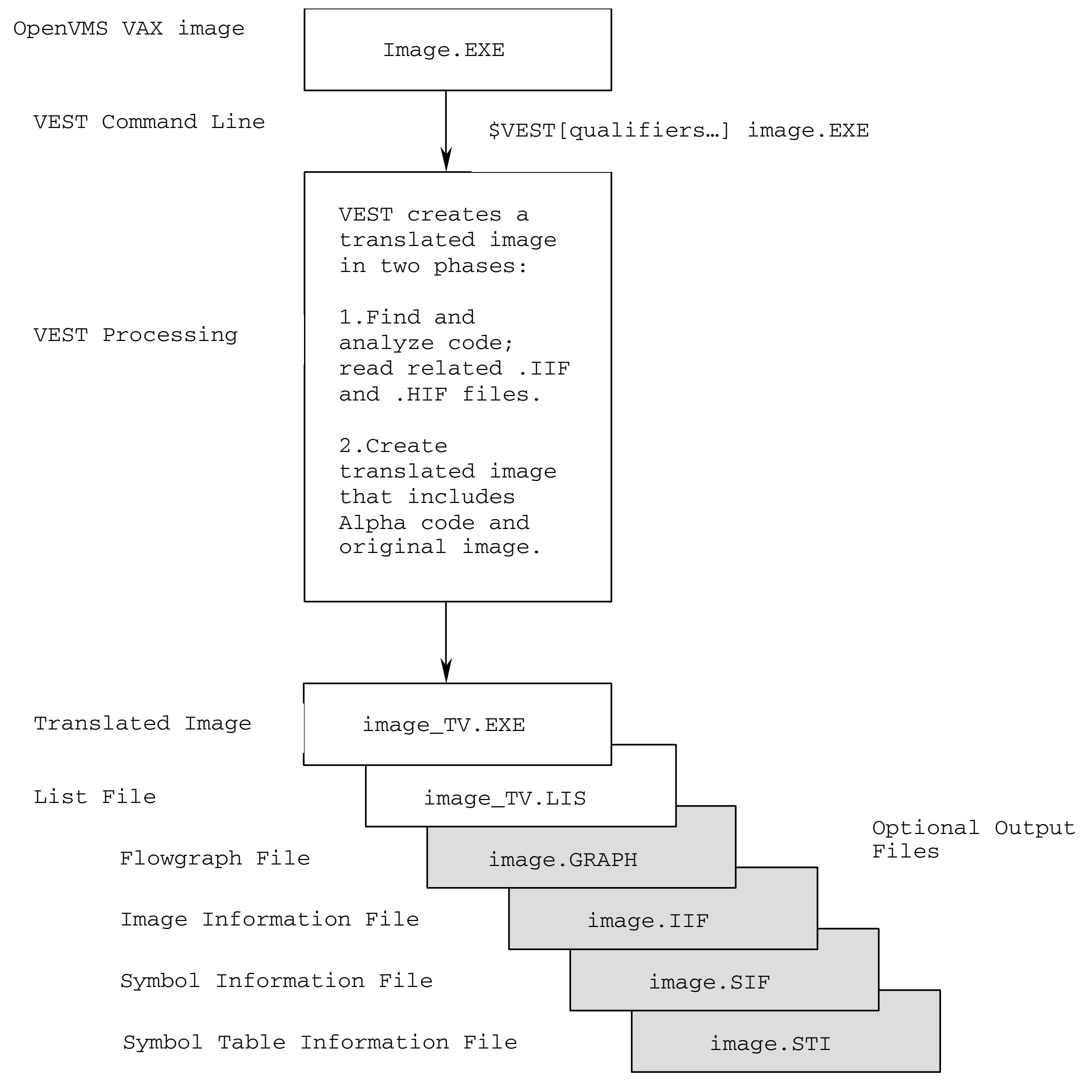

OpenVMS VAX image
Image.EXE
VEST Command Line
$VEST[qualifiers…] image.EXE
VEST Processing
VEST creates a translated image in two phases:
1.Find and analyze code; read related .IIF and .HIF files.
2.Create translated image that includes Alpha code and original image.
Translated Image
image_TV.EXE
List File
image_TV.LIS
Flowgraph File
image.GRAPH
Image Information File
image.IIF
Symbol Information File
image.SIF
Symbol Table Information File
image.STI
Optional Output Files

