Rational Application Developer

VisualAge Generator to EGL Migration Guide

Version 6 Release 0

SC31-6830-00

Rational Application Developer

VisualAge Generator to EGL Migration Guide

Version 6 Release 0

SC31-6830-00

Note

Before using this document, read the general information under "Notices" on page 319.

First Edition (November 2004)

This edition applies to the following licensed programs:

- Rational Web Developer
- Rational Application Developer

IBM welcomes your comments. You can send your comments by mail to the following address:

IBM Corporation, Attn: Information Development, Department 53NA Building 501, P.O. Box 12195, Research Triangle Park, NC 27709-2195.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Preface

This document is intended for those who want to migrate from VisualAge Generator 4.5 to the Enterprise Generation Language (EGL).

Who should read this book

This book is intended for programmers or system administrators who want to migrate code from VisualAge Generator 4.5 to the Enterprise Generation Language (EGL).

Related information

Related documents are provided in one or more of the following formats:

- Online book files (.pdf) on the product CD-ROM. Adobe Acrobat Reader is used to view the manuals online and to print desired pages.
- HTML files (.htm) on the product CD-ROM.

The most recent version of this book is available as an online book file (.pdf) on the following web site:

http://www.ibm.com/developerworks/rational/library/egldoc.html

Contents

Part 1. Migration overview

Chapter 1. Migration Overview

The Rational® Developer Products with the Enterprise Generation Language (EGL) component are the successor products for VisualAge Generator. Migration of your VisualAge Generator (VAGen) source code is required to convert to EGL. This migration guide provides information about planning your migration, using the migration tools to convert your source code, and additional steps needed to complete your migration after running the migration tools.

In addition to this Migration Guide, you should check the following for additional or more current information:

- The Rational Developer products online help system.
- The *EGL Reference Guide*.
- The web site and news group for VisualAge Generator. The web site is at: http://www.ibm.com/software/awdtools/visgen/
- The web site and news group for the Rational Developer product that you are using.

What is new in EGL that requires migration?

The Rational Developer products and EGL represent major changes and enhancements from VisualAge® Generator, including the following:

- Changes to the VAGen language, including many enhancements such as new data types, multidimensional arrays, dynamic arrays, the *case* statement, and improved web support.
- Changes to the user interface you use to develop your programs, including content assist, code templates to create a part, and a text editor for most part types.
- Changes to the generation and preparation process, including only Java™ generation rather than C++ and Java generation for distributed platforms and the use of a build server instead of preparation JCL templates for COBOL generation.
- Changes to runtime, including the use of Enterprise Developer Server for runtime services.
- Changes to library management, including your ability to choose your own source code repository to interface with your Rational Developer product.

The differences between the VAGen language and EGL are extensive. In the past when you upgraded from one version of Cross System Product or VisualAge Generator to a new version, there were only minor changes to the language. The previous migration tools were able to migrate each part independently of any other parts. However, due to the differences between the two languages, the VisualAge Generator to EGL migration tool must migrate each part in the context of other referenced or associated parts to determine the following:

- The part type of the referenced part
- Information that must move to the referencing part due to the new EGL syntax
- The location of the referenced part within the workspace

Cross-part migration is the term used to describe this situation in which the migration of one part depends on other parts. Cross-part migration is required to

produce the best possible conversion from the VAGen language to EGL. This in turn means that you need to carefully consider which groups of parts you migrate together.

Given the differences between VisualAge Generator and the Rational Developer products and the need for cross-part migration, this migration is a major undertaking and needs to be carefully planned.

Planning your migration

You need to consider the following tasks when planning your migration project:

- Plan a pilot project for migration:
  - Select the developers and systems support personnel that will be involved in the pilot project.
  - Select a small subset of your source code to use in the pilot project. Use this small subset to verify your environmental setup and your library management procedures and tools.
  - Upgrade to VisualAge Generator 4.5 with Fix pack 4. If you use VisualAge Generator with VisualAge for Java, you must also install APAR PQ88461 to upgrade the VisualAge Generator Utilities feature. Contact IBM® Support to obtain the fix for the APAR or check the VAGen web site at http://www-306.ibm.com/software/awdtools/visgen/support and then follow the link in the Download section. Also review Appendix F, "Situations where incorrect External Source Format causes problems in creation of EGL," on page 307 for additional VisualAge Generator APARs that might be necessary for your specific situation.
  - Install DB2® if it is not already available. DB2 is required for the migration database.
  - Review the capabilities of the Rational Developer product that you plan to use. Be sure that it includes the features that you require. For example:
    - If you plan to generate COBOL for the zOS environment, you must use Rational Application Developer for z/OS®.
    - If you plan to use iSeries™, you must use Rational Application Developer for iSeries.
  - Review the prerequisites for the Rational Developer product that you plan to use. In addition, review the prerequisites for your runtime environment. For example:
    - If you plan to generate COBOL for the zOS environment, be sure to review the prerequisites for the Enterprise Developer Server for zOS product.
    - If you plan to generate Java for the Unix System Services (USS) environment, be sure to review the prerequisites for the Enterprise Developer Options for zOS components shipped with Rational Application Developer for z/OS
    - If you plan to generate for iSeries, be sure to review the prerequisites for the runtime component of Rational Application Developer for iSeries
    - If you plan to generate Java for a workstation environment, be sure to review the prerequisites for the runtime component of the Rational Developer product that you plan to use.
- Obtain education for the team that will run the pilot project:
  - Rational Developer product environment
  - EGL language
  - VisualAge Generator to EGL migration tools

- Run the pilot project plan to do the following:
  - Install the Rational Developer product for the pilot team, and be sure to install on a machine that has the same regional settings as you used for developing your VAGen programs. For example:
    - If you developed your VAGen programs on a German machine, you should install your Rational Developer product on a German machine. This ensures that the comma used as a decimal point and German umlaut characters are migrated correctly.
    - If you developed your VAGen programs on a Chinese machine, you must install your Rational Developer product on a Chinese machine using the same code page. This ensures that your DBCS characters are migrated correctly.
  - Run the VAGen Migration Tool for the pilot set of code. See the following sections for information on the migration tool:
    - Chapter 2, “Migration Tool Philosophy,” on page 13
    - Part 2, “Migrating from VisualAge Generator 4.5 on Java to EGL,” on page 87
    - Part 3, “Migrating from VisualAge Generator 4.5 on Smalltalk to EGL,” on page 105
    - Part 4, “Stages 2 and 3— common to Java and Smalltalk migration,” on page 125
    - Part 5, “Completing the migration,” on page 149
  - Create library management processes:
    - Select and install a source code repository, including access from the developer workstations.
    - Define change management procedures that work with your corporate culture and your selected source code repository.
    - Develop any tools you need for your change management procedures, including the following:
      - Checkin and checkout procedures.
      - Version control procedures.
      - Tools to retrieve source code from the source code repository and to load a Rational Developer workspace or directory structure if you want to use batch generation.
  - iSeries COBOL target environment:
    - Follow directions in the *EGL Server Guide for iSeries*.
  - zOS COBOL target environments:
    - Install prerequisites for the Enterprise Developer Server for zOS V5.0 product, including any changes to your COBOL compiler and runtime.
    - Install the Enterprise Developer Server for zOS V5.0 product.
    - Install any required PTFs to Enterprise Developer Server for zOS.
    - Install the build server. Also customize the pseudo-JCL build scripts.
  - Java target environments:
    - Review the runtime platform differences if you are changing platforms (for example, from Windows® CICS® to Windows native). Make any code changes that result. See “Differences between distributed CICS and native workstation environments” on page 166 for a list of the differences.
    - Obtain JDBC support from your vendor if you are currently using ODBC support.

  - Generate and prepare your programs:
    - Review and modify your build descriptor parts. See “Reviewing your EGL build descriptor parts” on page 153 for a list of changes that cannot be handled by the VAGen Migration Tool.
    - Optionally, build an EGL batch generation server machine. This requires the use of a source code repository and the creation of tools to load a directory with all the parts you need for generation.
  - Testing:
    - Test your development environment to make sure that you can successfully debug programs. Debugging in the development environment might require access to DB2, remote VSAM files, and non-EGL programs that only run in your runtime environment.
    - Test at least a representative sample of your programs to ensure you understand any runtime differences. See “Runtime differences” on page 163 for a list of differences.
    - Test your library management procedures and tools using typical changes that you might make to the EGL source code. Be sure to test your procedures for changing common code, forms, tables, and programs for each target environment. Also test your procedures for adding common code, forms, tables, and programs for each target environment.
  - Refine your library management procedures and tools based on the results of the pilot project.
  - Document the findings of the pilot project, including:
    - Code changes that need to be made, particularly if you are changing target environments.
    - Changes developers need to make to any personal build descriptor parts.
    - References to sections of the Migration Guide that are particularly useful for your developers based on the problems you encountered during the pilot project.
    - Changes in runtime behavior that your end users will notice.
- Build a plan to complete your migration based on the findings from the pilot project.
- Provide education for the remaining developers:
  - Rational Developer product environment
  - EGL language
  - Your new source code repository
  - Your new library management process
  - Your new generation process

Determining whether you can migrate to EGL

EGL is the strategic component in the Rational Developer product to which VisualAge Generator customers should migrate. EGL support is not meant to be a complete replacement for **ALL** functions and platforms supported by VisualAge Generator Developer 4.5 (VAGen). Depending on your target environment and the types of programs you have developed with VisualAge Generator, you might need to wait for a future release of EGL.

The following list shows VisualAge Generator target environments supported in the current release of EGL.

- MVS CICS

- MVS™ Batch
- Unix System Services
- Windows Native
- AIX® Native
- Linux™ on Intel™ platforms
- iSeries

Note: While VisualAge Generator generates Java and C++ for certain platforms, EGL only generates Java.

For additional considerations in these supported environments, see the following:

- Special considerations for migrating to EGL — File and data base access, Table 1 on page 7
- Special considerations for migrating to EGL — User interface, Table 2 on page 7
- "VisualAge Generator features not available in EGL" on page 8

The following tables list special considerations for supported environments.

Table 1. Special considerations for migrating to EGL — File and data base access

VAGen file and database access	Special consideration
SQL	Supported in EGL.
Serial, indexed, and relative records	Supported in EGL.
Message queue records	Supported in EGL.
DL/I	Not supported in EGL. You must wait until a future release to migrate.
GSAM	Not supported in EGL. You must wait until a future release to migrate.
IMS™ Message Queues	Not supported in EGL. You must wait until a future release to migrate.
Btrieve	Not supported in EGL.
Local VSAM	Supported for COBOL generation and for Java generation for AIX. Not supported for debug or for Java generation for other environments.

Table 2. Special considerations for migrating to EGL — User interface

VAGen user interface	Special considerations
Text user interface, including print	Supported in EGL for both COBOL generation and Java generation.
Web transactions and User Interface (UI) records	Not supported in EGL. You must wait until a future release to migrate.
JSP and Java servlets which use VAGen Java wrappers	• You can migrate your JSP and Java servlets to a Rational Developer product using the information provided by that product. • You can migrate your VAGen server programs to EGL using this VAGen migration guide. You can generate the Java wrappers using EGL.

Table 2. Special considerations for migrating to EGL — User interface (continued)

VAGen user interface	Special considerations
Java GUI applications or applets that do *not* use VAGen parts on the free form surface, but which use VAGen Java wrappers.	• You can migrate your Java applications or applets to the Rational Developer product using the information provided by the Rational Developer product for migrating Java code from VisualAge for Java. • You can migrate your VAGen server programs to EGL using this VAGen migration guide. You can generate the Java wrappers using EGL.
Java GUI applications or applets that use VAGen parts on the free form surface.	Not supported in the current release.
Smalltalk GUI views or visual parts.	Not supported in the current release. The views with VAGen parts must be migrated to Java-based solutions. EGL will not have any Smalltalk-based solutions.

VisualAge Generator features not available in EGL

In addition to the special considerations listed in Tables 1 and 2, if you need any of the features in the following list, you should assess the impact of migrating now versus migrating in the future:

- Specialized editors and lists--
  - Graphical editor for developing or maintaining maps. EGL provides a text editor.
  - Listing of the program produced during generation.
- Specialized functionality--
  - Searching for references in a selected set of parts and limiting the list to a program's associates.
  - Filtering parts by part type or by subtype. EGL provides a search capability so you might be able to search on a specific part type or subtype.
- EBCDIC mode to facilitate testing when accessing host databases, files, or called programs.
- VisualAge Generator Templates.
- If you plan to use Java generation, also determine if you do any of the following:
  - Substitute your own message text for the EGL runtime messages.
  - Use CICS specific functions that cannot be converted to native runtime environments. See "Differences between distributed CICS and native workstation environments" on page 166 for details of the differences.

Terminology differences

VisualAge Generator Developer on Java (VAGen on Java), VisualAge Generator Developer on Smalltalk (VAGen on Smalltalk), and the EGL support in Rational Developer Products all use different terminology. To help you relate the VAGen terminology to the EGL terminology, the following six tables show the three sets of terminology.

Table 3. Code organization terminology differences

VisualAge Generator on Java	VisualAge Generator on Smalltalk	Enterprise Generation Language (EGL)
Workspace	Image	Workspace
Project	Configuration map	EGL Project
Package	Application	EGL source folder and EGL Package containing one or more EGL files
(no comparable concept)	(no comparable concept)	File (generally a Java package or a Smalltalk application will split into multiple files). An EGL file contains one or more EGL parts of one or more part types.
Class or Type	Class	EGL part type
Method or Member	Method	(no comparable concept)
VAGen part	VAGen part	EGL part within a file

Table 4. VAGen parts and concepts terminology differences

VisualAge Generator on Java	VisualAge Generator on Smalltalk	EGL
Shared data item	Shared data item	TypeDef to a DataItem part
Non-shared data item	Non-shared data item	primitive item definition
Data item part	Data item part	DataItem part
Map group part	Map group part	FormGroup part
Map part: • display map • printer map	Map part: • display map • printer map	Form part: • textForm • printForm
I/O option and I/O object	I/O option and I/O object	EGL I/O statement
Java application or applet (GUI)	Smalltalk view or visual part (GUI)	• Smalltalk view and visual parts are not supported. • Java applications and applets are supported *if* you did not use VAGen parts on the free form surface. If you did use VAGen parts on the free form surface, then the Java application or applet is not supported in the current release.
Generation options part	Generation options part	Build descriptor part
Generation option	Generation option	Build descriptor option
Linkage table part	Linkage table part	Linkage options part

Table 5. VAGen with IDE Windows terminology differences

VisualAge Generator on Java	VisualAge Generator on Smalltalk	EGL
Log • Shows error messages • Product closes only if you close BOTH the Log and the Workbench • Workspace is ALWAYS saved when you close the product	System Transcript • Shows error messages • Product closes if you close EITHER the System Transcript or the VisualAge Organizer • Image is OPTIONALLY saved when you close the product	Console • Shows messages. Problems view • Shows messages, especially those related to syntax validation. • Workspace is ALWAYS saved when you close the product.
Workbench • Shows the projects and packages in the workspace.	VisualAge Organizer • Shows the applications in the image.	EGL and Web perspectives: • Navigator and Project Explorer views show the projects, source folders, packages, and files in the workspace.
Scrapbook	Workspace	Scrapbook page editor
Repository Explorer	Application Editions Browser	No comparable concept in EGL. The repository you decide to use might have a comparable concept.
VAGen Parts Browser • 3 panes show package, part type, and VAGen parts • Filtering is included in the browser	VAGen Parts Browser • 3 panes show application, part type, and VAGen parts • Filtering is included in the browser	EGL and Web Perspectives: • Navigator and Project Explorer views show the projects, source folders, packages and files in the workspace. • Outline view shows the parts within a file.
VAGen options	VAGen preferences	EGL preferences
VAJava options	VASmalltalk preferences	Other Rational Developer product preferences
References tool to find parts that use a specific part name or text string	References tool to find parts that use a specific part name or text string	EGL Search
Associates tool to find all parts referenced by a specific part	Associates tool to find all parts referenced by a specific part	EGL Parts Reference

Table 6. VAGen Workspace management terminology differences

VisualAge Generator on Java	VisualAge Generator on Smalltalk	EGL
Repository	Library	None. CVS and Clear Case LT are provided depending on the Rational Developer product that you use. You can choose your own repository management system.

Table 6. VAGen Workspace management terminology differences (continued)

VisualAge Generator on Java	VisualAge Generator on Smalltalk	EGL
Add / Delete	Load / Unload	If you decide to use a repository, the repository might have a comparable concept.
Replace with	Load another edition	Replace with local history **Note:** The repository you decide to use might have additional facilities.
Compare with	Browse changes	Compare with local history **Note:** The repository you decide to use might have additional facilities.

Table 7. VAGen Repository management terminology differences

VisualAge Generator on Java	VisualAge Generator on Smalltalk	EGL
Administrator	Library Supervisor	If you decide to use a repository, the repository might have a comparable concept.
Repository management: • Purge / Restore • Compact	Library management: • Purge / Salvage • Clone	If you decide to use a repository, the repository might have a comparable concept.