Figure 1-2: Translated Image Environment

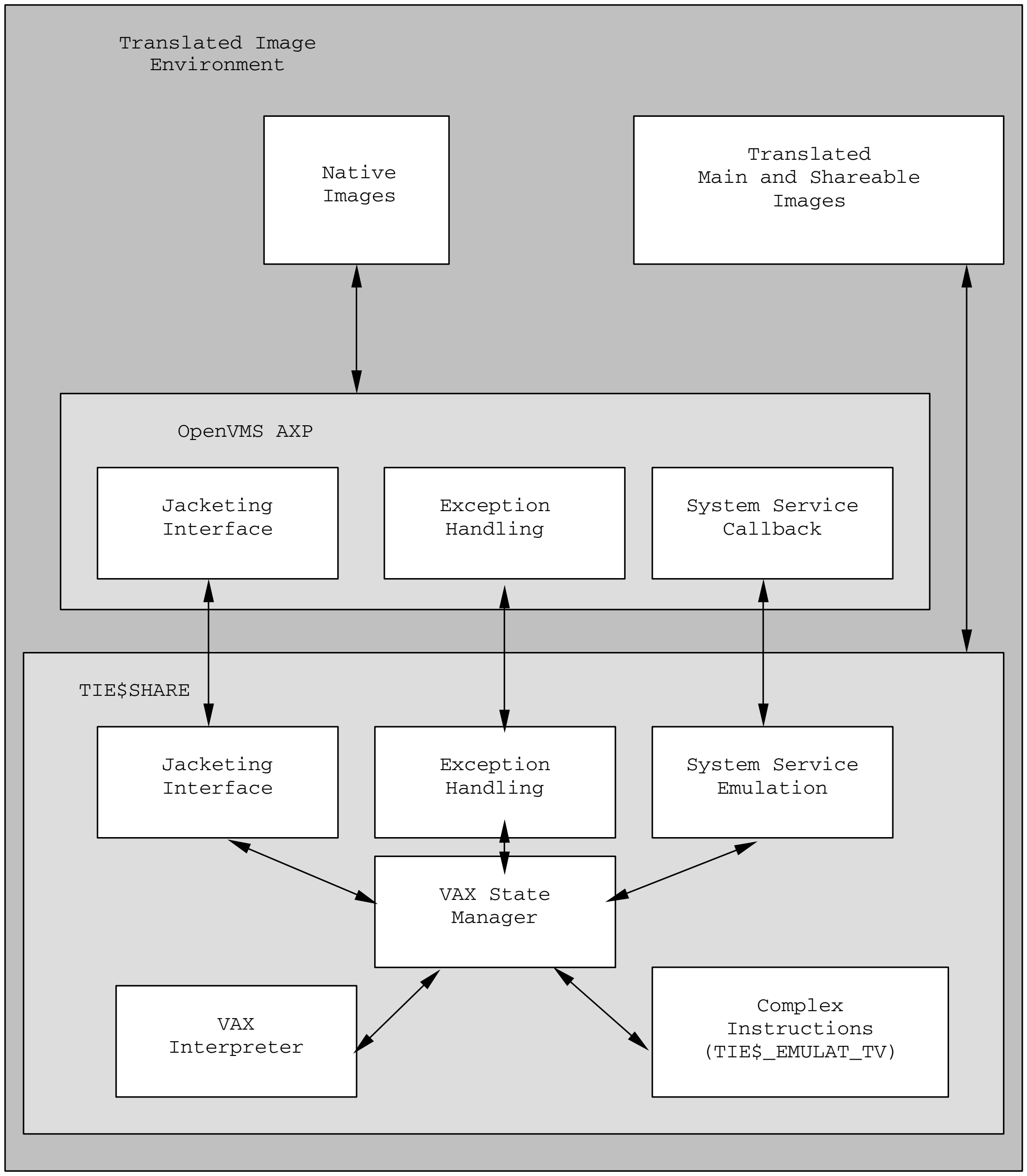
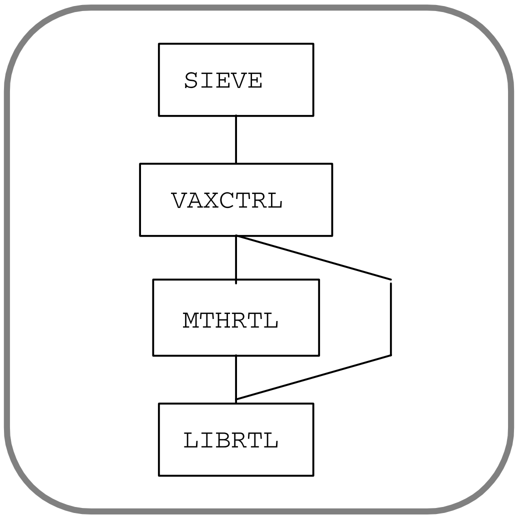