Table 8. VAGen source code management terminology differences

VisualAge Generator on Java	VisualAge Generator on Smalltalk	EGL
Ownership: • Project owner • Package owner • Class owner	Ownership: • Configuration map manager • Application manager • Class owner	If you decide to use a repository, the repository might have a comparable concept.
Version and release	Version and release	If you decide to use a repository, the repository might have a comparable concept.
Project: 1. A project is required. 2. VAGen Project List Part specifies relationships between projects. 3. The package owner can always release the package to the project.	Configuration map: 1. Usage is optional. 2. Required map specifies relationships between configuration maps. 3. Optionally, you can delegate the release of applications or restrict their release to the configuration map manager.	Project: 1. A project is required. 2. EGL Build Path property for the project. However, this does not automate loading projects together into the workspace. 3. No comparable concept, unless provided by the repository.

Table 8. VAGen source code management terminology differences (continued)

VisualAge Generator on Java	VisualAge Generator on Smalltalk	EGL
Package: 1. No comparable concept 2. No comparable concept 3. No comparable concept 4. Group members 5. Versioning the project automatically versions the included packages.	Application: 1. Prerequisite application 2. Subapplications 3. Privileges 4. Group members 5. You must version the application before you version the configuration map	Folder or Package: • If you decide to use a repository, the repository might have a comparable concept.
Class or Type: • Versioning the package or project automatically versions the included classes	Class: • You must version and release the class before you version the application	EGL part type: • No comparable concept in EGL.
VAGen parts: • There is a date and time stamp for each part • Packages containing duplicate part names CAN be added to the workspace. • There is a duplicate parts tool to locate the duplicate parts	VAGen parts: • There is a date and time stamp for each part • Applications containing duplicate part names CANNOT be loaded into the image.	EGL parts: • Parts are in EGL files; only the EGL file has a date and time stamp. • You can have duplicate parts in the workspace. EGL uses a combination of a project's EGL build path, the file's import statements, and the containerContextDependent property to determine the name space that is searched to resolve references to part names. Part names must be unique within the name space. The EGL build path for a project limits which additional projects are considered when looking for a part name. The import statement for a file limits which additional packages and/or parts within the EGL build path are considered when looking for a part name. The containerContextDependent property for a record or a function specifies that EGL should use the EGL build path and import statements for the file containing the program rather than from the file containing the record or function.

Chapter 2. Migration Tool Philosophy

The VisualAge Generator to EGL migration tool is actually a series of tools. This chapter provides a high-level overview of the tools and describes the techniques used by the tools.

The design of the VisualAge Generator to EGL migration tools has several major objectives:

- Preserve the program behavior from VisualAge Generator to EGL.
- Preserve the Java project and package structure from VisualAge Generator to EGL when appropriate.
- Preserve the Smalltalk configuration map and application structure from VisualAge Generator to EGL when appropriate.
- Enable you to perform incremental migration of subsystems, one subsystem at a time.
- Enable you to migrate multiple versions of your subsystems.

The design of the VisualAge Generator to EGL migration tools also has several secondary objectives:

- Use batch mode processing as much as possible with opportunities for you to optionally review the planned migration at critical points before proceeding to the next step.
- Store information about the planned migration in a database so that it can be preserved across multiple project versions and multiple subsystems. This also enables you to save intermediate results as backup. This is important if you have large numbers of parts in your repository.
- Provide a set of sample programs for the tool that extracts the VAGen source from your repository and loads the migration database. You can optionally tailor the sample programs to more accurately reflect your environment.

The design of the VisualAge Generator to EGL migration tools is based on the following assumptions:

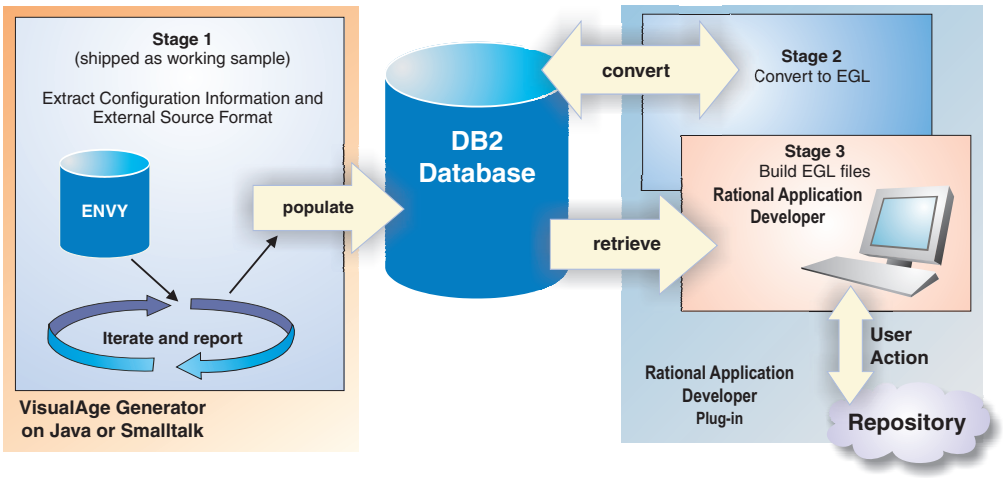
- Migration is from VisualAge Generator 4.5 **using External Source Format that is produced by** VisualAge Generator 4.5.
- The parts to be migrated **are valid VisualAge Generator parts**. Programs, tables, and map groups can be validated and/or generated in VisualAge Generator 4.5.

There are two methods for using the VisualAge Generator to EGL migration tools:

- Stage 1 to 3 migration, which is described in “Overview of the VisualAge Generator to EGL Migration Tools” on page 14. This is the primary technique for migrating your source code.
- Single File Migration, which is described in “Overview of Single File Migration” on page 20. This technique is useful for migrating a few programs to verify that your environment is working properly.

Overview of the VisualAge Generator to EGL Migration Tools

To achieve the objectives listed, the VisualAge Generator to EGL migration tool is actually a series of tools that are organized into three stages as shown in the following figure.



- The tool for Stage 1 runs in the VAGen environment. The Stage 1 tool extracts information about the organization of your source code and the source code itself from your Java repository or Smalltalk library. The Stage 1 tool loads this information into a migration database. The VAGen source code is stored in External Source Format.
- The tool for Stage 2 runs in the Rational Developer product environment. The Stage 2 tool uses the information that is stored in the migration database to create EGL syntax for the VAGen parts that were stored in the migration database during Stage 1. The Stage 2 tool stores the resulting EGL source code in the migration database.
- The tool for Stage 3 also runs in the Rational Developer product environment. For each EGL project you want to create, the Stage 3 tool extracts the EGL source for the parts that belong to that project and creates an EGL project in the file system for you. Optionally, if you are only working with one version of a set of projects, the Stage 3 tool can import the projects into your the Rational Developer product workspace.

After you have the projects in your workspace, you can then version the projects with whatever source code repository you have decided to use. You use the tools provided by the source code repository to manage your source code.

Migration tool terminology

To achieve a good cross-part migration, when you migrate a part, you must provide not only the part itself but all parts that it references. For example, when you migrate a program, you should provide not only the program, but also all the parts that the program references. For a program, the set of parts that you need when you migrate the program is the same set of parts that you need when you generate the program in VisualAge Generator. This set of parts is the program's associates list.

In VisualAge Generator, the common techniques for providing all the parts for generation are as follows:

- Project List Parts (PLPs) in VisualAge Generator on Java

- Configuration maps in VisualAge Generator on Smalltalk.

The migration tool makes use of these two techniques. The tool uses the following terminology:

- If you are migrating from VisualAge Generator on Java:
  - A *high-level PLP project* is a Java project that contains a Project List Part (PLP) and is not referenced by any other PLP.
  - A *migration set* consists of all the VAGen projects referenced in a Java high-level PLP project, including all VAGen projects in the entire PLP chain starting at the high-level PLP project.
- If you are migrating from VisualAge Generator on Smalltalk:
  - A *high-level configuration map* is a Smalltalk configuration map that is not listed as a required map by any other configuration map.
  - A *migration set* consists of all the Smalltalk configuration maps listed as required maps in a Smalltalk high-level configuration map. The migration set includes all the configuration maps from the entire chain of Smalltalk required maps starting at the high-level configuration map.
- A *migration plan* is a file that specifies the information for one or more migration sets. If you specify a migration plan file name in your Stage 1 preferences then all the migration sets that match your repository filters are placed in the same migration plan file. If you do not specify a migration plan file name, then each migration set is placed in a separate migration plan file.

Note: If you are migrating from VisualAge Generator on Java and do not currently use PLP projects, you can create PLP projects to use just for migration. Alternatively, you can do one of the following:

- If you have information in a database or other system that specifies what is needed for generation in terms of Java project versions, then you can write a tool to create the migration plan file (or files) automatically from your database.
- Create the migration plan file (or files) by hand.

If you are migrating from VisualAge Generator on Java, see the section "Migration plans and high-level PLP projects" on page 101 for more details.

Stage 1 Details

The Stage 1 tool is shipped as a sample program with the Rational Developer product. You install the sample program to run on either VisualAge Generator Developer on Java or VisualAge Generator Developer on Smalltalk, depending on the VisualAge Generator Developer 4.5 product that you currently use. The two sample programs differ somewhat due to the differences in the Java and Smalltalk environments. However, the basic steps for using the Stage 1 sample programs are the same in both environments. The basic steps for Stage 1 are:

- Step 1. Define rules and preferences to direct the Stage 1 migration.
- Step 2. Run the tool and produce one or more of the following outputs:
  1. One or more migration plan files
  2. A report showing how each migration plan file will be migrated
  3. A log file containing messages about any problems detected
  4. A migration database

Step 1

Define rules and preferences that provide the Stage 1 tool with information about what you want to migrate, including the following:

1. How to filter Java project names so that only the projects you want to migrate will be considered. For Smalltalk, you specify how to filter the Smalltalk configuration map names. This improves performance for Stage 1 because the tool only processes those Java projects or Smalltalk configuration maps that match your filters.
   - From those Java projects that match your filters, the Stage 1 tool selects any Java projects that contain a high-level Project List Part. A Java project contains a high-level Project List Part (PLP) if the Java project is not referenced by any other PLPs.
   - From the Smalltalk configuration maps that match your filters, the Stage 1 tool selects any high-level configuration maps. A high-level configuration map is one that is not listed as a required map by any other configuration map.
2. Whether you want to create one migration plan that reflects everything that could migrate based on your filter, or whether you want to create multiple migration plans, with one migration plan for each Java high-level PLP project version or Smalltalk high-level configuration map version.
3. How to create the EGL project, package, and file names from the Java project and package names or from the Smalltalk configuration map and application names. The information you can specify includes the following:
   - Rules that indicate which Java projects and packages or Smalltalk configuration maps and applications contain common code.
   - Renaming rules to be used when creating the EGL project and package names.
   - Names to be used for the EGL files that contain common parts or unused parts.
4. The name of the migration database and the user ID and password that are needed for access to the database.
5. Which outputs you want the Stage 1 tool to produce in Step 2. You can choose to create all the outputs in a single step or you can create the outputs in sequence so that you have a chance to review your rules and preferences before creating the next, more time-consuming output.

Step 2

Based on the rules and preferences you have defined, the Stage 1 tool produces the following possible outputs:

1. **Migration plan file (or files).** A migration plan file contains migration sets. Each migration set is one high-level PLP project version from the Java repository or one high-level configuration map version from the Smalltalk library. The dependent Java project versions or the required Smalltalk configuration map versions are specified in the migration set.
   - If the migration preference file does not specify a value for the migration plan filename option, then multiple migration plan files are created. Each high-level PLP project version for Java results in one migration plan file that contains one migration set. Similarly, each high-level configuration map version for Smalltalk results in one migration plan file that contains one migration set version.
   - If the migration preference file specifies a value for the migration plan filename option, then each high-level PLP project version for Java results in a migration set entry within the single migration plan file. Similarly, each high-level configuration map version for Smalltalk results in a migration set entry within the single migration plan file.

- For example, consider an Order Entry system that is made up of 5 Java projects and a sixth Java project that contains a PLP that specifies the versions of the other 5 projects. If you request multiple migration plans and 3 versions, then 3 migration sets will be created -- one for each version of the Java Order Entry project that contains the PLP part. Similarly for Smalltalk, if you want to migrate 3 versions of a configuration map that reflects that code that makes up the Order Entry system, there will be 3 migration sets -- one for each version of this high-level configuration map

You can direct the Stage 1 tool to stop at this point so you have the opportunity to review the migration plan file (or files) to ensure that the Java project versions or Smalltalk configuration map versions that you want to migrate are correctly reflected in the migration plan file (or files).

2. **A report showing how each migration set will be migrated.** The Stage 1 tool can produce this report **without loading the database**. This helps you ensure that your filters and preferences select the correct set of Java projects or Smalltalk configuration maps and that you are satisfied with the naming conventions of EGL projects, packages, and files that resulted from your renaming rules. Reviewing the report gives you an opportunity to refine your rules and preferences if you are not happy with the proposed EGL structure before the Stage 1 tool actually loads the database. You can iterate through the previous steps as many times as necessary until you are satisfied with what will be migrated and the proposed EGL structure. The report shows the following:
   - For Java, each migration set lists the project versions that are included. For each project version, you can see the package versions, and for each package version, you can see a list of the VAGen parts.
   - For Smalltalk, each migration set lists the configuration map versions that are included. For each configuration map version, you can see the application versions, and for each application version, you can see a list of the VAGen parts.

   For each VAGen part, you can see the corresponding EGL project, package, and file name where the part will be placed. For each VAGen part, you can also see both the associates list created by VisualAge Generator and the EGL file where the associate is placed. See section “Placing parts in EGL files” on page 32 for information on how the VAGen parts are assigned to files during Stage 1 – 3 migration.
3. **A log file provides messages** if any of the VAGen program, table, map group, or control part names conflict with the EGL reserved word list. These parts are not renamed during migration. You can either rename the parts in VisualAge Generator or wait until you have migrated to EGL.
4. **A migration database** loaded with the information and VAGen source code based on the migration plan files. You can select one migration plan to use in loading the database or all the migration plan files in a directory. The Stage 1 migration tool loads the data base with the following:
   - Information about each migration set within the selected migration plan file (or files).
   - The set of associated Java projects or Smalltalk configuration maps for the migration set.
   - The VAGen part definitions in External Source Format for each VAGen part in the set of Java projects or Smalltalk configuration maps.
   - The corresponding EGL project, package, and file names for each Java project, package and VAGen part or each Smalltalk configuration map, application and VAGen part.

- A report is also created during this step so that you have a complete record of what was loaded into the migration database. This report is in the same format as the previous report.

The Stage 1 tool is shipped as a sample program for both the Java and Smalltalk versions of VisualAge Generator. You can use the Stage 1 tool ″as is″ or you can modify the sample program to better fit your environment. For example, you might currently store configuration information outside the Java repository or Smalltalk library. This configuration information might specify which versions of your source code are required for generation. In this situation, you could use the sample programs as a guide to writing your own tool to load the migration database from a combination of your configuration information and your Java repository or Smalltalk library.

If you modify the Stage 1 sample programs, you might want to modify the migration database to include additional information to assist in the analysis of your code. You can add additional columns to the existing SQL tables or you can add additional tables to the migration database. However, these new columns and tables will not be used in Stages 2 and 3 of migration. Additionally, if you modify the Stage 1 sample programs, you must be sure to populate the SQL tables with the information shown in the sample programs. If you do not, Stages 2 and 3 will not be able to migrate your code.

See Chapter 4, “Stage 1 — Extracting from Java,” on page 89 for details about installing and running the Stage 1 tool on VisualAge Generator Developer on Java. See Chapter 5, “Stage 1 — Extracting from Smalltalk,” on page 107 for details about installing and running the Stage 1 tool on VisualAge Generator Developer on Smalltalk.

Stage 2 Details

The Stage 2 tool is shipped in the Eclipse plugin com.ibm.etools.egl.vagenmigration and runs in the Rational Developer product environment. Because the information you want to migrate is now in the migration database, you use the same Stage 2 tool regardless of whether you are migrating from VisualAge Generator on Java or VisualAge Generator on Smalltalk. The basic steps for running the Stage 2 tool are as follows:

1. You define rules and preferences to tell the Stage 2 tool what you want to migrate, including the following:
   - Specific details about how you want your EGL source code to be created. For example, the Stage 2 migration tool must split VAGen working storage records into two EGL basic records:
     a. A record that is named the same as the original working storage record and which contains all the non-level 77 items.
     b. A second record that is named the same as the original working storage record with a suffix and which contains all the level 77 items.

     There is a Stage 2 migration preference that enables you to specify the suffix you want the Stage 2 tool to use whenever it creates a new record to contain level 77 items.
   - Which migration set or sets you want to migrate. For example, if you created 3 migration sets to migrate 3 different versions of the Order Entry system, you might only want to migrate one version initially. This gives you the flexibility to limit migration, without having to migrate everything in the migration database at the same time.

- The name of the migration database and the user ID and password that are needed for access to the database. Both the Stage 2 and Stage 3 migration tools attempt the database logon with the user ID and password used to logon to the Windows machine if the database user ID and password are not specified explicitly.
- Whether you want to automatically start the Stage 3 tool after Stage 2 completes. If you run Stage 3 automatically, you can choose to load one version of the EGL projects into your workspace. You can also choose to load the EGL projects into a temporary directory so that you can interface with your source code repository at a later time.

2. Based on the rules and preferences you have defined, the Stage 2 tool does the following:
   a. Retrieves parts for one migration set from the database.
   b. Converts the External Source Format source code to EGL source code.
   c. Stores the EGL source code in the migration database. Messages associated with part migration are also stored in the migration database. This improves performance for Stage 2 because if the same part edition is used in another Java project version or Smalltalk configuration map version, the EGL source code is already available and not converted again.
   d. Creates a log file of any potential problems that are encountered, including generatable parts that conflict with the EGL reserved word list or ambiguous situations that the migration tool is unable to resolve.
   e. Iterates to process the next selected migration set.

The Stage 2 migration tool can be run in batch mode. See Chapter 6, “Stage 2—Conversion to EGL syntax,” on page 127 for details about installing and running the Stage 2 tool on the Rational Developer product. You cannot modify the Stage 2 migration tool.

Stage 3 Details

The Stage 3 tool is shipped in the same Eclipse plug-in (com.ibm.etools.egl.vagenmigration) as the Stage 2 tool and also runs in the the Rational Developer product environment. Because the information you want to migrate is now in the migration database, you use the same Stage 3 tool regardless of whether you are migrating from VisualAge Generator on Java or VisualAge Generator on Smalltalk. The basic steps for running the Stage 3 tool are as follows:

1. You define rules and preferences to tell the Stage 3 tool what you want to migrate, including the following:
   - Which migration set or sets you want to migrate. For example, if you created 3 migration sets to migrate 3 different versions of the Order Entry system, you might have migrated all 3 versions through the Stage 2 tool, but only want to migrate one version through the Stage 3 tool. The most common reason for doing just one version in Stage 3 is that you want to version this code in your source code repository, then migrate the next version with the Stage 3 tool and version it in your source code repository.
   - The name of the migration database and the user ID and password that are needed for access to the database.
2. Based on the rules and preferences you have defined, the Stage 3 tool does the following:
   a. Creates a ″to do″ list for the migration set. This ″to do″ list contains a consolidated list of the messages produced by Stage 2 that might require you to perform additional tasks to complete the migration.

b. Creates the EGL project and package structure in your workspace based on the information stored in the migration database during Stage 1.

c. Creates the .egl source files based on the EGL source code that was stored for the VAGen parts during Stage 2. The .egl source files include most import statements that are needed to resolve EGL part references. See "EGL build path and import statements" on page 29 for details about the import statements.

d. Creates the .eglbld files based on the EGL XML source that was stored for VAGen control parts during Stage 2. The control parts are generation options (EGL build descriptor parts), linkage options, resource associations, bind control and linkedit parts.

3. At this point you should do the following:

   a. Review the workspace for any messages in the Problems view to see if there are any validation errors. You can do this in conjunction with the log produced in Stage 2.

   b. Generate (without preparing) all programs and tables to ensure proper migration for your target environment. When you generate the programs, be sure to use the genFormGroup and genHelpFormGroup build descriptor options so that all your formGroups are generated. *This step is optional, but it is strongly recommended.*

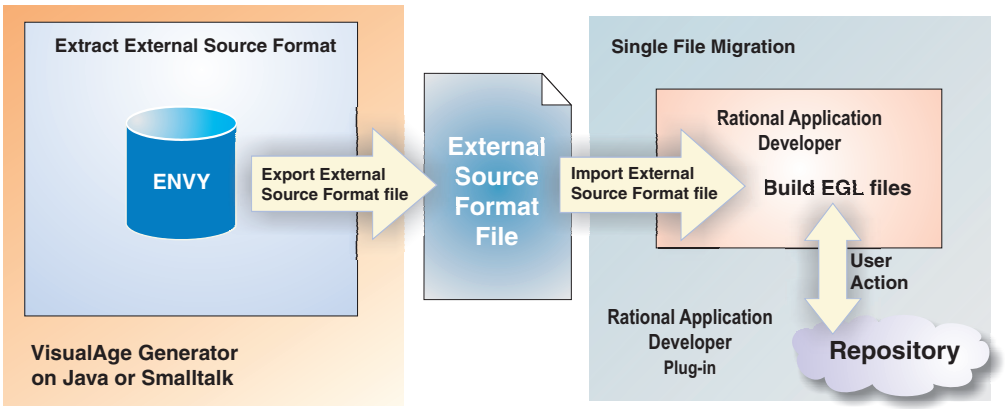
   c. Version or commit the EGL projects into your source code repository.

   d. Generate and test the migrated code. This step is also optional, *but it is strongly recommended.*

The Stage 3 migration tool can be run in batch mode. See Chapter 7, "Stage 3 — Import," on page 139 for details about running the Stage 3 tool on EGL. The Stage 3 tool is installed automatically at the same time you install the Stage 2 tool. You cannot modify the Stage 3 migration tool.

Overview of Single File Migration

When you are first getting accustomed to the Rational Developer product and setting up your environment, you might want to migrate just a few programs to verify your environment, ensure generation and preparation are working properly, and ensure your runtime environment is properly configured for EGL. In this case, you might not want to go through the full Stage 1 to 3 migration. The Stage 2 migration tool provides a mechanism for you to migrate programs using what is called single file migration as shown in the following figure:



Single file migration is a more manual process than Stage 1 to 3 migration. In single file mode, you use VisualAge Generator to export External Source Format

source code to a file. Then you use the Rational Developer product to create an EGL project and EGL package. From the Rational Developer product, you can then use the Import wizard to import the External Source Format file. The single file migration tool runs and does the following:

- Creates the target EGL source file, if it does not exist. If the file does exist, you have the option to overwrite or append to the file. Depending on your preferences and the parts contained in the External Source Format file, the migration tool might create additional EGL files.
- Converts the External Source Format source code to EGL source code.
- Creates a log file of any potential problems that are encountered, including generatable parts that conflict with the EGL reserved word list or ambiguous situations that the migration tool is unable to resolve.

The External Source Format to EGL conversion that occurs during single file migration is essentially the same syntax conversion that occurs during Stage 2 of the Stage 1 to 3 migration. However, single file migration has several limitations that do not make it suitable for large scale migrations. The limitations include:

- Only parts in the single External Source Format file are considered during migration. To achieve the best possible migration, include a program with all its associates in the External Source Format file.
- The placement of VAGen parts into files is different from that of Stage 1 – 3 migration. In single file mode, assuming you specified *targetFile.egl* as the target EGL file name, the following occurs:
  - All control parts are placed in a file called targetFile.eglbld.
  - If you do not select the preference to separate generatable parts into EGL files, all the remaining parts are placed in a file called targetFile.egl.
  - If you select the preference to separate generatable parts into EGL files, the following occurs:
    - Each program part is placed in a file called programName.egl, where programName is the name of the program.
    - Each table part is placed in a file called tableName.egl, where tableName is the name of the table.
    - Each map group and all maps in the map group are placed in a file called mapGroupName.egl, where mapGroupName is the name of the map group.
    - All the remaining parts are placed in a file called targetFile.egl. targetFile can be the same as programName.egl if you want to place all the remaining parts in the same file as the program. There is no attempt to determine which parts are shared by multiple generatable parts.
  - All the files are placed in same EGL project, source folder, and package.
- Because all the output files are placed in the same EGL package, the migration tool does not include any import statements. In addition, because all the parts are placed in the same EGL package, your original Java project and package structure are not preserved. Similarly, your original Smalltalk configuration map and application structure are not preserved.
- If the same part occurs multiple times in the External Source Format file, only the last definition is migrated.
- There are two alternative techniques for dealing with common parts when migrating in single file mode. Be sure you understand the disadvantages of each technique before choosing one of them. The two techniques are as follows:
  - If you migrate two External Source Format files to the same EGL package and the two files contain the same part, you will have duplicate parts that cannot

be resolved by EGL. This happens if you migrate a file with Program1 and its associates and a second file with Program2 and its associates and the two programs share some common parts. You can avoid this problem by migrating each program with its associates to a separate EGL package. This will still result in duplicate parts in the workspace, but because they are in different packages, EGL will be able to resolve the part references.

  - If you split the common parts out into a separate External Source Format file, you might not have all the information necessary to do a good VisualAge Generator to EGL migration on a single-file basis. For example, if you have an SQL record in one file, and a function that uses modified SQL for the record is in a different file, the migration tool cannot completely build the I/O statement for the function. In addition, if the common parts are in a different package, you must add EGL import statements to each file that needs to reference the common package (or packages).

- Additional processing that is done by Stages 2 and 3 is also not performed in single file mode. Some examples are as follows:
  - Forms are not nested within form groups.
  - Parts are not sorted by part name within part type in the file.
  - Only one line is included between the parts in the output files.

See "Migration challenges" on page 22 and "Techniques used by the VisualAge Generator to EGL Migration Tool" on page 27 for a better understanding of the differences between single file migration and Stage 1 to 3 migration.

Migration challenges

There are several differences between the VisualAge Generator and EGL approaches to writing and managing source code. The following differences are of particular importance to migration:

- EGL syntax in some cases is more precise than VisualAge Generator
- Differences in when and how part references are resolved
- Differences in handling common code

These differences are explained in more detail in the following sections.

Precise EGL Syntax

Even though the syntax of the two languages differs greatly, the VAGen language can, for the most part, be migrated to the EGL language while preserving the same behavior as the original VAGen program. However, there are number of situations in which the EGL syntax is more precise or more restrictive than in VisualAge Generator. These situations are rare in typical programs. However, when they do occur, the migration tool requires cross-part migration to determine the exact EGL syntax that preserves the behavior you required in VisualAge Generator. Cross-part migration means that the migration tool needs to have one or more other referenced parts available to be able to do a correct migration of the current part. The following are some examples:

- In VisualAge Generator you use the DISPLAY I/O option for both display (text) and printer maps. EGL provides the display statement for text forms and the print statement for print forms. To facilitate migration from VisualAge Generator, there is an EGL preference to indicate that you want VisualAge Generator Compatibility. The VisualAge Generator Compatibility preference permits the use of the display statement for print forms. During migration, if the program, its map group, and the map are all available, then the migration tool

can determine whether to migrate to a display or print statement. However, if the DISPLAY function is being migrated without a program, then the migration tool cannot definitively determine whether to use an EGL display or print statement. In this situation, the migration tool uses the display statement because it is tolerated for print forms in VisualAge Generator Compatibility mode.

- In VisualAge Generator you use the SET map PAGE statement for both display (text) and printer maps. This causes the screen to be cleared if the next CONVERSE or DISPLAY is for a display map and a page eject if the next DISPLAY is for a printer map. EGL provides the clearScreen system library function for text forms and the pageEject system library function for print forms. The VisualAge Generator Compatibility preference does not affect the use of clearScreen or pageEject. During migration, if the program, its map group, and the map are all available, then the migration tool can determine whether to migrate to the clearScreen or pageEject system library function. However, if the SET map PAGE statement is used in a function that is being migrated without a program, then the migration tool cannot definitively determine whether to use the clearScreen or pageEject system library function. In this situation, the migration tool uses EZE_SETPAGE, which is intentionally invalid EGL syntax. This results in an error in the Problems view so that you are aware you need to correct the function.
- In VisualAge Generator, you can specify either an edit table or an edit function as the edit routine for a map variable field. You cannot specify both. In EGL, you can specify both the validatorTable and the validator properties. If the edit table or the edit function is available during migration, the migration tool can determine whether to set the validatorTable or the validator property. However, if the part specified by the edit routine is not available, the migration tool cannot definitively determine whether to set the EGL validatorTable or validator property. In this situation, the migration tool attempts to determine whether the edit routine is a table or function by using information such as the length of the edit routine name and the existence of an edit message. If the migration tool still cannot make a determination, it uses the validator property. There will only be an error in the Problems view if the validator is not a function or cannot be found.

The migration tool uses all the available parts in the migration set to resolve ambiguous situations. To minimize these ambiguous situations, always include associated parts when you migrate. For example, when you migrate a program, be sure to include all the parts that you need to generate the program in VAGen. This ensures the best possible migration of your parts. For an overview of how the migration tool resolves ambiguous situations, see the following sections:

- “Migrating with a program” on page 34
- “Migrating with associated parts” on page 35
- “Migrating without associated parts” on page 35

See Chapter 3, “Handling ambiguous situations,” on page 43 for a complete list of the situations where the migration tool must do cross-part migration to achieve a correct migration and the steps the migration tool takes to try to make an intelligent choice if the additional parts are not available.

When and how part names are resolved

At definition time, VisualAge Generator does not require that all parts exist. In the program structure diagram, VisualAge Generator indicates missing maps, records, tables and functions with a question mark. However, in other places such as the

use of a shared data item, there is no indication if the part does not currently exist. When you save a part in VisualAge Generator, there is some basic syntax validation, but there is no cross-part validation until you test, validate, or generate. In EGL, whenever you save a file, there is more extensive validation -- including validation that all part names can be resolved. This gives you the earliest possible warning when there is a problem.

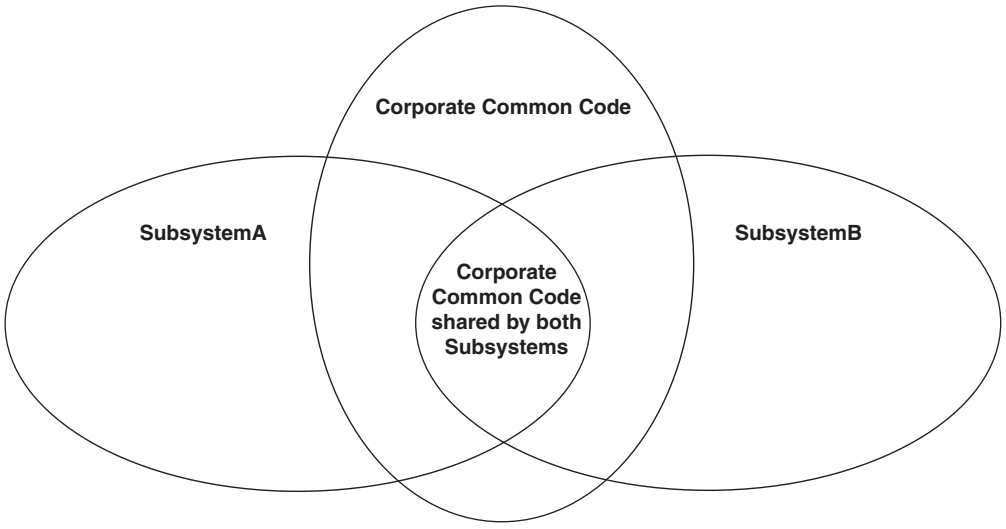
VisualAge Generator searches all parts in the workspace to find a particular part name. If there are duplicate part names in VisualAge for Java, then test and generation stop until the duplicate part problem is fixed. VisualAge for Smalltalk does not permit you to load duplicate parts into the image. In EGL, you are permitted to have duplicate part names in your workspace. EGL uses a combination of the EGL build path for a project, import statements in a file, and the containerContextDependent property for records and functions to determine which definition of a part to use.

The migration tool sets the EGL build path for projects and includes import statements in files based on the available parts in the migration set. To obtain the correct EGL build path and import statements, always include all the associated parts when you migrate. For example, when you migrate a program, be sure to include all the parts that you need to generate the program in VisualAge Generator. This ensures the best possible migration of your parts. See the following sections for more details:

- "EGL build path and import statements" on page 29
- "containerContextDependent Property" on page 30

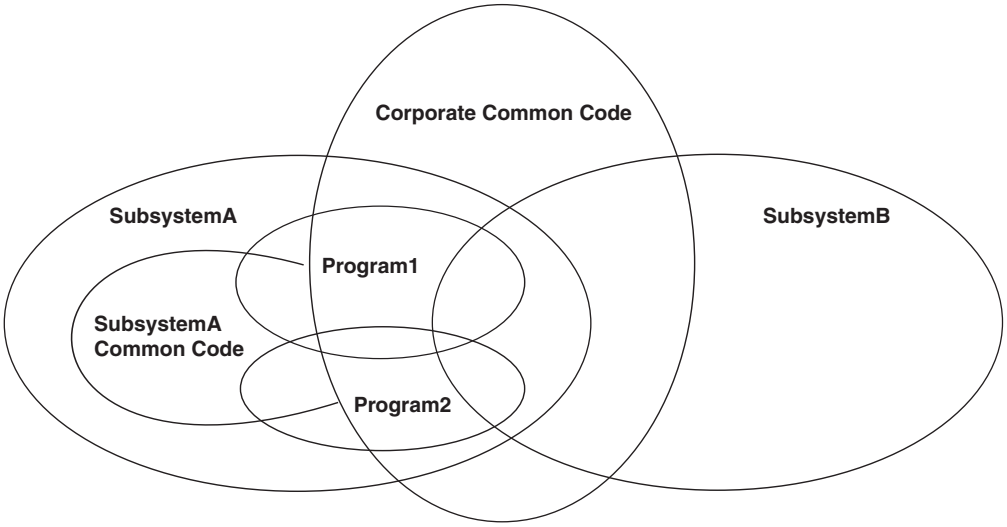
Common code scenarios

Common code is code that is shared between subsystems or programs. The following figure shows common code that is shared by two subsystems.