Figure 2-1: SIEVE.EXE Dependency Graph

```
 Image Dependency Graph for AXP00[GROUP.TEST]SIEVE.EXE;
```



```
.
.
.
```

Figure 4-1: The SIEVE Call Flowgraph

```
CALL graph: SIEVE V1.0
```

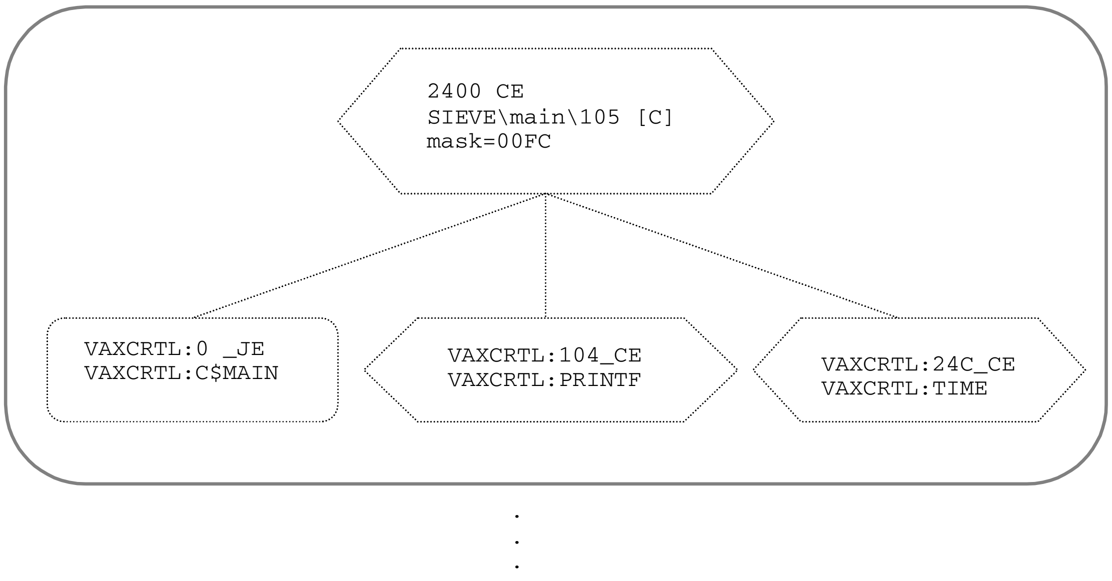


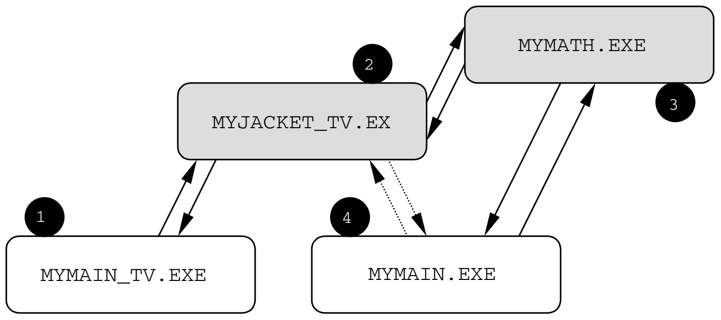
Figure 4-2: DHRYSTONE Flowgraph at 0.33 Scale

Figure 4-3: Example Subroutine in DHRYSTONE Flowgraph at 1.0 Scale

Figure 4-4: Error Graph for DHRYSTONE Basic Block 4CD0_CE

Figure 4-5: DSTGRAPH File Showing Unaligned Data

Figure 6-1: Role of Jacket Image



MYMATH.EXE
The native shareable image

MYJACKET_TV.EXE
The translated jacket image

MYMAIN_TV.EXE
A translated main program

MYMAIN.EXE
A native main program

The jacket image MYJACKET_TV intercepts all calls 1 from translated images to the shareable image MYMATH_TV and either services the calls itself or forwards 2 them to MYMATH. Native main image calls go directly 3 to MYMATH, except for calls to MYJACKET_TV 4 that have no equivalent in the native version of the shareable image.