In this case, there are one or more Java projects or Smalltalk configuration maps that contain Corporate Common Code. The code in these projects or configuration maps can be shared by multiple subsystems. In this example, SubsystemA and SubsystemB use subsets of the common code. Some of the Corporate Common Code is used by both subsystems. For example, Corporate Common Code might include SQL record definitions that are used by many subsystems.

The next figure shows the same basic sharing of Corporate Common Code by the two subsystems, with SubsystemA shown in more detail.



SubsystemA has SubsystemA Common Code that is used by multiple programs within SubsystemA, but only by programs within SubsystemA. In this case, Program1 and Program2 each make use of some of the SubsystemA Common Code as well as some of the Corporate Common Code. Between the two programs, there is some overlap of both the SubsystemA Common Code and the Corporate Common Code, including overlap with Corporate Common Code that SubsystemB uses. For example, SubsystemA Common Code might include SQL record definitions that are used only by programs within SubsystemA. SubsystemA Common Code might also include a map group definition that is used by several programs within SubsystemA.

Common code and VisualAge Generator

To facilitate the use of common code, VisualAge Generator determines at test and generation time how a particular piece of source code should be interpreted. The advantage of this is that each subsystem or program can make slight variations in the code, just by varying the specific map group that a program uses or by varying data item or record definitions that are in the workspace during generation. The following are three examples:

- Much of the same logic can be shared by an online program that interacts with a terminal and a batch program that prints a similar report as follows:
  - ProgramA is main transaction program using MapGrpA which contains display maps named HEADER, DETAIL, and TRAILER. ProgramA displays the partial HEADER map, displays DETAIL lines in a floating area, and then converses the TRAILER map which contains an input field where the user can request the next report. ProgramA uses the SET HEADER PAGE statement to clear the screen.
  - ProgramB is a main batch program using MapGrpB which contains printer maps named HEADER, DETAIL, and TRAILER. Program B produces a hardcopy version of the same report that ProgramA displays on a terminal. ProgramB displays the partial HEADER map, displays the DETAIL print lines in a floating area, and then displays the TRAILER map at the bottom of the page. ProgramB uses the SET HEADER PAGE statement to force a page eject.
  - The number of lines in the floating area differs between the main transaction and the main batch programs. However, the logic for data retrieval, data manipulation, and displaying the HEADER and DETAIL maps is the same for both programs. Because of this, ProgramA and ProgramB were designed to

use common functions to retrieve data from the database, manipulate the data, and display the HEADER and DETAIL maps.

  - This common code technique works in VisualAge Generator because the same DISPLAY I/O option can be used for both display and printer maps. In addition, the same SET HEADER PAGE statement can be used for both display and printer maps. VisualAge Generator interprets the DISPLAY I/O option and the SET map PAGE statement based on the specific program it is testing or generating.
  - EGL requires different statements for display and print forms -- display and clearScreen for a text form; print and pageEject for a print form. In VisualAge Generator Compatibility mode, the display statement is tolerated for a print form. However, clearScreen only applies to text form output, even in VisualAge Generator Compatibility mode.

- A less typical example is the use of a common error handler function called SET-MESSAGE-TEXT which retrieves message text from a VAGen table called MSGTBLE and stores it in a function parameter called MESSAGE-TEXT, where MESSAGE-TEXT is a shared data item.
  - Assume that SubsystemA and SubsystemB run in different CICS regions. In this case, the two subsystems can each provide their own definition of the MSGTBLE and their own definition of the MESSAGE-TEXT function parameter. This might occur if the subsystems provide different size error message fields on their respective map definitions.
  - VisualAge Generator uses the definition that is currently loaded in the workspace when it generates a program. As long as each subsystem always loads its own definition of the MESSAGE-TEXT data item into the workspace before test or generation, VisualAge Generator will use the definition that is correct for that subsystem. The disadvantages of this technique are that you must control what is in the workspace when you generate and you cannot have both subsystems in the workspace at the same time.
  - EGL permits you to have both subsystems in the workspace at the same time. In this situation, EGL uses a combination of the EGL build path, import statements, and the containerContextDependent property for the SET-MESSAGE-TEXT function to resolve the reference to the MESSAGE-TEXT data item definition.
- A slightly different example is the use of a common error record called ERROR-RECORD which contains a shared data item called MESSAGE-TEXT2.
  - Assume that SubsystemA and SubsystemB have different definitions of MESSAGE-TEXT2. This might occur if the subsystems need to build message text for different screen sizes.
  - VisualAge Generator uses the definition that is currently loaded in the workspace when it generates a program. As long as each subsystem always loads its own definition of MESSAGE-TEXT2, VisualAge Generator will use the definition that is correct for that subsystem. The disadvantages of this technique are similar to the SET-MESSAGE-TEXT function example. You must control what is in the workspace when you generate and you cannot have both subsystems in the workspace at the same time.
  - EGL permits you to have both subsystems in the workspace at the same time. In this situation, EGL uses a combination of the EGL build path, import statements, and the containerContextDependent property for the ERROR-RECORD to resolve the reference to the MESSAGE-TEXT2 data item definition.

Common code and the migration tool

Common code is generally code that is used in many programs. You need to include the common code in every migration set because it influences the migration tool in the following ways:

- If common code is available, the migration tool is able to resolve most ambiguous situations. This minimizes or eliminates the code changes you must do manually.
- If common code is available, the migration tool can properly set the EGL build path for projects and include the correct import statements for your EGL files. This minimizes the number of EGL build path changes and additional import statements you must do manually.
- The first time the migration tool migrates a part version, the tool stores the EGL created for the part into the migration database. The original External Source Format is also retained in the migration database. If another migration set uses the same part version, the migration tool uses the original External Source Format for reference when creating EGL for the new parts in the additional migration set, but does not convert the part to EGL again. The migration tool also uses the EGL for the part version when building the EGL projects, packages, and files for the additional migration set. This technique provides the necessary reference information for the migration tool to resolve ambiguous situations during cross-part migration, while improving performance by only migrating each part version one time.

To ensure the best possible migration, when you are migrating a subsystem, you should always include Corporate Common Code and the Subsystem Common Code in your migration set.

Techniques used by the VisualAge Generator to EGL Migration Tool

Overview of techniques

There are a number of techniques that the migration tool uses to determine the corresponding EGL syntax and to preserve the VisualAge Generator behavior. These techniques are as follows:

- Editor and build descriptor preferences
- Program properties
- EGL build path and import statements
- containerContextDependent property
- EGL reserved word list
- Placing parts in EGL files
- Migrating with a program
- Migrating with associated parts
- Migrating without associated parts
- Overwriting and merging files.

These techniques are explained in the following sections. There are also some general rules that govern the migration tool.

Editor and build descriptor preferences

Before you start Stage 2 of migration, you should turn on the *VisualAge Generator Compatibility* preference for your the Rational Developer product workspace. The *VisualAge Generator Compatibility* preference provides support for the following VAGen behaviors:

- Use of the hyphen (-) and national language characters @, and # in part names
- The primitive data types **numc** and **pacf**
- Defaulting the array index to 1 for single dimension arrays
- The **deleteAfterUse** property on **use** declaration for a data table, which is the replacement for keep after use
- The SQL item property **sqlDataCode**
- The **call** statement options of *externallyDefined* and *noRefresh*, which are the replacements for the NONCSP and NOMAPS options
- The **transfer** and **show** statement *externallyDefined* option, which is the replacement for the NONCSP option on DXFR and XFER statements
- A display printForm statement is implemented the same as a print printForm statement
- The initial value of a form field is used only when displaying a field on the screen that has not had a value assigned to it. The preference does not set the initial value of the field in storage.
- The **handleSysLibErrors** system variable, which is the replacement for EZEREPLY
- The **handleHardDliErrors** system variable which is the replacement for EZEDLERR
- The **getVAGSysType** system function, which provides the old VAGen values for EZESYS
- The **connectionService** system function, which is the replacement for EZECONCT
- The **segmentedMode** system variable, which is the replacement for EZESEGM
- The **sqlIsolationLevel** system variable, which is the replacement for EZESQISL.
- Host variables that are not in an SQL record are initialized to blanks or zeros depending on the primitive type if the value retrieved from the database is null. Host variables in SQL records are only initialized if the *isNullable=yes* property is set for the data item.
- Even precision for decimal fields (VAGen PACK fields) is incremented by 1 except for host variable references in SQL WHERE clauses and the EGL *prepare* statement.

The VAGen migration tool automatically adds the vagCompatibility=″YES″ option to every VAGen generation option part that it migrates to an EGL build descriptor. The vagCompatibility build option directs generation to provide the same support as the *VisualAge Generator Compatibility* preference.

Program properties

There are two program properties that the migration tool includes for every program:

- *includeReferencedFunctions = yes.* The migration tool always includes this program property so that functions do not have to be nested within the program. This enables you to keep just one copy of common functions in a separate project or package and import them, rather than including the common functions in each

program. When you use Stage 1 – 3 migration, the migration tool also includes any necessary import statements for functions that are in a different package from the program.

- *allowUnqualifiedItemReferences = yes.* The migration tool always includes this program property so that references to data items do not need to be qualified. The EGL rules for unqualified data items incorporate the VAGen rules so that unqualified items resolve to the same record, table, or map (form) as in VisualAge Generator. The migration tool does not add qualifications.

EGL build path and import statements

EGL enables you to have multiple definitions for a part in the workspace at the same time. The EGL build path for a project limits which other projects will be considered when looking for a part name. The import statement in a file determines which packages, other than the current package, and which parts within the EGL build path will be considered when looking for a part name.

In most situations, the EGL build path and import statements are sufficient to resolve any part references. For example, the EGL build path and import statements for a program are sufficient to resolve a record name if you use the record as a type definition in a record declaration in a program. The EGL build path and import statements are also sufficient to resolve data item references if you only have one definition of the item that can be used with a record definition, function local storage or function parameter list.

For example, you might be working on SubsystemA and SubsystemB which have two different definitions of RECORDX. All programs in SubsystemA need to use the SubsystemA definition of RECORDX. This can be achieved as follows:

- The EGL build path property for projects in SubsystemA needs to include the project that provides SubsystemA's definition of RECORDX.
- Files for programs in SubsystemA that use RECORDX as a type declaration for a record need to include an import statement for the package within SubsystemA that contains the definition of RECORDX.

The EGL build path property for the SubsystemA projects limits the projects that will be searched to just the projects within SubsystemA and the common projects. The import statements in the files within SubsystemA limit which packages within the EGL build path will be considered. Even if RECORDX uses shared data item ITEM1 and the two subsystems have different definitions of ITEM1, the EGL build path and import statements are sufficient to resolve the references to ITEM1. The project that contains RECORDX in each subsystem must specify an EGL build path property that includes the subsystem project that contains that corresponding subsystem definition of ITEM1. The file containing RECORDX in each subsystem must have an import statement that specifies the subsystem package that contains the corresponding subsystem definition of ITEM1.

When you use Stage 1 – 3 migration, the migration tools do the following:

- Set the EGL build path for each project based on the parts the project needs to reference in other projects.
- Include most import package statements for each file based on the parts the file needs to reference in other packages within the EGL build path for the project that contains the file. These import statements are based on the part associates that were determined by VisualAge Generator during Stage 1 of migration.
- The migration tool adds import statements for data item parts. During Stage 2 migration, the migration tool determines if a data item part has an edit routine.

If the table or function specified as the edit routine is included in the migration set, then the migration tool updates the migration database to include the part specified as the edit routine as an associate of the data item.

- The migration tool does not add import statements for the following situations because these are not associates in VisualAge Generator:
  - For a function that transfers to a program using a CALL, DXFR, or XFER statement. If you are generating for Java, you must add the import statement for the package containing the program within the file containing the function or fully qualify the program name with the package name. Alternatively, for the CALL statement, you can use an entry in a linkage part to specify the name of the package where the program is located.
  - For build parts in *.eglbld* files. VAGen control parts, such as the generation options parts, do not list their associated parts, so the information is not readily available to the migration tool. In addition, due to the way EGL processes build descriptor parts, you will likely need to do some reordering of the *nextBuildDescriptor* values (VAGen /OPTIONS). This reordering will in turn require modification of any imports the migration tool might have done.

When you use single file migration, the migration tool functions as follows:

- Does not set the EGL build path because the output is always going into a single existing project.
- Does not include the import package statements because all the files that are created go into the same package. Thus, no import statement is required.

containerContextDependent Property

Note: This section describes a capability that is only partially implemented in EGL version 6.0. If you specify containerContextDependent for a function, the resolution of function invocations within that function occurs at generation time (not at development time) and includes reference to the name space of the program or page handler that uses the invoking function. At this time, containerContextDependent has no effect on records or items.

The description that follows reflects what is intended for the final implementation.

As described in "EGL build path and import statements" on page 29, the EGL build path and import statements are generally sufficient to provide the part name resolution that you need. However, EGL expects to resolve all part name references whenever you save a file. EGL adds an error message to the Problems view if it cannot resolve the part name. Depending on your architecture, you might also need to use the containerContextDependent property for records or functions.

Consider the situation where RECORDX is used as the type definition for a function parameter in FUNCTIONY. Assuming that RECORDX and FUNCTIONY are in different projects and packages, EGL expects the following:

- The EGL build path for the project that contains FUNCTIONY must include the project that contains the definition of RECORDX
- The file containing FUNCTIONY must include an import statement for the package that contains RECORDX.

If all subsystems have the same definition of RECORDX, then the EGL build path and import statements are sufficient, and EGL can resolve the part reference for RECORDX whenever you save the file containing FUNCTIONY.

However, consider the situation in which SubsystemA and SubsystemB both use FUNCTIONY, but have different definitions of RECORDX. In this situation, the EGL build path and import statements cannot point to both subsystems at the same time. EGL supports the containerContextDependent property for functions. In this situation, you can specify *containerContextDependent=yes* for FUNCTIONY. This specifies that the part name references for the function parameters and local storage are not to be resolved until FUNCTIONY is used within a program. When you test or generate a program that uses FUNCTIONY, the EGL build path of the project containing the program and the import statements of the file containing the program determine where to find the definition of RECORDX. Using *containerContextDependent=yes* enables you to achieve the same flexibility provided by VisualAge Generator for the function. The EGL build path for each project in the subsystem and the import statement for any files containing programs in the subsystem point to that subsystem's definition of RECORDX.

The containerContextDependent property is also supported for records. For example, SubsystemA and SubsystemB might both use the same definition of RECORDZ. However, RECORDZ uses shared data item ITEM1 and the subsystems have different definitions of ITEM1. In this case, you can specify *containerContextDependent=yes* for RECORDZ so that EGL validation will not attempt to resolve ITEM1 until RECORDZ is used in a program. The EGL build path of the project containing the program and the import statements of the file containing the program determine where to find the definition for ITEM1.

The migration tool does not attempt to set the containerContextDependent property for you. This is because the migration tool does not require that you migrate all you subsystems at the same time and does not do a complete analysis of all definitions of all parts to determine when there are duplicate part definitions. You can add the containerContextDependent property as necessary if you determine that there are duplicate part names that need to be resolved at test and generation time (as in VisualAge Generator) rather than at definition time (as in EGL).

EGL reserved word list

EGL has a reserved word list. Parts cannot be named the same as any EGL reserved word. In addition, EGL does not permit the use of the # symbol as the first character of an EGL part name. The # symbol is permitted as the first character of an EGL build part name. If a VAGen part is named the same as an EGL reserved word, or if a VAGen part starts with the # symbol, the migration tool does the following based on the part type:

- The migration tool does not rename programs, map groups, or tables because these parts frequently have references from non-VAGen programs or the runtime environment (for example, a CICS PROGRAM definition).
- The migration tool renames data items, records, maps, and functions by prefixing the part name with a Renaming Prefix. The Renaming Prefix is one of the VAGen Migration Syntax Preferences that you can specify for Stage 2 or single file mode migration.
- The migration tool does not rename control parts, except for the following:
  - The migration tool removes the .BND suffix from the end of a bind control part name.
  - The migration tool removes the .LKG suffix from the end of a link edit part name.

– The migration tool changes any other dots to underscores in control part names. The tool also changes dots to underscores in control part names that are referenced in the /OPTIONS, /RESOURCE, and /LINKEDIT generation options.

The Stage 1 migration tools provide a list of the program, map group, table, and control part names that conflict with the EGL reserved word list. If you do not rename these parts before you migrate, the Stage 2 migration tool (or single file mode) will also issue an error message. There will be an error in the Problems view. You can also correct the problem in your Rational Developer product by renaming the program, form group, or data table and optionally using the EGL *alias* property.

Placing parts in EGL files

When you migrate using Stage 1 – 3 migration, each Java package or Smalltalk application migrates to the corresponding EGL package based on your Stage 1 renaming rules. The VAGen parts within the original Java package or Smalltalk application are placed in one or more EGL files within the corresponding EGL package based on the following:

- The type of part:
  - Generatable part -- program, table, or map group
  - Control part -- generation options, resource associations, linkage table, linkedit, or bind control
  - Other migratable parts -- data item, record, map, or function
  - Non-migratable parts – PSB and DL/I records are not migrated to any file.
- A Stage 1 preference that enables you to identify Java project or package names that contain common parts. Similarly, there is a Stage 1 preference for Smalltalk that enables you to identify configuration map or application names that contain common parts.
- Whether the part is used by some other part. The Stage 1 migration tool determines whether a part is used based on the following:
  - A part is ″used″ if it appears on the VAGen associates list of any generatable part **in the migration set**.
  - A part is ″used″ if it is in a common Java project or package or in a common Smalltalk configuration map or application as specified in your Stage 1 preferences.