Tables

Table 2-1: VEST Command Qualifiers

Category	**Qualifiers**
List file and message displays	/LIST
	/SHOW
	/WARNINGS
Output files	/EXECUTABLE
	/FLOWGRAPH
	/IIF
	/LIST
	/SIF
Performance considerations	/FEEDBACK
	/FLOAT
	/INTERPRET
	/OPTIMIZE
	/PRESERVE
Auditing and flowgraph	/AUDIT
	/DST
	/FLOWGRAPH
	/RESTRICT
	/VIEW
Shareable libraries	/IIF
	/JACKET
	/SIF
Information files	/FEEDBACK
	/IIF
	/INCLUDE_DIRECTORY
	/SIF

Table 2-2: VEST Output Files

Default name	Qualifier(s)	Description
image_TV.EXE	/EXECUTABLE[=filespec] Default: /EXECUTABLE	The translated image. VEST truncates any input image file name that exceeds 36 characters in order to append "_TV" and not exceed the 39-character limit on file names.
image_TV.LIS	/LIST[= filespec]/SHOW Default: /LIST /SHOW=MESSAGES	The list file. The /SHOW qualifier determines what the list file contains. By default it includes the VEST messages.
image .IIF	/IIF[= filespec] Default: /IIF for a shareable image; ignored for a main image	] The image information file (.IIF file), which describes a shareable image's entry points. VEST ignores the /IIF qualifier when translating an executable image. See Section 5.2 for details.
image .GRAPH	/FLOWGRAPH /VIEW=ERROR Default: /NOFLOWGRAPH	The flowgraph file, which contains data about the input image's program flow. The /VIEW qualifier determines the kind of data included in the file. The FLOWGRAPH command uses this data to create PostScript formatted diagrams of the image. See Section 4.3 for details.
image .SIF	/SIF[=filespec] Default: /NOSIF	The symbol information file (.SIF file), which describes the contents of the global symbol table (GST) and symbol vector in a translated shareable image. See Section 6.4 for details.
image .STI	/DST Default: /NODST	The symbol table information file (.STI) file, which describes the format of all memory references in the debugger symbol table (DST) of the input image. See Section 4.4.

Table 4-1: VEST Performance Qualifiers

Qualifier	Description
/FEEDBACK	Translate an image with /FEEDBACK enabled so that the TIE can record any entry points it finds when it interprets code. The TIE writes the information to an .HIF file. When you retranslate the image, VEST uses the .HIF file to locate and translate the code that was previously interpreted. (See Section 4.2.3.)
/FLOAT	Use this qualifier to choose between D53 and D56 floating point operations in the translated image. D53 is faster because it uses hardware emulation; D56 is slower because it uses software emulation.
/INTERPRET	Use this qualifier to control how much code to interpret at run time: all, none, or code in writeable image sections.
/OPTIMIZE	Use this qualifier to choose from a variety of code optimizations pertaining to data alignment, pass 2 of a VEST analysis, and instruction scheduling. In general, the more optimizations you enable, the better the performance.
/PRESERVE	Use this qualifier to select VAX characteristics you need to preserve exactly. These characteristics include condition codes, floating point exceptions, instruction atomicity, integer exceptions, memory atomicity, and read/write ordering. The more you choose to preserve, the slower the translated software runs.

Table 5-1: Interface Properties

Property Name	Property Value	Description
absolute_ok	None	Specifies OpenVMS Alpha support for the entry point; present only in P1_SPACE.IIF and S0_SPACE.IIF files; is always attached to the same offset as a callentry, jsbentry, or branchentry.
astback	p n	Specifies the parameter of a called routine that is itself a callentry called as an AST routine (p2, for example); is always attached to the same offset as a callentry, jsbentry, or branchentry. n is a decimal number.
branchentry	None	Defines a branchentry; indicates the start of a basic block of code.
callback	p n	Specifies the parameter of a called routine that is itself a callentry; is always attached to the same offset as a callentry, jsbentry, or branchentry. n is a decimal number.
callentry	[symbolic_name]	Defines a callentry and an optional symbolic name for the image offset.
caselimit[2]	+ integer	Specifies maximum number of cases, where integer is a hexadecimal number.
dataentry[2]	[+byte_count]	Defines the image region as data, not code; if specified, +byte_count determines the number of bytes from the associated offset to be considered data. If the image region actually is code, this property forces VEST to interpret it. byte_count is a hexadecimal number.
dataentry	[symbolic_name]	Within a .IIF file only, defines a dataentry, or data cell, and an optional symbolic name for the image offset.
delta_sp	+/-integer	Specifies a change in the stack pointer by the routine; is always attached to the same offset as a callentry, jsbentry, or branchentry.
dv_set_to[1]	p n	Upon return from the routine at the specified address, the VAX decimal overflow trap enable bit (DV) in the processor status word (PSW) has been set to the value of the routine parameter p n.
external[2]	None	Specifies an entry point to be exported.
flagif	message_id	Causes VEST to issue a message of the type specified if a (FLAGAST, for call is made to an entry with one of these properties. example)