The Stage 1 migration tool determines the placement of all parts. The Stage 1 migration tool places VAGen parts within a single Java package or Smalltalk application into EGL files within the corresponding EGL package as follows:

- All control parts are placed in a single file called eglPackageName.eglbld, where eglPackageName is the name of the corresponding EGL package.
- Each program part is placed in a file called programName.egl, where programName is the name of the program.
- Each table part is placed in a file called tableName.egl, where tableName is the name of the table.
- Each map group and all maps in the map group are placed in a file called mapGroupName.egl, where mapGroupName is the name of the map group. If there is no map group part, the Stage 1 migration tool creates a dummy map group part. Because the map group and all maps in the map group must be placed in the same file, these parts must be considered as a group. This can result in some parts being moved to a different EGL package or project if the

parts were not originally in the same Java package or Smalltalk application. The migration tool determines where to place the mapGroupName.egl file as follows:

- If the map group and all its maps are **in the same Java package**, the mapGroupName.egl file is placed in the corresponding EGL package. Similarly, if the map group and all its maps are in the same Smalltalk application, the mapGroupName.egl file is placed in the EGL package that corresponds to the Smalltalk application. In this situation, the migration tool handles the mapGroupName.egl file in the same manner as the program and table files. This is the most common situation.
- If the map group and its maps are spread across **several Java packages within a project**, then the project name, plus a suffix is used to create the name of a new EGL package to contain the mapGroupName.egl file. This new EGL package is placed within the original project. Similarly, if the map group and its maps are spread across several Smalltalk applications within a configuration map, the configuration map name, plus a suffix is used to create the name of a new EGL package to contain the mapGroupName.egl file. For both Java and Smalltalk, you can control the suffix with a Stage 1 preference.
- If the map group and its maps are spread across **several Java projects**, then the migration set name, plus a suffix is used to create the name of a new EGL project that contains the mapGroupName.egl file. Similarly, if the map group and its maps are spread across several Smalltalk configuration maps, the migration set name, plus a suffix is used to create the name of a new EGL project that contains the mapGroupName.egl file. For both Java and Smalltalk, you can control the suffix with a Stage 1 preference.

• All the remaining parts are placed as follows:

- If the part is **used by only one** program in the migration set, the part is placed as follows:
  - If the part is in the same package as the program, then the part is placed in the same file as the program. For example, the main function of a program (ProgramA-MAIN) is placed in the same file as the program (ProgramA) provided the function is not used in any other programs. The file is named for the program – ProgramA.egl.
  - If the part is in a different package from the program that uses it, the part is placed as follows:
    - If the part is in a common project or package, the part is placed in the file called commonParts.egl in the part's original package. You control the name for *commonParts* with a Stage 1 preference.
    - If the part is not in a common project or package, the part is also placed in the file called commonParts.egl in the part's original package.
- If the part is **used by several** programs in the migration set, then the part is placed in the file called commonParts.egl within its original package. For example, if ProgramA calls ProgramB and passes RecordR, then RecordR is placed in the file called commonParts.egl in the EGL package that corresponds to the original Java package or Smalltalk application that contains RecordR.
- If the part is **not used** by any programs in the migration set, the part is placed into a file as follows:
  - If the part is in a common Java project or package, then the part is placed in the file called commonParts.egl within the EGL package that corresponds to the original Java package that contains the part. Similarly, if the part is in common Smalltalk configuration map or application, then the part is placed

the file called commonParts.egl within the EGL package that corresponds to the original Smalltalk application that contains the part.

- If the part is not in a common Java project or package, then the part is placed in the file called unusedParts.egl within the EGL package that corresponds to the original Java package that contains the part. Similarly, if the part is not in a common Smalltalk configuration map or application, then the part is placed in the file called unusedParts.egl within the EGL package that corresponds to the original Smalltalk application that contains the part. You control the name for unusedParts with a Stage 1 preference.

- There are the following special considerations:
  - Any function used as an edit routine on a map contributes to whether the function is used or not used. However, the migration tool always places the function with either the program or in the commonParts file. The migration tool never places the edit function in a file that is created for the map group.
  - Any shared item that is used in a table contributes to whether the data item is used or not used. However, the migration tool always places the data item with either the program or in the commonParts file. The migration tool never places the data item in a file that is created for the table.

Note: If you migrate multiple migration sets or migration set versions without clearing out the migration database, **the first migration set version processed in Stage 1 that contains a part edition controls the project, package and file name for the EGL part.** To ensure that parts are placed according to the definition of each migration set version, you should clear out the migration database between versions.

The Stage 1 migration tools for Java and Smalltalk are provided as sample code. You can modify the Stage 1 migration tools to place parts differently based on your own library management philosophy. For example:

- If ProgramX calls ProgramY and passes records ProgramY-Parm and Common-Parm, you might want ProgramY-Parm to be placed in the file with ProgramY and Common-Parm to be placed in the commonParts file. Given knowledge of your naming conventions, you can modify the Stage 1 migration tool to change the file placement algorithm.
- For large packages, you might want to split the parts into separate files by part type.
- If the same part edition appears in multiple migration set versions, but should be placed in different EGL projects, packages, or files depending on the migration set version, you might want to update the migration database for the new EGL project, package, and file name for each part whenever you process a migration set version. If you make this change, be sure to process each migration set version completely through Stages 1 – 3 before starting to migrate the next migration set version.

Migrating with a program

Normally when you migrate, you specify a migration set that identifies all the Java projects or Smalltalk configuration maps that should be migrated as a group. Using the migration set, the migration tool migrates programs and their associates first. This enables the tool to use the context of a specific program to assist in resolving situations where the EGL language is more precise or more restrictive than VisualAge Generator. The first program to migrate, together with its associates, determines the EGL syntax for any ambiguous situation within that program or its associates. A different program might result in a different resolution for the same ambiguous situation in a shared data item, common record, map, table or function.

Because a part version is only migrated once, the first program that uses the common part controls the resolution of any ambiguous situation for its associates.

Consider the example in which ProgramA is a main transaction program using display maps and ProgramB is a main batch program using printer maps. The programs share common functions that display the HEADER and DETAIL maps. The common functions also use the SET map PAGE statement to clear the screen or force a page eject. In this case, if ProgramA migrates first, the migration tool creates the EGL source for the functions to use the display statement and clearScreen system library function. If ProgramB migrates first, the migration tool creates the EGL source to use the print statement and the pageEject system library function.

Whenever you migrate programs and their associates, the first program that uses a shared data item, common record, map, table, or function controls the resulting EGL code. In most cases, because the programs use the common code in the same way, this technique provides the most appropriate migration of your VAGen source. However, as you can see from this example, the specifics of what you intended the common code to accomplish might not be reflected in the resulting EGL source. In this example, regardless of which program migrates first, you will not be able to test or generate the program that migrates second. In VisualAge Generator Compatibility mode, you can use the display statement to resolve the problem with the I/O statement. However, to resolve the problem with the choice of clearScreen or pageEject might require adding a new variable, TEXT-OR-PRINT, that each program initializes and which the common function tests to determine whether to execute the clearScreen or pageEject system library function.

Migrating with associated parts

If a program and its associates are not available, the migration tool makes use of all the parts that are available in the migration set (or in the External Source Format file if you are migrating in single file mode). In this case, if the additional part that is needed for cross-part migration is available, the migration tool can make a decision with a high probability that it is the correct choice.

Consider the example in which a map variable field specifies an edit routine. If a VAGen table that is named the same as the edit routine is available in the same migration set (or the External Source Format file), then the migration tool assumes that this is the table that would always be used and migrates to the validatorTable property. If there is a function that is named the same as the edit routine, then the migration tool migrates to the validator property. In either case, because there is a part with the same name as the edit routine, the migration tool has a high probability that it made the correct choice. If a table or function with the same name as the edit routine is not available, then the migration tool processes the map variable as though it was migrating without associated parts.

In many cases, migration with associated parts can provide very similar migration to what you would achieve when you migrate programs with their associates. The disadvantage of migrating without the program is that you can quickly shift from migrating with associated parts to migrating without associated parts even within a single function based on the specific statement that is being migrated and the other parts that are included in the file.

Migrating without associated parts

Sometimes even when a program is available, not all of its associates are included in the migration set. Or you might be migrating some common parts that were

used in the past by a subsystem, but which are not currently in use. In this case, the associates of a part that is being migrated might not be available. The migration tool still converts the part using one of the following techniques:

- **Flexibility in EGL syntax.** For example, a DISPLAY I/O option is migrated without an associated map. In this case the migration tool makes the choice of using a display statement and includes a warning message in the migration log. Even if the migration tool guessed incorrectly, because you use VisualAge Generator Compatibility mode, the display statement will be accepted even if the form is a print form.
- **Intelligent guess.** For example, a map variable field specifies an edit routine, but there is no part named the same as the edit routine in the migration set (or the External Source Format file). In this case, the migration tool makes use of other information. The tool looks at the following to try to determine whether to use the validatorTable or validator property:
  - Length of the edit routine name, because 8 or more characters indicate it is a function.
  - Edit routine name is EZEC10 or EZEC11, which indicate it is a function.
  - Edit message is also specified, because the message can only be used with an edit table or EZEC10 or EZEC11.

  Any of these enable the migration tool to make an intelligent choice between setting the validatorTable or validator property. If there is nothing to make a definitive choice, the migration tool uses the validator property and includes an error message in the migration log. If the migration tool guessed incorrectly there should also be an error in the Problems view.
- **Deliberately invalid syntax.** For example, a SET map PAGE is migrated without an associated map. In this case, the migration tool could make the choice between using an EGL sysLib.clearScreen function for a text form and an EGL sysLib.pageEject function for a print form. However, both choices are equally probable. Therefore, the migration tool creates intentionally invalid syntax and converts to sysLib.EZE_SETPAGE. This results in an error in the Problems view and forces you to correct the problem.
- **Direct conversion without information due to missing associates.** (The missing associates can result in problems undetectable by the migration tools.) For example, RecordA specifies that it is redefining the storage of RecordB. In VisualAge Generator, the redefinition information is stored in the record definition for RecordA. When you generate, RecordA and RecordB must be available and the redefinition is done for the RecordA in the program. In EGL the redefinition information is only stored in the program. If RecordA is not available when migrating the program, the migration tool has no way to detect that RecordA needs to include the redefines property within the program. Without the redefines property, EGL test and generation treat RecordA and RecordB as separate data areas. **The program will not run the same as in VisualAge Generator** -- data might not be initialized correctly and abends could occur. This is why we strongly encourage you to generate and test your migrated programs.

Overwriting and merging files

The Stage 2 and 3 migration wizards provide several related preferences that control processing for multiple versions of the same migration set. These preferences are as follows:

- Migrate remaining VAGen parts.
- Import into workspace -- with or without Override existing files.
- Save migrated files to temporary directory.

Migrate remaining VAGen parts controls whether the migration tool converts all parts in the migration set to EGL.

- If you do not select *Migrate remaining VAGen parts*, only generatable parts and their associates are converted into EGL and stored in the migration database. Data items, records, and functions are not converted unless they are an associate of one or more generatable parts. Control parts are not converted. Deselecting *Migrate remaining VAGen parts* can be useful if you are migrating a subsystem project and a common project in a single migration set. In this situation, the following happens:
  - For the subsystem project, only parts that are actually used within the subsystem are converted.
  - For the common project, any generatable parts and their associates are converted. In addition, any data items, records, and functions that are used by the subsystem are also converted. Other data items, records, and functions that might be used by other subsystems but which are not used by the current subsystem are not converted to EGL.

  There are two advantages of deselecting *Migrate remaining VAGen parts*:
  - For the subsystem project, you have the opportunity to clean up your code because the migration tool only converts parts that are actually used.
  - For the common project, you can defer converting parts until they are actually used by another subsystem. When you include the common project in the migration set for another subsystem, any additional parts used by this subsystem are converted to EGL and stored in the migration database. This is particularly useful if your common project has associates in various subsystems or contains parts that are associates of generatable parts in various subsystems. Deferring the migration of the common parts until a subsystem uses the part enables the common parts to migrate "with associates." When you migrate the next migration set that contains the common project, your selection for *Override existing files* controls what happens to the originally migrated common parts files.
- If you select *Migrate remaining VAGen parts*, the generatable parts and their associates are converted to EGL and stored in the migration database. Then any data items, records, and functions that are not associates of generatable parts are converted to EGL. All control parts are also converted to EGL. There are two advantages of selecting *Migrate remaining VAGen parts*:
  - For the subsystem project, all the parts are converted to EGL regardless of whether they are used or not. This is useful if you must preserve code for historical purposes.
  - For the common project, selecting *Migrate remaining VAGen parts* is particularly useful if you know that the parts in the common project do not have associates in the subsystem projects that you plan to migrate in the future. You can convert all the common parts one time and have the EGL stored in the migration database. Then if the common project is included in the migration set for other subsystems, the EGL is already converted and available to be imported into the workspace or saved into the temporary directory with the new subsystem.

  If you select *Migrate remaining VAGen parts* for your first migration set version, you should continue to select *Migrate remaining VAGen parts* for other migration set versions. You should also specify *Override existing files*. By specifying both options you ensure that all the parts in the migration set are included in the EGL file.

Import into workspace controls whether the migration tool builds the EGL projects, packages, and files in your workspace. If you select *Import into workspace*, there are additional options that you can select.

- If you are migrating multiple versions of a migration set, you can choose which version to have imported into your workspace at the end of migration. You can choose either the *Latest version* (most recent version) or the *Oldest version*. The advantage of selecting the latest version is that this is the version which you are most likely to want to generate for additional testing. The advantage of selecting the oldest version is that this positions you to store the EGL projects, packages, and files that correspond to the oldest version into your source code repository first.
- You can specify how you want to handle the situation in which an EGL file that is being created by the current migration already exists in your workspace.
  - If you select *Override existing files*, the EGL file is replaced by a new file containing **only** parts in the current migration set. The migration tool does not convert VAGen part editions again if they were already converted for a previous migration set. However, the migration tool does include the EGL for the part editions in the file if the parts are included in the current migration set. Select *Override existing files* if you decide to change your VAGen Migration Syntax Preferences. In this situation, you could restore your database to the end of Stage 1 and then run Stage 2 and 3 again with your new preferences. Specifying *Override existing files* enables you to run Stage 2 and 3 without having to clean out the EGL workspace first. Selecting *Override existing files* is also useful if you specify *Migrate remaining VAGen parts*. In this situation, if you migrate another version of a migration set, the EGL files are replaced by new files containing the part editions that are included in the current migration set version.
  - If you do not select *Override existing files*, the existing EGL file is modified to contain any additional parts that are in the current migration set. **Parts that are already in the EGL file are not changed, even if the current migration set uses a different part edition.** Deselecting *Override existing files* is useful only if you do not specify *Migrate remaining VAGen parts* and you are migrating just one version of a common project, but with several different subsystems. In this situation, you can gradually build up the EGL files for a common project in stages as you migrate different subsystems. Only the common parts used by the first subsystem are initially included in the EGL file. When you migrate the second subsystem, the migration tool adds any additional parts required for the second subsystem into the EGL file. The migration tool does a merge of the original file and the additional parts so that the parts continue to be organized alphabetically within part type.

You can select *Import into workspace* when you are migrating multiple versions of a migration set. However, the better technique is to deselect *Import into workspace* and instead select *Save migrated files to temporary directory*. Using the temporary directory enables the migration tool to create all the migration set versions.

Save migrated files to temporary directory enables you to migrate multiple versions of a migration set and store all the versions outside the workspace. This option requires the use of multiple simultaneous instances of the Rational Developer product. Therefore, due to the large amount of memory resource required, it is recommended only for batch mode. When you select *Save migrated files to temporary directory*, you must also specify a high level directory. The migration tool creates a subdirectory for each migration set version within this high level directory. *Save migrated files to temporary directory* works particularly well if you also specify *Migrate remaining VAGen parts*. In this situation, each

subdirectory corresponding to a migration set version contains **all** the parts from the VAGen project versions that are included in the migration set version.