Property Name	Property Value	Description
fu_set_to[1]	p n	Upon return from the routine at the specified address, the VAX floating underflow trap enable bit (FU) in the PSW has been set to the value of routine parameter p n.
iv_set_to[1]	p n	Upon return from the routine at the specified address, the VAX integer overflow trap enable bit (IV) in the PSW has been set to the value of the routine parameter p n .
jmpback	p n	Specifies the parameter of a called routine that is a branchentry; is always attached to the same basic block as the call or jsb placeholder. Note that the code jumped back to must execute in the same context as the calling basic block. Use the unkctxt_jmpback property if the code jumped to modifies any registers, thus changing the context.
jsbentry	[symbolic_name]	Defines a JSB entry point and an optional name for the image offset.
noreturn	None	Specifies a routine that does not return; is always attached to the same offset as a callentry or jsbentry
sets	list of resources in the form "R0 R1..." [3]	Specifies resources set by routine
+sets	list of resources in the form "R0 R1..." [3]	Adds resources to those determined by VEST
-sets	list of resources in the form "R0 R1..." [3]	Removes resources from those determined by VEST.
symbol	symbolic_name	Defines a symbolic name for the image offset.
unkctxt_jmpback	p n	Specifies the parameter of a called routine that is a branchentry. To be used when the called routine modifies any registers, thus changing the context. In contrast to the jmpback property, VEST makes no assumptions about the CALL frame or register contents when the called routine jumps to p n and does not connect the jumped to code to the calling basic block.
uses	list of resources in the form "R0 R1..." [3]	Specifies resources used by routine.
+uses[2]	list of resources in the form "R0 R1..." [3]	Adds resources to those determined by VEST.

Property Name	Property Value	Description
-uses[2]	list of resources in the form "R0 R1..." [3]	Removes resources from those determined by VEST.
usv_offset	+ integer	Specifies usv offset for global symbols in image; integer is a hexadecimal value.

[1] See Section 5.4.6 for information about using this property name.

[2] For use in .HIF files only.

[3] See Section 5.4.5 for a list of the resources you can specify.

Table 6-1: .SIF Directive Syntax

Field	Meaning
sym_name	The case-sensitive name of the symbol.
sym_value	The OpenVMS VAX image offset in hexadecimal associated with sym_name for relocatable symbols or the value of an absolute symbol. You can obtain this value from the output of the DCL command ANALYZE/IMAGE for the OpenVMS VAX image.
sv_flag	Either +S to add sym_name to the symbol vector or -S to exclude sym_name from the symbol vector
gst_flag	Either +G to add sym_name to the GST or -G to exclude sym_name from the GST
usv	The offset in hexadecimal from the start of the symbol vector at which to place sym_name ; if usv equals 0xFFFFFFFF, add sym_name to the end of the symbol vector after all other symbols have been preallocated. Never specify usv 0, which is reserved for VEST use. Note that usv must be a multiple of 16. Ignored if -S is used.
sym_type	The data type in hexadecimal VEST should specify in the GST entry if gst_flag equals +G. The data types are defined in the VAX Procedure Calling and Condition Handling Standard in the VAX Architecture Guide , Appendix C. Ignored if -G is used.
sym_flags	A hexadecimal number specifying the flags in the GST entry if gst_flag equals +G. The flags are defined according to the characteristics of the symbol as outlined in the chapter on VAX Object Language in the OpenVMS Linker Utility Manual . Ignored if -G is used.

Table A-1: VEST Message Categories

Performance	Source Analysis	Synchronization	Verbose
STKUNAL	NONSTDCALLS	FLAGAST	ENDPASS2
VAXDFLOAT	NONSTDCALLU	FLAGASYNC	NOHIF
VAXHFLOAT	READCF0	FLAGIO	PASS1
VAXPACKED	READCF1	FLAGMP	PASS2
	READCF2		READING
	READCF3		RWINTERP
	READCF4		RWTRANS
	READJSBRET		
	STKMISMATCH		
	WRITECF0		