General rules

There are some general rules that govern what the migration tool does when migrating your source code. The rules are as follows:

- Any new variables that are added to support the EZE words or other statements must be added to the program. They cannot safely be added as local storage in a function. This is because a segmented converse is not supported if any function currently in the stack has local storage, parameters, or a return value. Because the function context is not known, there is no way to determine whether the function would be in a function stack that leads to a segmented converse. Therefore any new variables are added to the program, not as function local storage. See "Intermediate variables required for migration" on page 65 for details.
- You might have parts from Cross System Product or older releases of VisualAge Generator that were migrated to VisualAge Generator 4.5, but never modified within VisualAge Generator 4.5. In some cases, information is missing from the External Source Format or is specified in a way that is not supported in EGL. The migration tool attempts to supply the missing information or correct the information. This includes the following:
  - For main transactions, if the VAGen segmentation information is missing, the migration tool defaults the EGL segmented property to *no*.
  - For SQL records, if the SQL data code is missing, the migration tool converts the item to a fixed length item.
  - For SQL functions, the migration tool attempts to create any missing required SQL clauses based on the record specified as the I/O object.
- Within a function, the migration tool converts all statements to something. This preserves your IF / ELSE / END and WHILE / END logic. However, the resulting statements might not be syntactically correct. For example:
  - If an EZESYS value is not supported in EGL, the migration tool uses the VAGen value. There will be a message in the Problems view if the value is not supported.
  - EZE DL/I status words (EZEDL*) are migrated to EGL system variables even though DL/I is not currently supported by EGL. The EGL system variables are a "best guess" as to the possible system variable names. The migration tool issues an error message. There will also be an error in the Problems view.
  - The CONVERSE I/O option is always migrated to an EGL *converse* statement even though it might be used with a UI record. The migration tool does not issue a warning message. If the CONVERSE is for a UI record, there will be an error in the Problems view because the UI record cannot be migrated.
  - The XFER statement is always migrated to an EGL statement even though it might be used with a UI record. The migration tool issues an error message. There will also be an error in the Problems view.
  - The EZESCRPT special function word is commented out. There is no corresponding EGL replacement. Because EZESCRPT could not be used in an IF, WHILE, or TEST statement the structure of your function's logic is preserved. The migration tool issues an error message. There will not be an error in the Problems view.

- When trying to resolve a part reference without a program, the migration tool looks at the parts in the migration set. Therefore it is important that you define your migration sets to include groups of projects that are reasonable to use together. For example:
  - Do not include projects from several subsystems that have conflicting part names.
  - Do include common Java project or common Smalltalk applications when migrating a subsystem.
- There are some things the migration tool does **not** do during migration:
  - The migration tool does not add import statements for the following situations because these are not associates in VisualAge Generator:
    - For a function that transfers to a program using a CALL, DXFR, or XFER statement. If you are generating for Java, you must add the import statement for the package containing the program within the file containing the function or fully qualify the program name with the package name. Alternatively, for the CALL statement, you can use an entry in a linkage part to specify the name of the package where the program is located.
    - For build parts in *.eglbld* files. VAGen control parts, such as the generation options parts, do not list their associated parts, so the information is not readily available to the migration tool. In addition, due to the way EGL processes build descriptor parts, you will likely need to do some reordering of the *nextBuildDescriptor* values (VAGen /OPTIONS). This reordering will in turn require modification of any imports the migration tool might have done.
  - EGL does not permit implicit items. VisualAge Generator permits implicit items, but does not recommend them. Implicit items are items that are used in a program, but which are not explicitly defined in a record, table, or map used by the program. The migration tool does not create definitions for implicit items due to the performance impact of evaluating every unqualified data item to determine whether it is an implicit item. You should provide definitions for implicit items before you migrate. To resolve the problem before you migrate, validate the program in VisualAge Generator to determine whether the program actually used implicit items. If implicit items were used, VAGen validation will provide a message with the correct definition of the item. If you do not create a definition for the implicit item before you migrate, there will be an error in the Problems view and you can correct the problem in the Rational Developer product.
  - The migration tool does not attempt to set the containerContextDependent property. This is something you can add later to specific common records or functions that have the need to reference other parts that are provided by a subsystem. See the section "containerContextDependent Property" on page 30 for more details of how to use this property for records and functions.
  - The migration tool assumes that the VAGen syntax is valid and that a program using the parts included in your migration set can be successfully validated in VisualAge Generator. The migration tool does not attempt to repair invalid syntax. For example:
    - If the elements of a map array have different edit characteristics or attributes, the characteristics for the first element of the array determine what is migrated to EGL. The migration tool does not issue a message.
    - If the lengths of shared data items in a record have changed so that the length of a parent item does not match the sum of the lengths of items in its substructure, the migration tool does not change any item lengths and

does not issue a message. There will be an error in the Problems view indicating that sum of the lengths of the children does not match the length of the parent.

– The migration tool does not attempt to improve inefficient code even if it results in syntax errors in EGL. For example:
  - If the same record is listed twice in a program's Tables and Additional Records list, the migration tool does not remove it and does not issue a message. There will be an error in the Problems view. Similarly, if the same table is listed twice in a program's Tables and Additional Records list, the migration tool does not remove the extra table and does not issue a message. There will be an error in the Problems view.
  - If a record is not used in a program, but is listed in a program's Tables and Additional Records list, the migration tool does not remove it and does not issue a message. There will not be an error in the Problems view. Similarly, if a table is not used in a program, but is listed in the program's Tables and Additional Records list, the migration tool does not remove the table and does not issue a message. There will not be an error in the Problems view.
  - If a working storage record is listed in the program's Tables and Additional Records list and the record only contains level 77 items, the migration tool does not remove the record and does not issue a message. There will be an error in the Problems view indicating the record cannot be found. This is because the only record that now exists includes your Level77 suffix preference as part of the record name.
  - If a VAGen program includes a map group or a help map group, an actual map group part did not have to exist. For example, if the program never DISPLAYs or CONVERSEs a map or if the program never uses a map as a called parameter, an actual map group part did not have to exist. In this situation, the migration tool includes the use statement for the formGroup, but does not include the import statement in the program because the map group was not an associate of the program in VisualAge Generator. The migration tool does not issue an error message. EGL requires an actual formGroup part. If there is no formGroup part or if the formGroup part is not in the same package as the program, there will be an error in the Problems view indicating that the formGroup specified in the program's use declarations cannot be found.

– The migration tool does not necessarily detect or provide warning messages for the use of facilities that were not documented in VisualAge Generator, even if there is no equivalent EGL support. For example:
  - The VAGen EZEBYTES function is only documented to support items and records. There was undocumented support for maps. The EGL sysLib.bytes function only supports items and records. The migration tool does not provide a warning message if you used EZEBYTES for a map. There will be an error in the Problems view.
  - If a CALL statement specifies an unqualified item as an argument, and there are multiple definitions for this item name within the program, the item is assumed to be a level 77 item in the program's primary working storage record. EGL requires that the item be qualified. The migration tool does not add the qualification for you and does not provide a warning message. There will be an error in the Problems view.

Known restrictions for the migration tools

General

Be sure to review the EGL restrictions, specifically any restrictions for validation, debugging, or generation.

Stage 1 on Java and Smalltalk

- If you migrate multiple migration sets or migration set versions without clearing out the migration database, the first migration set version that contains a part edition controls the project, package and file name for the EGL part. This generally does not cause a problem if the following are true:
  - If your parts do not move between Java packages and you do not have differently named migration sets that include the same Java package name.
  - If your parts do not move between Smalltalk applications and you do not have differently named configuration maps that include the same Smalltalk applications.

  If your situation differs from what is described above, the workaround is to migrate one migration set version all the way through Stages 1 – 3, then clear out the migration database, and then migrate the next migration set version all the way through Stages 1 – 3.

Stages 2 and 3 on the Rational Developer product

- Restrictions for the VAGen Migration wizards:
  - The Cancel button on the progress window is inoperable. You cannot cancel the Stage 2 or 3 migration tool after it starts other than by using the Task Manager.
- If you want to switch back and forth between your migration database and your application databases, you must shut down the Rational Developer product each time you switch.
- Stage 2 and 3 are not supported in batch mode on Linux environments.

Syntax migration

- The migration tool correctly converts SQL keywords used as column names within the SQL record structure. However, the migration tool does not handle SQL keywords used as column names within the SQL defaultSelectCondition for a record or within the SQL clauses for a function. The workaround is to modify the SQL defaultSelect Condition or SQL clause as described in “SQL reserved words requiring special treatment” on page 174. This section provides a list of SQL keywords and details of the syntax required for the SQL defaultSelectCondition and SQL I/O statements.
- The migration tool converts SQL I/O with execution time statement build to a prepare statement. However, if any host variables in the SQL statement are smallint, int, bigint, binary, or decimal fields, the prepare statement is not correct. The workaround is to change the prepare statement by replacing each smallint, int, bigint, binary, or decimal field host variable with the ? symbol and then adding the variable names in a using clause for the execute, open, or get statement that is associated with the prepare statement. Refer to the ″prepare″ topic in the online helps for examples of the correct syntax.
- Parts are sorted alphabetically by VAGen part name within the part type.

Chapter 3. Handling ambiguous situations

There are a number of situations where the migration tool might not have all the information from VisualAge Generator needed to produce the corresponding EGL statement. These situations are called ambiguous situations because the corresponding EGL statement could change or produce different results depending on the inputs from VisualAge Generator. In these ambiguous situations, the migration tool might not migrate to EGL statements that match what you intended in VisualAge Generator. In many of the ambiguous situations, the EGL statements that are produced vary, depending on your migration process:

- Whether you migrate with a program and its associates, and if so, the order in which programs are migrated.
- Whether you migrate without a program, but with the necessary associated parts, so that cross-part migration can still be accomplished.
- Whether you migrate without associates, so that the migration tool is limited in the information it has available to make an intelligent choice.

Migrating with a program and its associates is the preferred way of migrating with associates because it guarantees the maximum information. The tables that follow explain the differences between migrating with and without associated parts for the following part types:

- Shared data items
- Records
- Tables
- Map groups and maps
- Programs
- Functions, including I/O statements
- Other statements
- EZE words

The tables also show some potential problems that can arise for these ambiguous situations and suggest possible solutions for these problems. No one solution is best for every situation. For example, when there are two parts with the same name, renaming the one that is the least frequently used would have the least effect in other areas of your code.

Handling ambiguous situations for data items

Pack data items with even length

VisualAge Generator: The length for pack data items is specified as the number of digits, up to a maximum of 18. Even lengths are recorded within the shared data item definitions and for nonshared data items within record definitions. However, in most editors, and in test and generation, the length that is used is the next higher odd length, with a maximum of 18. Only the SQL Record Editor displays the even length. For even length items used as host variables in SQL WHERE clauses or in SQL statements that specify Execution Time Statement Build, test and generation create a temporary variable with the even length.

EGL: In VisualAge Generator Compatibility mode, EGL test and generation provide the same support as in VisualAge Generator. For decimal items with even precision, test and generation increase the precision by one in all records and use a temporary variable with the even precision in SQL where clauses or prepare statements. If VisualAge Generator Compatibility mode is not specified, EGL uses the precision specified for the data item.

Associated part needed for migration: Not applicable.

Table 9. Pack data items with even length

Migrating with the associated part	Migrating without the associated part
The migration tool migrates pack data items with even lengths as follows: • Uses the even or odd length specified in VisualAge Generator for shared data item definitions, regardless of whether the item is ever used in an SQL row record. • Uses the even or odd length specified in VisualAge Generator for nonshared items in all record definitions, because the item might be used as a host variable in an SQL where clause or prepare statement. • Uses an odd length (or 18 if the item is the maximum length) for nonshared items in tables, function parameters, function return values, and function local storage because the information to determine an even number of digits was not recorded in VisualAge Generator in these situations.	The migration tool does the same things as mentioned in the *Migrating with the associated part* column.
Potential Problem: A problem only arises if you eliminate the use of VisualAge Generator Compatibility mode. In this situation, overflow might occur due to having fewer significant digits than in VisualAge Generator Compatibility mode. **Potential Solution:** Before eliminating VisualAge Generator Compatibility mode, do the following: • Review your SQL table and view definitions to determine if you have any SQL columns that require even precision. If so, assess the SQL performance impact of using host variable lengths that do not match the SQL column definition. • Review all decimal data item definitions and primitive data item definitions in EGL records for even length items. Assess whether overflow might occur for any of these items.	**Potential Problem:** The same potential problem and solution as listed in the *Migrating with the associated part* column apply.

Shared edits and messages

VisualAge Generator: A shared data item definition can specify default edits and messages for both maps and User Interface (UI) records.

EGL: There is only one set of edit and message properties for a data item. Even though UI records are not supported in this release of EGL, the migration tool merges the map and UI properties for the data items. This preserves as much of your data item information as possible.

Associated part needed for migration: Not applicable.

Table 10. Shared edits and messages

Migrating with the associated part	Migrating without the associated part
The migration tool merges map and UI edits as follows: • For *each* edit or message, the migration tool does the first of the following that applies: – If a UI edit is specified, the migration tool converts the UI edit and its associated message information to the corresponding EGL properties. – If only a map edit is specified, the migration tool converts the map edit and its associated message to the corresponding EGL properties. – If the UI message is specified without its associated UI edit, the migration tool converts the UI message to the corresponding EGL property. – If the map message is specified without its associated map edit, the migration tool converts the map message to the corresponding EGL property. – If UI and map edit and message information are not specified, the migration tool does not set the corresponding EGL properties. The normal EGL defaults apply. • In VisualAge Generator, Justify and Hex edit are only specified for map edits, so they are always used to set the corresponding EGL properties.	Except as noted later in this section (Handling ambiguous situations for data items), the migration tool migrates the default edits and messages in the same way both with and without the associated parts.
Potential Problem 1: A problem only arises if conflicting map edits and UI edits exist in VisualAge Generator and you really intend the edits to differ between maps and UI records. The problem does not occur until the item is added to a text or print form. **Note:** If you never used VAGen Web Transactions, only map edits should exist in VisualAge Generator and you should not have a problem. **Possible Solution:** Other than adding a comment to the data item definition to list the VAGen map item edits and messages, there is nothing you can do for the data item definition. If you add the item to a text or print form, you can override any properties that need to differ for that particular form. **Potential Problem 2:** A problem can also arise if you use the item in an EGL UI record. The item might have some additional edits or messages that were migrated from the VAGen map edits. **Solution:** Always review the edits for items defined with a type definition in a UI record.	The same potential problems mentioned in the *Migrating with the associated part* column apply. You can use the same solutions.

Map item edit routine for shared data items

VisualAge Generator: A shared data item definition can have a map item edit routine that is a table, a function, or EZEC10 or EZEC11. The edit message is only used if the edit routine is EZEC10, EZEC11, or a table.

EGL: A data item can have both a validatorTable and a validator function. The data item can also have both a validatorTableMsgKey and a validatorMsgKey.

Associated part needed for migration: Either the table or the function part.

Table 11. Map item edit routine for shared data items

Migrating with the associated part	Migrating without the associated part
The first time the shared data item is migrated, the migration tool does the first of the following that applies: • If the editRoutineName is EZEC10 or EZEC11, the migration tool sets the validator property to the EGL equivalent system library function. The migration tool also sets the validatorMsgKey to the edit message, if any. • If the editRoutineName is a function, then the migration tool sets the validator property. The migration tool omits the validatorMsgKey because it is not used in VisualAge Generator. The migration tool also adds the function as an associate of the data item part so that an import statement is created in Stage 3. • If the editRoutineName is a table, then the migration tool sets the validatorTable property. The migration tool also sets the validatorTableMsgKey to the edit message, if any. The migration tool also adds the table as an associate of the data item part so that an import statement is created in Stage 3. • If the edit routine *is not* specified but the edit message *is* specified, the migration tool sets the validatorTableMsgKey to the edit message. **Note:** Even when migrating in program context, the editRoutineName might not be available if the shared item is not used on a map or if its edits on the map differ from what was specified in the shared item definition.	If a function or table with the same name as the editRoutineName is not available, the migration tool does the first of the following that applies: • If the editRoutineName is EZEC10 or EZEC11, the migration tool sets the validator property to the EGL equivalent system library function name. The migration tool also sets the validatorMsgKey to the edit message, if any. • If the editRoutineName is longer than 7 characters, it must be a function name, so the migration tool sets the validator property. The migration tool omits the validatorMsgKey because it is not used in VisualAge Generator. • If an edit message is specified, the migration tool sets the validatorTable and validatorTableMsgKey. • Otherwise, the migration tool sets the validator property and issues an error message. If the edit routine *is not* specified but the edit message *is* specified, the migration tool sets the validatorTableMsgKey to the edit message.
Potential Problem: A problem only arises if a VAGen function and table have the same name, most likely in different subsystems. In this situation, one subsystem uses a function and another subsystem uses a dataTable. The problem does not occur until the item is added to a text form. **Possible Solution:** Rename the dataTable so there are no longer two parts with the same name. Specify both a validator and validatorTable property for the item definition. If you add the item to a text form, delete the validator or validatorTable property, whichever is not needed for that particular form. Disadvantage: You must modify your programs and forms to use the new dataTable name.	**Potential Problem 1**: A problem arises if the migration tool guesses incorrectly. The problem does not occur until the item is added to a text form. **Possible Solution**: Correct the data item definition. **Potential Problem 2**: The same problem listed under the *Migrating with the associated part* column can also occur. You can use the same solution.

Fill characters for shared data items

VisualAge Generator: The default fill character for map edits is null for character, mixed or DBCS; blank for numerics; and 0 for hex. The default fill character for UI edits is blank for character, mixed, DBCS, unicode, and numerics; and 0 for hex. Null is not a valid fill character for UI records. A different fill character can be specified for map edits and UI edits.