Table C-1: Problem: Runs Slowly

Symptom	Action	Further information
Translated image runs, but slowly.	Define the logical names TIE$FORCE_FEEDBACK and TIE$DISPLAY_STATISTICS, which request the TIE to provide information about translated image execution. Issue the following command to request .HIF feedback files: $ DEFINE TIE$FORCE_FEEDBACK "T" Issue the following command to request the TIE run-time statistics: $ DEFINE TIE$FORCE_FEEDBACK "T" Rerun the translated image to obtain a .HIF feedback file and run-time statistics.	A translated image should run at approximately the same speed as the original image running on a comparable OpenVMS VAX system. Defining the logical name TIE$FORCE_FEEDBACK forces the TIE to use a .HIF file to record points found when interpreting See Section 4.2.3 for detailed information about .HIF files and run-time feedback. Defining the logical name TIE$DISPLAY_STATISTICS causes the TIE to display statistics about all the translated images activated during program execution. The statistics may help you figure out why your image is running slowly. See Section 4.2.2 for a detailed description of the statistics, to which some of the following problem descriptions refer.
TIE writes to a .HIF file when the program completes its run.	Retranslate the image with the .HIF file available to VEST and rerun the translated program	See Section 4.2.3 for further information.
TIE statistics include Fault-on-Execute (FOE) conditions converted to lookups	Use the debugger to find FOE problems. Type SET BREAK/EXCEPTION and record addresses at which FOEs occur. Then use the VEST machine code listing for the image to map the addresses back to the original VAX code.	The record addresses that the debugger locates point you to the translated code at which the FOEs are occur. ring. FOEs typically occur when code pushes an address on the stack and then branches to it-a PUSHAB followed by an RSB instruction, for example. The number of FOEs reported determine whether fixing them is worth the effort since on a DEC 7000 running OpenVMS VAX Alpha, the TIE processes up to 25,000 FOEs per second.

Symptom	Action	Further information
TIE statistics describe complex instructions used.	If possible, recompile the program using a qualifier that instructs the compiler not to use certain sets of instructions. For example, the COBOL qualifier /INSTRUCTION_SET allows you to disable generation of packed decimal instructions. An OpenVMS VAX COBOL program compiled with this qualifier runs faster when translated.	Classes of complex instructions include character string, D-56 floating point, H character string, D-56 floating point, H See Appendix E for a list of VAX complex instructions and the way in which the TIE handles each instruction.
TIE statistics show number of instructions interpreted	Enable .HIF feedback and rerun the image. Then retranslate the image with the .HIF file	On a DEC 7000 running OpenVMS Alpha, the TIE interprets up to 40,000 instructions per second. If the number of instructions the TIE interprets is relatively large, then taking steps to avoid interpreting code is worthwhile. The same effort may not be worthwhile if the number of instructions interpreted is relatively small. Symptom statements that follow describe various reasons why the TIE interprets code.
VEST encountered an instruction that it cannot translate	If possible, recode the instruction so VEST can translate it.	For example, in the following case statement, VEST cannot determine what the CASE limit is: CASE sel, base, (R3) As a result, VEST cannot translate the instruction and the TIE must interpret it.
A branch was made to code that wasn't found during translation.	Enable .HIF feedback and rerun the image. Then retranslate the image with the .HIF file	See the beginning of this table for instructions on obtaining .HIF feed back and Section 4.2.3 for detailed information.
A branch was made to code in a write able image section descriptor (ISD) and /INTERPRET=WRITEABLE was specified during translation.	If no self-modifying code exists in the image, modify the offending PSECT statement to specify NOWRT or retranslate the image with the VEST qualifier /NOINTERPRET. If self-modifying code exists, move all static code out of the writeable ISD, leaving only self-modifying code.	This problem usually occurs only in VAX MACRO-32 code. Isolating self-modifying code limits the amount of run-time interpretation required to make the translated program work properly.

Symptom	Action	Further information
Translated image runs slowly and restrictive VEST options were used	Use restrictive VEST qualifiers only when necessary. When possible, split off the part of the program that needs to be restricted and create a separate shareable image. For example, if only one routine encounters dirty zeros during computation, split off the routine into a separate shareable image and then translate it with the /PRESERVE=FLOAT qualifier with out affecting the rest of the program.	Unnecessary restrictions on VEST behavior significantly impact performance. The restrictive VEST qualifiers include: /FLOAT /INTERPRET /OPTIMIZE /PRESERVE Software uses dirty zeros to indicate an array element that hasn't been filled in. A dirty zero occurs when the exponent equals 0, the sign bit equals 0, and the mantissa does not equal 0. On OpenVMS VAX systems, a dirty zero is computationally the same as a clean zero. But on OpenVMS Alpha systems, a dirty zero results in a fault. The TIE provides support for dirty zeros, but you must first translate the image using the /PRESERVE=FLOAT qualifier.