EGL: There is only one default fillCharacter property for a data item. Even though UI records are not supported in this release of EGL, the migration tool merges the map and UI fillCharacter properties for the data items. This preserves as much of your data item information as possible.

Associated part needed for migration: Not applicable.

Table 12. Fill characters for shared data items

Migrating with the associated part	Migrating without the associated part
The first time the shared data item is migrated, the migration tool does the first of the following that applies: • If the UI edits specify a fill character, the migration tool migrates the character to the EGL fillCharacter property. • If the map edits specify a fill character, the migration tool migrates the character to the EGL fillCharacter property. • If neither the UI edits nor the map edits specify a fill character, the migration tool does not set the EGL fillCharacter property.	The migration tool does the same thing as mentioned in the *Migrating with the associated part* column.
Potential Problem: A problem only arises if you add a data item that was migrated using one type of edits to a different type of user interface (form or user interface record). **Possible Solution:** Always review the fill character when adding a new item to a form or user interface record.	The same problem listed under the *Migrating with the associated part* column can occur. You can use the same solution.

Handling ambiguous situations for records

Redefined records

VisualAge Generator: The redefined record type provides a different data item layout for another record. The redefined record specifies the name of the record it is redefining. For example, RecordA is a REDEFINED record that redefines RecordB. VisualAge Generator determines whether RecordA is really a redefinition of RecordB based on the use of RecordA within the program. If RecordA is used as a called parameter, RecordA is not treated as a redefinition of RecordB.

EGL: RecordA is a basicRecord. Redefinition information is only provided within a program definition -- not in the definition of RecordA.

Associated part needed for migration:

- When migrating a redefined record: not applicable.
- When migrating a program: the redefined record (RecordA).

Table 13. Redefined records

Migrating with the associated part	Migrating without the associated part
When migrating a redefined record, the migration tool does the following: • Migrates the redefined record (RecordA) to a basicRecord. • Includes a VAGen Info comment in RecordA indicating it redefined RecordB. • Issues an informational message that RecordA redefines RecordB.	When migrating a redefined record, the migration tool does the same thing mentioned in the *Migrating with the associated part* column.
When migrating a program, if RecordA is available, the migration tool does the following: • If RecordA is treated as a redefinition of RecordB in VisualAge Generator, the migration tool includes the redefines property in the declaration for RecordA. • If RecordA is not treated as a redefinition of RecordB in VisualAge Generator, the migration tool does not include the redefines property in the declaration for RecordA.	When migrating a program, if RecordA is not available, the migration tool does not know that RecordA is a redefined record. The migration tool does *not* include the redefines property in the declaration for RecordA.
Considerations for new use: A problem only arises if you need to use RecordA and RecordB in a new program. You must remember to include the redefines property for RecordA whenever you want RecordA to be treated as a redefinition of RecordB.	**Potential Problem 1:** A problem arises if the VAGen program uses RecordA as a redefinition. Immediately after migration, the program will not be a valid EGL program because the definition for RecordA and the import statement will be missing. There will be an error in the Problems view. If you migrate RecordA and add the import statement to the program, this will convert the program into a valid EGL program. However, there will be two data areas -- one for RecordA and one for RecordB. EGL will not detect this change during validation or generation. **The program will not run the same as in VisualAge Generator**. **Solution:** If you migrate additional records or add import statements to a program, review the record definitions for a VAGen Info comment. If there is a VAGen Info comment specifying that that RecordA is a redefinition for RecordB, update the program to include the *redefines* property for the declaration of RecordA. **Considerations:** The same considerations for new use listed under the *Migrating with the associated part* column can also occur.

Level 77 items in records

VisualAge Generator: Working storage records can have level 77 items.

EGL: Records cannot have level 77 items.

Associated part needed for migration:

- When migrating a working storage record: not applicable.
- When migrating a program: the primary working storage record.
- When migrating a function: the working storage record.

Table 14. Level 77 items in records

Migrating with the associated part	Migrating without the associated part
When migrating any working storage record that contains level 77 items, the migration tool does the following: • Splits the working storage record that contains level 77 items into two basicRecords -- one for the working storage structure and one for the level 77 items. The new level 77 record name is the original working storage record name with a customer-supplied suffix. • Places the new level 77 record in the same file with the original working storage record. • Issues an informational message that the level 77 record is being created.	When migrating any working storage record that contains level 77 items, the migration tool does the same things mentioned in the *Migrating with the associated part* column.
When migrating a program, if the primary working storage record is available and contains level 77 items, the migration tool adds a record declaration to the program for the new level 77 record if the primary working storage record contained level 77 items.	When migrating a program, if the primary working storage record is not available, the migration tool does not know whether the primary working storage record contains level 77 items. The migration tool does *not* include a record declaration for a level 77 record.
When migrating a function, if the working storage record is available, the migration tool changes qualified references to the level 77 items within the function to use the new level 77 record name.	When migrating a function, if the working storage record is not available, the migration tool does *not* change the qualification of item names.
Potential problem: A problem only arises for the level 77 item if there are two records of the same name, possibly in different subsystems, and the item is a level 77 item in one record and not in another. **Possible solution:** Move the item to a (new) common record and change the item qualification in all functions. Alternatively, do not qualify the item in the functions. **Considerations for new use:** There is a potential problem if you specify the original working storage record as the inputRecord property for a new program. Be sure to consider whether you also need to add a declaration for the new level 77 record.	**Potential Problem 1:** A problem arises if the primary working storage record contained level 77s and the program used the level 77s. Generation for the program will fail due to missing item definitions. **Solution:** Add the level 77 record to the program. **Potential Problem 2:** A problem arises for a function if the qualified data item is really a VAGen level 77 item. **Solution:** Modify the function to provide the correct qualifier for the data item. **Potential Problem 3:** The same problem listed under the *Migrating with the associated part* column can also occur. You can use the same solution. **Considerations:** The same considerations for new use listed under the *Migrating with the associated part* also apply.

Alternate specification records

VisualAge Generator: A record can specify another record as the alternate specification (altspec) record. For example, if RecordA specifies an altspec of RecordB, then RecordB provides the structure for RecordA. For SQL records, RecordB also provides the list of tables and keys for RecordA. If RecordA specifies a key item, that item is merged with the keys from RecordB when determining the default selection condition.

EGL: A record can embed another record to obtain the record structure. Only the record structure is included. Each SQL record must explicitly state its entire set of tables and keys.

Associated part needed for migration: If RecordA is an SQL record, you need the record specified as the altspec record (RecordB).

Table 15. Alternate specification records

Migrating with the associated part	Migrating without the associated part
If RecordA is an SQL record that specifies an alternate specification of RecordB and RecordB is available, the migration tool does the following for RecordA: • Creates the list of table names from the list of tables specified in RecordB. • Creates the list of keys by merging the following: – The items, if any, in RecordB that specified key=yes. – The item, if any, in RecordA that is specified as the key item. The order of the keys is the order in which the items appear in the structure of RecordB. • Includes the embed statement pointing to RecordB. • Migrates any !itemColumnName variables in the default selection conditions of RecordA to the corresponding SQL column names from RecordB.	If RecordA is an SQL record that specifies an alternate specification of RecordB, and RecordB is not available, the migration tool does the following for RecordA: • Sets the tableNames property to ###TABLES_NOT_FOUND### • Sets the keyItems property to ###KEYS_NOT_FOUND###, followed by the item, if any, that is specified as the key item in RecordA. • Includes the embed statement pointing to RecordB. • Migrates any !itemColumnName variables in the default selection conditions of RecordA to !itemColumnName without any substitution. • Issues error messages that the tables and keys could not be determined. • Issues an error message if there are any !itemColumnName variables.
Potential Problem: A problem only arises for SQL if the definition for RecordB differs, generally in different subsystems. There is no problem for non-SQL records. **Solution:** Duplicate the definition of RecordA so that each subsystem has its own definition of RecordA. Alternatively, specify *containerContextDependent=yes* for RecordA.	**Problem:** EGL validation for RecordA results in messages in the Problems view. **Solution:** Edit RecordA and include the appropriate table and key information based on RecordB. Also replace any !itemColumnName variables in the default selection condition with the corresponding SQL column name from RecordB.

Different definitions with the same record name

VisualAge Generator: Records include shared data items based on the projects and packages currently in the workspace. This enables different subsystems or programs to have different definitions of the same record name.

EGL: Provides the containerContextDependent=yes property for record definitions. This property enables you to specify that the data items used for type definitions are based on the program's part space.

Associated part needed for migration: Not applicable. You should have complete understanding of your VAGen part structure for all subsystems to be able to set this record property.

Table 16. Different definitions with the same record name

Migrating with the associated part	Migrating without the associated part
The migration tool does not set the *containerContextDependent=yes* property for record definitions. If you need this capability, you must set the property for each record that requires it.	The migration tool does the same thing as mentioned in the *Migrating with the associated part* column.

Handling ambiguous situations for tables

Reserved words and table names

VisualAge Generator: VisualAge Generator does not have reserved words. The # symbol is not valid in VAGen table names.

EGL: EGL has reserved words. In addition, EGL does not permit the # symbol as the first character of a part name. A dataTable name cannot be a reserved word.

Associated part needed for migration: Not applicable.

Table 17. Reserved words and table names

Migrating with the associated part	Migrating without the associated part
The migration tool does not rename the table for you. The migration tool used in Stage 1 of migration issues an error message if the table name matches the reserved word list. If you do not change the table name, the migration tool used in Stage 2 of migration also issues an error message.	The migration tool does the same thing as mentioned in the *Migrating with the associated part* column.
Potential Problem: A problem only arises if a dataTable name matches the reserved word list. EGL validation results in a message in the Problems view. **Solution:** Rename the table in VisualAge Generator before you migrate, or rename the dataTable in EGL after you migrate. If you change the name in VisualAge Generator, be sure to change all references to the table in programs, maps, functions, and data item definitions. If you change the name in EGL, you must change the name of the table and all references to it. This includes references in the following places: • Program use declaration statements • Logic statements in programs and functions • Data item validatorTable properties • Form field validatorTable properties If you want to keep the original table name as the name for the generated table, set the *alias* property to the original table name. If you do not specify the *alias* property, be sure to change any non-EGL references to the table name, including CICS program definitions.	The same problem listed under the *Migrating with the associated part* column can occur. You can use the same solution.

Handling ambiguous situations for map groups and maps

Reserved words and formGroup names

- **VisualAge Generator:** VisualAge Generator does not have reserved words. The # symbol is not valid in VAGen map group names.
- **EGL:** EGL has reserved words. In addition, EGL does not permit the # symbol as the first character of a part name. A formGroup name cannot be a reserved word.
- **Associated part needed for migration:** Not applicable.

Table 18. Reserved words and formGroup names

Migrating with the associated part	Migrating without the associated part
The migration tool does not rename the formGroup for you. The migration tool used in Stage 1 of migration issues an error message if the map group name matches the reserved word list. If you do not change the map group name, the migration tool used in Stage 2 of migration also issues an error message.	The migration tool does the same thing as mentioned in the *Migrating with the associated part* column.
Potential Problem: A problem only arises if the formGroup name matches the reserved word list. EGL validation results in a message in the Problems view. **Solution:** Rename the map group in VisualAge Generator before you migrate or the formGroup in EGL after you migrate. If you rename the map group in VisualAge Generator, be sure to rename all the maps that belong to the map group. Also change all references to the map group in all program definitions. If you rename the formGroup in EGL, you must change the name of the formGroup and all references to it, including references in program use declaration statements. If you want to keep the original map group name as the name for the generated formGroup, set the *alias* property to the original map group (formGroup) name. If you do not specify the *alias* property, be sure to change any non-EGL references to the formGroup name, including CICS program definitions.	The same problem listed under the *Migrating with the associated part* column can occur. You can use the same solution.

Map group and formGroup requirements

VisualAge Generator: A map group is only required if there is a floating area specification.

EGL: A formGroup is always required to contain the forms.

Associated part needed for migration: The map group and all maps in the map group.

Table 19. Map group and formGroup requirements

Migrating with the associated part	Migrating without the associated part
If a map group does not exist, the migration tool does the following: • Creates a formGroup for all maps with the same map group name. • Puts all the forms for the same formGroup in the same EGL file. • Nests the forms within the formGroup definition if not migrating with single file migration. • Issues an error message indicating that the formGroup requires editing to nest the forms if migrating in single file mode.	The migration tool does the same things mentioned in the *Migrating with the associated part* column. However, if you do not have the map group and all its maps in the same migration set, there can be problems as described below.

Table 19. Map group and formGroup requirements (continued)

Migrating with the associated part	Migrating without the associated part
Potential Problem: None. All maps for the map group should be included in the same migration set. Because the migration set represents what is generated, the migration set should include all maps in the map group. If you are migrating in single file mode, be sure to include all the maps in the map group in the same External Source Format file.	**Potential Problem:** If all maps for the same map group name are not included in the same migration set (or External Source Format file for single file mode migration), the formGroup will not include all the forms. **Possible Solution 1:** Be sure the migration set includes all maps with the same map group name. **Possible Solution 2:** Add the missing forms to the EGL file and nest them within the formGroup definition.

Floating areas and starting positions

VisualAge Generator: VisualAge Generator permits, but does not recommend, having different floating area sizes and starting positions for different device types that have the same device size.

EGL: EGL formGroups and print forms only specify the device size. EGL text forms specify both the device size and the form size. EGL only permits one set of margin specifications for a device size.

Associated part needed for migration: Not applicable.

Table 20. Floating areas and starting positions

Migrating with the associated part	Migrating without the associated part
The migration tool does the following: • Issues an error message if two or more devices have the same device size but different floating area sizes or starting positions. • Includes the EGL equivalent device size and margin specifications for each VAGen floating area specification. • Includes a warning message in the EGL source.	The migration tool does the same things mentioned in the *Migrating with the associated part* column.
Potential Problem: A problem only arises if two or more devices with the same device size specify different floating area sizes or starting positions in VisualAge Generator. EGL validation results in a message in the Problems view. **Possible Solution:** Review the error messages. Edit the formGroup definition to specify the one floating area size and starting position that you require for this device size. Then delete the warning message from the EGL source.	The same problem listed under the *Migrating with the associated part* column can occur. You can use the same solution.

Map groups, maps, and device sizes

VisualAge Generator: VisualAge Generator supports device sizes for display maps with a depth and width of 6x40, 12x40, 16x64, and 255x160.

EGL: EGL supports common device sizes for text forms, but does not permit device sizes of 6x40, 12x40, 16x64, and 255x160 for COBOL generation. These devices are supported for Java generation..

Associated part needed for migration: Not applicable.

Table 21. Map groups, maps, and device sizes

Migrating with the associated part	Migrating without the associated part
When migrating a map group, the migration tool includes the original depth and width in the *screenSize* property within the *ScreenFloatingArea* property. The tool also issues a warning message.	The migration tools does the same thing as mentioned in the *Migrating with the associated part* column.
When migrating a display map, the migration tool includes the original depth and width in the *screenSizes* property for the migrated text form. The tool also issues a warning message.	The migration tools does the same thing as mentioned in the *Migrating with the associated part* column.
Potential Problem: These devices are not supported by EGL COBOL generation. There will be an error during generation. **Solution:** If you generate for COBOL, edit the *formGroup* and the text form in EGL and either remove or change the obsolete screen size.	The same problem listed under the *Migrating with the associated part* column can occur. You can use the same solution.

Map names and help map names

VisualAge Generator: Map names are two-part names consisting of the map group and the map name. The main map group and the help map group for a program can both contain a map with the same name. For example, for Program X, main map group GROUPA and help map group GROUPH can each contain a map named MAP1. A map name is limited to 8 characters. A map name can be the same as the program name. The # symbol is not valid in VAGen map group names, but the # symbol is permitted in the map name portion of a map name.

EGL: Form names do not include the formGroup name. Instead, text and print forms are defined within a formGroup part. Within a program, all the text form and help text form names must be unique. EGL also requires that all form names in the main formGroup and help formGroup be unique (no duplicate form names in the two formGroups for a program). EGL does not permit a form name to be the same as the name of a program. In addition, EGL does not permit the form name to be a reserved word or to use the # symbol as the first character of the form name. EGL allows form names to be longer than 8 characters at definition time. At generation time, if an alias is specified, the alias is used as the form name. Duplicate names are permitted in the main formGroup and help formGroup for the generated code.

Associated parts needed for migration: When migrating a map group, you need the program and its map group, help map group, and all the maps in both map groups.

Table 22. Map names and help map names

Migrating with the associated parts	Migrating without the associated parts
Based on the first program to migrate either the main map group or the help map group, the migration tool does the following: • Performs any renaming for map names due to reserved words or the # symbol being used as the first character of the map name portion of the name. Maps in both the program's main map group and help map group are renamed as necessary. • Checks the names of *all* maps in the program's help map group for duplicate names with the main map group. • Compares the program name to the names of all maps in the program's main map group and help map group. If a map in the help map group does not have the same name as any map in the main map group, the migration tool does not change the help map name. If a map in the help map group has the same name as any map in the program's main map group, the migration tool does the following: • If the help map contains only constants, the migration tool does the following: – Renames the help map to helpMapName plus a customer-specified suffix. – Includes the alias property with the original help map name. – Changes the helpForm property for any map to specify the new help map name. • If a map in the help map group contains variables, the migration tool does the following: – Issues an error message. – Does not rename the map. – Migrates the map. This is because the map could be used by some other program that specifies the help map group as that program's main map group. If a map in the help map group only contains constant fields and the map name is the same as the program name, the migration tool renames the help map. The same processing is done as occurs when renaming can be done for conflicting map names in the help map group and main map group. If a map in the help map group contains any variable fields and is named the same as the program name or if a map in the main map group is named the same as the program, the migration tool does not rename the map. The same processing is done as occurs when renaming cannot be done for conflicting map names in the help map group and main map group.	When migrating map groups, if a program is not available, the migration tool does not know that two map groups are related and does not know whether a map group is ever specified as a help map group. The migration tool does the following: • Performs any renaming for map names due to reserved words. • Does *not* check for additional renaming of help maps.