Table C-2: Problem: Executes improperly or returns incorrect results

Symptom	Action	Further information
Floating point results include precision errors.	If the image uses D floating point, check to see if the image really needs56 bits of precision. If it does, re translate with the /FLOAT=D56_FLOAT qualifier. If the image uses D floating point, if you translated it with the point, if you translated it with the/FLOAT=D56_FLOAT qualifier, and precision errors still occur, then the image uses either native routines to perform some computations or translated run-time libraries (RTLs) with insufficient precision. If you need full D-56 precision, substitute translated RTLs with 56 bits of precision for the native routines. For example, issue a command like the following: `$ DEFINE MTHRTL_TV MTHRTL_D56_TV`	The Alpha AXP hardware supports 53-bit mantissas (D-53). By default, VEST translates images using hardware D floating support. Native routines using D floating point always use D-53. Only translated images provide support for D-56.
Translated image uses unsupported system services.	If the image was linked with the SYSLIB qualifier, relink the image without it. If user code within the image calls unsupported system services, rewrite the code to use only system services supported by OpenVMS VAX.	Old source code often includes unsupported system services to perform common tasks for which system services were not yet available.
Parameters passed incorrectly from translated to native routines.	Check the source code of the translated caller to verify that it passes the types of data expected by the native routine. If the interface to the native routine has been changed intentionally, provide a jacket routine to convert from one interface to the other. Otherwise, fix the parameters of the native routine. If the native code was written in VAX MACRO-64, check the signature array to see if it contains the correct values.	

Symptom	Action	Further information
WHEN statements in VAX BASIC programs execute improperly or not at all.	Retranslate the image with the VEST qualifier/OPTIMIZE=NOSCHEDULE.	BASIC implements WHEN statements using a PC correlation table. Code scheduling blurs the line between VAX instructions and prevents correlation from working. The correlation table assumes that code for a particular line is sequential and is not scheduled with code from other lines that may be outside the scope of the WHEN statement.
Translated image produces erratic results	If the image uses global sections and requires byte or word granularity, retranslate the image with the VEST qualifier /PRESERVE=MEMORY_ATOMICITY to avoid word tearing. If the image requires VAX style instruction atomicity because data sharing occurs between AST threads and normal code, retranslate the image with the VEST qualifier /PRESERVE=INSTRUCTION_ATOMICITY.	If the image shares data between execution threads, then you need to set /PRESERVE qualifiers properly when you translate the image.
Translated shareable images do not work properly	Check the following: What .HIF and .IIF files are used when translating the shareable and main images? Use the qualifier /WARNINGS=VERBOSE to determine which files VEST reads. Are the image_TV logicals defined? Is a .SIF file used to keep the visible interface of a shareable image the same from one translation to another? If not, then you need to relink or retranslate all images that depend on that particular shareable image.	See Chapter 6 for information about translating shareable images and using .SIF files.

Table C-3: Problem: Crashes or exits with fatal messages or access violations

Symptom	Action	Further information
Translated image exits with fatal system message TRANSCALLER or is printed only when the offset of the NOCALLTRANS.	Recompile the native code using the/TIE qualifier and relink it using the/NONATIVE_ONLY qualifier. If the native code was written in VAX MACRO-64, you must code signature arrays into the procedure descriptors for the routines contained in native code.	The TRANSCALLER message includes the address of the procedure descriptor. of the offending image. This message is printed only when the offset of the signature array is 0. If the offset is 1, then the default signature information is being used. If greater than 1, then the field is an offset to the signature array. These structures are described in detail in the OpenVMS Alpha 32-bit Calling Standard. The NOCALLTRANS message indicates that the autojacketing routine EXE$NATIVE_TO_TRANSLATED detected that the caller doesn't have a procedure signature array. in The VAX MACRO-64 Assembler for OpenVMS Alpha does not support the /TIE qualifier because you must code the procedure signature block by hand. When coding procedure descriptors, you must specify a procedure signature block if the routine is to be callable by translated code.
Translated image crashes with the message HPARITH	Examine the summary bits: If the summary is 2, a dirty zero was reported. Retranslate the image with the /PRESERVE=FLOAT qualifier. If the summary is 8, a floating point overflow was encountered. Retranslate with the /FLOAT=D56_FLOAT qualifier.	The summary bits indicate the reasons for the crash. If the image uses D floating point, it may be encountering an overflow be cause of rounding at the last three bits of precision.

Translated image incurs an access violation.	Check to see if the program dynamically sets restrictive page protections and depends on VAX page size. If it does, remove the dependency on VAX page size. Relinking the OpenVMS VAX image with the /BPAGE=16 qualifier may help if the pages being protected are not mixed in an image section descriptor (ISD) with pages for which the protection is not changed at run time. Use the VEST qualifier /SHOW=MACHINE_CODE to get a machine code listing. Then use the listing to analyze the access violation.	OpenVMS Alpha Version 5.4-2 or later provides an item code for SYS$GETSYI that gives the page size at run time. For programs linked on previous versions of OpenVMS VAX, you must code carefully to avoid page size dependencies. If the original program was compiled and linked with /DEBUG, then the VEST machine code listing includes symbolic information that can be used to track problems back to source code.

Table C-4: Problem: Exceptions never terminate

Symptom	Action	Further information
Exceptions never terminate or the image crashes when exception handlers are used.	Check to see that the exception PC is not correlated with some program action. Check to see if the program performs its own stack unwinds. Check to see if the program knows how many frames to unwind at a particular location.	In some cases, it may be possible to force VEST to generate code that handles this properly. See Example C-1. In other cases it may be necessary to recode to eliminate the need for PC correlation. Since OpenVMS VAX and OpenVMS Alpha stacks are interlaced, a program performing its own stack unwinds may inadvertently unwind OpenVMS Alpha frames. Only use SYS$UNWIND to perform stack unwinds. The only safe way to determine how many frames to unwind within an exception context is to use the depth count in the mechanism array.
Exception handler changes values in registers by modifying the signal array	Rewrite the affected code.	This behavior is unsupported.
Exception handler modifies the PSL or is dependent on values in the PSL.	Rewrite the affected code	The program status longword (PSL) is not available in an exception handler. The OpenVMS Alpha system reports the Alpha AXP program status (PS).
Exception handler changes the PC.	Rewrite the affected code to use SYS$UNWIND.	Changing the PC in the signal array and then specifying SS$_CONTINUE or using SYS$UNWIND only works if the original PC and new PC are translated or in VAX code.

Table D-1: Untranslatable Images

Description	Message
Images linked prior to VAX VMS Version 4.0	BADEXE
Images that have user-written system services or other nonuser-mode code	ISDPROTECT ISDVECTOR
Images linked against a specific version of VMS and that reference system space	LNKSYS
Images that make direct reference to system space ad dresses	UNSUPABSREF
Images that make absolute reference to P1 space outside the supported range of OpenVMS VAX system service vector entries, which includes all system services prefixed by SYS$ (SYS$QIOW through SYS$EVDPOSTEVENT)	UNSUPABSREF
Images that make absolute reference to P0 space addresses not mapped by any image	UNSUPABSREF
Images that are based, that is, they have been linked such that code and data are tied to fixed addresses	ISDBASED
Images that cannot be translated because VEST cannot include both shared and unshared writeable image sections in a single Alpha AXP page	ISDCONFLICT

Table D-2: Images Translatable with Warnings

Description	Message
Images that contain vector instructions	VECTOROPC
Images that contain privileged instructions	PRIVOPC
Images that modify return addresses	WRITECF4 WRITEJSBRET
Images that modify the call frame	WRITECF0 WRITECF1 WRITECF2 WRITECF3

Table D-3: Images with Undetectable Translation Problems

Description	Where to find information
Images that depend on 512-byte page protection granularity	Migrating to an OpenVMS Alpha System: Planning for Migration
	Migrating to an OpenVMS Alpha System: Recompiling and Relinking Applications
Images that depend on global sections being aligned on 512-byte boundaries	Migrating to an OpenVMS Alpha System: Planning for Migration
	Migrating to an OpenVMS Alpha System: Recompiling and Relinking Applications
Images that use most of the VAXP0 or P1 space or are otherwise sensitive to the space taken up by the Translated Image Environment (TIE) and translated code, for example, an image that allocates a lot of dynamic memory	Undetectable

Table D-4: Translatable Images with Performance Issues

Description	Message
Images that include self-modifying VAX code, which must be interpreted	RWINTERP
Images that generate VAX code at run time, which must be interpreted	Undetectable
Images with code that inspects the instruction stream (I-stream); VEST generates slower RET and RSB code in this case	READCF4 READJSBRET
Images that depend on D_floating, H_floating, or packed-decimal instructions; these instructions require software emulation	VAXDFLOAT VAXHFLOAT VAXPACKED