Table 22. Map names and help map names (continued)

Migrating with the associated parts	Migrating without the associated parts
Potential Problem 1: A problem can arise if a formGroup is used as a main formGroup in one program and a help formGroup in another program. **Possible Solution:** Separate the help formGroup into two formGroups, one containing only help forms and the other containing forms with variable fields. Specify the formGroup that contains only help forms as the help formGroup for the original program. Specify the formGroup containing the forms with variable fields as the main formGroup and the formGroup containing only the help forms as the help formGroup for the second program. **Potential Problem 2:** A problem arises if a map in the help formGroup contains variable fields and has the same name as a map in the main formGroup. **Possible Solution:** Same as possible solution for Problem 1. **Potential Problem 3:** A problem can arise if the same help formGroup is shared by multiple programs. In this case, the migration tool might not rename all the help forms that need to be renamed for the various programs. **Possible Solution:** Rename all the necessary forms in the help formGroup by adding your help map suffix to the name. Also change all corresponding text forms in all form groups to specify the new help form name.	**Potential Problem:** A problem only arises if the formGroup is used in a program and there is a conflict between the form names in the main formGroup and help formGroup. **Possible Solutions:** The same solutions as shown for *Migrating with the associated part* apply.

Numeric variable fields

VisualAge Generator: A numeric field on a map has one length. The length should be long enough to allow for all the digits, the decimal point, sign, currency symbol, and numeric separator. However, if the field is not long enough at runtime, VisualAge Generator omits the currency symbol and numeric separator. VisualAge Generator also omits the sign if it is positive. If necessary to fit into the space allowed, VisualAge Generator drops the high order digits.

EGL: Variable fields on a form specify both a type definition, which includes the number of digits and decimals, and a fieldLen property that specifies the space that the data occupies on the form. If the fieldLen is not big enough to contain all the digits and formatting characters at runtime, EGL issues a runtime message.

Associated part needed for migration: Not applicable.

Table 23. Numeric variable fields

Migrating with the associated part	Migrating without the associated part
When migrating a numeric field on the map, the migration tool sets the length and fieldLen as follows: • The migration tool always sets the fieldLen to the same length as specified for the variable field in VisualAge Generator. • The migration tool sets the length and decimals in the type definition as follows: – If the variable field does not specify decimals, the migration tool sets the length in the type definition to the fieldLen. – If the variable field specifies decimals, the migration tool sets the length in the type definition to fieldLen minus 1 to allow for entry of the decimal point. This technique avoids any overflow problems that might occur at run time.	The migration tool does the same thing as mentioned in the *Migrating with the associated part* column.
Potential Problem: If the field length on the form is not large enough at run time to contain all the digits, decimal point, sign, currency symbol, and numeric separator characters, EGL issues a run time error message. **Solution:** Change the form definition so that the fieldLen is large enough to contain the largest possible number that will occur at run time and all the formatting characters that you specify.	The same problem listed under the *Migrating with the associated part column* can occur. You can use the same solution.

Variable map fields and edit routines

VisualAge Generator: A map variable field can have an edit routine that is a table, a function, EZEC10, or EZEC11. The edit message is only used if the edit routine is EZEC10, EZEC11, or a table.

EGL: A form field can have both a validatorTable and a validator function. A form field can also have both a validatorTableMsgKey and a validatorMsgKey.

Associated part needed for migration: Either the table or function part.

Table 24. Variable map fields and edit routines

Migrating with the associated part	Migrating without the associated part
The first time the map is migrated, the migration tool does the first of the following that applies: • If the editRoutineName is EZEC10 or EZEC11, the migration tool sets the validator property to the EGL-equivalent system library function. The migration tool also sets the validatorMsgKey to the edit message, if any. • If the editRoutineName is a function, then the migration tool sets the validator property. The migration tool omits the validatorMsgKey because it is not used in VisualAge Generator. • If the editRoutineName is a table, then the migration tool sets the validatorTable property. The migration tool also sets the validatorTableMsgKey to the edit message, if any.	If a function or table with the same name as the editRoutineName is not available, the migration tool does the first of the following that applies: • If the editRoutineName is EZEC10 or EZEC11, the migration tool sets the validator property to the EGL equivalent system library function name. The migration tool also sets the validatorMsgKey to the edit message, if any. • If the editRoutineName is longer than 7 characters it must be a function name, so the migration tool sets the validator property. The migration tool omits the validatorMsgKey because it is not used in VisualAge Generator. • If an edit message is specified, the migration tool sets the validatorTable and validatorTableMsgKey. • If an edit message is not specified, the migration tool sets the validator property and issues an error message.
Potential Problem: A problem only arises if a VAGen function and dataTable have the same name (most likely in different subsystems) and two programs share the same formGroup (most likely in the same subsystem) and one program expects to use the function and the other program expects to use the dataTable. **Possible Solution:** Review programs that share a formGroup. If the situation arises, create a separate formGroup to use the validatorTable. Disadvantage: There are now two formGroups to maintain. You can minimize this disadvantage by moving identical forms to a common file and then specifying the use formName statement in each formGroup to point to the common forms.	**Potential Problem:** A problem only arises if the migration tool guesses incorrectly. Any program that uses this map might expect a table when the migration tool specified a function. **Possible Solution:** Review the uses of maps that have error messages.

Map fields and the numeric hardware attribute

VisualAge Generator: VisualAge Generator supports the numeric hardware attribute for character constant fields, character variable fields, and numeric variable fields. The numeric hardware attribute prevents the end user from typing non-numeric data in a variable field.

EGL: EGL only supports the *isDecimalDigit* attribute for character variable fields. Numeric fields have a soft edit to ensure that only valid numeric characters and formatting characters such as a sign or decimal point are entered into the field.

Associated part needed for migration: Not applicable.

Table 25. Map fields and the numeric hardware attribute

Migrating with the associated part	Migrating without the associated part
The migration tool does the following: • For any or character variable on a map that specified the numeric hardware attribute, the tool includes *isDecimalDigit = yes* property. • For any character constant on the map, the tool always omits the *isDecimalDigit* property. • For any numeric variable field on the map, the tool always omits the *isDecimalDigit* property.	The migration tool does the same thing as mentioned in the *Migrating with the associated part* column.
Potential Problem: The end users will notice a slight change at run time because they will be able to type non-numeric data into numeric fields. EGL will issue a runtime error message if this occurs. **Possible Solution:** Consider notifying your end users that this is an expected difference when changing from VAGen-generated code to EGL-generated code.	The same problem listed under the *Migrating with the associated part* column can occur. You can use the same solution.

Map arrays and attributes

VisualAge Generator: VisualAge Generator permits, but does not recommend, using different attributes for the elements of an array. For example, in VisualAge Generator the protection, input required, require fill on input, numeric hardware attribute, modified data tag, and light pen detect can vary for each element of the map array.

EGL: In EGL, the only properties that can be overridden for an array item are the field presentation properties (color, highlight, intensity, protect, modified, and outline) plus cursor, position, and value.

Associated part needed for migration: Not applicable.

Table 26. Map arrays and attribute fields

Migrating with the associated part	Migrating without the associated part
The migration tool uses the following properties for the first element of the array (array index 1) to set the EGL equivalent properties: input required, require fill on input, numeric hardware attribute, and light pen detect. EGL uses the properties for the first element of the array for **all** the elements of the array.	The migration tool does the same thing as mentioned in the *Migrating with the associated part* column.
Potential Problem: A problem only arises if you used different attributes for the elements of the array. **Possible Solution:** Change the properties for the first element of the array to the least restrictive values and add logic in a validator function to verify that each element of the array meets the necessary criteria. Also notify your end users of any differences in the appearance of the form at runtime.	The same problem listed under the *Migrating with the associated part* column can occur. You can use the same solution.

Unnamed variable fields

VisualAge Generator: VisualAge Generator permits, but does not recommend, unnamed variable fields on a map. At generation time, unnamed variable fields are converted into constants. Programs and functions can never reference the unnamed variable field.

EGL: EGL does not permit unnamed variable fields on a form.

Associated part needed for migration: Not applicable.

Table 27. Unnamed variable fields

Migrating with the associated part	Migrating without the associated part
For any unnamed variable fields on the map, the migration tool checks to see if any of the following are specified: • Initial value • Protect = yes • Cursor = yes • Outlining other than ″No outlining″ • Highlighting other than ″No highlighting″ If any of the above are specified, the migration tool creates a constant field with the corresponding EGL properties and issues a warning message.	The migration tool does the same thing as mentioned in the *Migrating with the associated part* column.
If none of the properties are specified for the unnamed variable field, the migration tool does the following: • Does not create a constant field on the form. • Issues a warning message.	The migration tool does the same thing as mentioned in the *Migrating with the associated part* column.
Potential Problems: None. You could not reference the field in VisualAge Generator.	**Potential Problems:** None.

Unprotected map constants

VisualAge Generator: VisualAge Generator supports the use of unprotected constants on a map. At test and generation time, unprotected constants are treated as though the protection is set to autoskip.

EGL: EGL does not support the use of unprotected constants on a form. For constants on text forms, EGL supports both protect=yes and protect=skip. For print forms, EGL does not support the protect property.

Associated part needed for migration: Not applicable.

Table 28. Unprotected map constants

Migrating with the associated part	Migrating without the associated part
When migrating a form, for an unprotected constant field, the migration tool does the following: • If the form is a text form, the migration tool sets the EGL protect property to skip and issues an error message. • If the form is a print form, the migration tool omits the protect property and does not issue a message. The protect property is not used in EGL print forms.	The migration tool does the same thing as mentioned in the *Migrating with the associated part* column.
Potential Problem: None.	**Potential Problem:** None.

Fields at row=0, column=0

VisualAge Generator: VisualAge Generator 4.5 tolerates fields positioned at row=0, column=0 from older releases of Cross System Product or VisualAge Generator. However, VisualAge Generator 4.5 did not provide a way to create fields at this position. You could not set attribute information for fields positioned at row=0, column=0.

EGL: EGL does not support fields positioned at row=0, column=0. Every field must include an attribute byte.

Associated part needed for migration: Not applicable.

Table 29. Fields at row=0, column=0

Migrating with the associated part	Migrating without the associated part
When migrating a form, if a field is positioned at row=0, column=0, the migration tool does the following: • If the field is a constant field and the first character of the value is blank, the migration tool does the following: – Removes the first character from the value and reduces the field length by 1. – Sets the position property to (1,1). – Includes default presentation properties for the field. – Issues a warning message. • If the field is a constant field and the first character of the value is not blank OR if the field is a variable field, the migration tool does the following: – Does not change the value or the field length. – Sets the position property to (0,0). – Includes default presentation properties for the fields. – Issues an error message.	The migration tool does the same thing as mentioned in the *Migrating with the associated part* column.

Table 29. Fields at row=0, column=0 (continued)

Migrating with the associated part	Migrating without the associated part
Potential Problem 1: If the field cannot be changed and is at position=(0,0), there will be an error in the Problems view. **Solution 1:** Modify the form and change the position of the field. You might need to move other fields or reposition constants to make room for the attribute byte for the field. Also review the default presentation properties to ensure that the correct color, highlighting, and so on are used. **Potential Problem 2:** If a constant field is changed to position=(1,1), there might be a different runtime appearance due to the default presentation properties. **Solution 2:** Review the migration warning messages and be sure to test any forms where the migration tool adjusted the position of a field.	**Potential Problem:** The same problems listed under the *Migrating with the associated part* column can occur. You can use the same solutions.

Handling ambiguous situations for programs

Program names and reserved words

VisualAge Generator: VisualAge Generator does not have reserved words. The # symbol is not valid in VAGen program names.

EGL: EGL has reserved words. In addition, EGL does not permit the # symbol as the first character of a part name. A program name cannot be a reserved word.

Associated part needed for migration: Not applicable.

Table 30. Program names and reserved words

Migrating with the associated part	Migrating without the associated part
The migration tool does not rename the program for you. The migration tool used in Stage 1 of migration issues an error message if the program name matches the reserved word list. If you do not change the program name, the migration tool used in Stage 2 of migration also issues an error message.	The migration tool does the same as mentioned in the *Migrating with the associated part* column.
Potential Problem: A problem only arises if a program name matches the reserved word list. EGL validation results in a message in the Problems view. **Solution:** Rename the program. You can do this either in VisualAge Generator or in EGL after you migrate. If you rename the program in EGL, you must change the name of the program and all references to it, including references on call, transfer, and show statements. If you want to keep the original program name as the name for the generated program, set the *alias* property to the original program name. If you do not specify the *alias* property, be sure to change any non-EGL references to the program name, including CICS program definitions.	The same problem listed under the *Migrating with the associated part* column can occur. You can use the same solution.

Implicit data items in programs

VisualAge Generator: VisualAge Generator permits, but does not recommend, the use of implicit data items (items that are not explicitly defined in a record, map, table, called parameter list, function parameter list, or function local storage).

EGL: EGL does not permit implicit data items.

Associated part needed for migration: Not applicable.

Table 31. Implicit data items in programs

Migrating with the associated part	Migrating without the associated part
The migration tool does not create definitions for implicit items for you. The migration tool used in Stage 2 of migration issues a warning message if the program allows implicit items.	The migration tool does the same thing as mentioned in the *Migrating with the associated part* column.
Potential Problem: A problem only arises if the program actually uses implicit items. Review the ″TO DO″ list log for any programs that allow implicit items. If the program actually used implicits there will be errors in the Problems view. **Solution:** You can add a definition for the implicit item to the program either in VisualAge Generator or in EGL. VAGen validation shows the definition that is needed for the implicit item.	The same problem listed under the *Migrating with the associated part* column can occur. You can use the same solution.

Associated program parts

VisualAge Generator: The program's associates can be in multiple projects and packages for VisualAge Java or in multiple configuration maps and applications for VisualAge Smalltalk.

EGL: The program's associates can be in multiple projects, folders, packages, and files.

Associated parts needed for migration: For a program: All associates.

Note: See "EGL build path and import statements" on page 29 for additional information on import statements.

Table 32. Associated program parts

Migrating with the associated part	Migrating without the associated part
The migration tool does the following: • Includes a package statement to specify the package in which the EGL file is to be placed. • Includes import statements in the EGL file for any packages that contain associates needed by any of the parts in the current file and which are in a different package from the current file. The import statements are only included if you migrate using Stages 1 through 3. • For the program the migration tool does the following: – Includes a declaration for the program's primary working storage record. If there are level 77s in the VAGen primary working storage record, the tool also includes a declaration for the new level 77 item record. – Includes declarations for all records in the VAGen Tables and Additional Records list, including the redefines property, if applicable, for any VAGen redefined records. – Includes declarations for all I/O records. – Includes declarations for records used as parameters on MQ API calls (the records specified as attributes of an MQ Message record in VisualAge Generator). – Includes use declarations for any tables in the VAGen Tables and Additional Records list. – Includes use declarations for the program's map group and help map group.	The migration tool makes use of all program associates that are available.

Table 32. Associated program parts (continued)

Migrating with the associated part	Migrating without the associated part
Potential problem: None.	**Potential Problem 1:** A problem only arises if there are missing parts. If the migration tool detects missing parts, it issues a warning message that identifies the missing parts. The migration tool does not make any assumption about the missing part(s). This can result in a variety of problems in the migrated program, including the following: • Missing import statements. • Missing level 77 record declaration. • Missing redefines property for VAGen redefined records. • Missing I/O record declarations. • Missing declarations for records used as parameters on MQ API calls. Except for the missing redefines property, there will be errors in the Problems view to help you identify the problem(s). ***Note: the migration tool does not detect all missing parts.*** **Possible Solution 1A:** Change your migration set to include all the parts that are needed to validate the program in VisualAge Generator. Migrate the program again using the new migration set so that all the program's associates are migrated together. **Possible Solution 1B:** Locate the missing parts in EGL and correct the EGL program. **Potential Problem 2:** For missing level 77 items, see "Level 77 items in records" on page 48. **Potential Problem 3:** For missing redefined records, see "Redefined records" on page 47.

Intermediate variables required for migration

VisualAge Generator: Some VAGen statements require intermediate variables to provide the equivalent support in EGL.

EGL: EGL provides system library functions that provide some information required for VAGen migration. This support is only available in VisualAge Generator Compatibility mode.

Associated part needed for migration: Not applicable.

Table 33. Intermediate variables required for migration

Migrating with the associated part	Migrating without the associated part
When migrating any program, the migration tool always does the following: • Includes declarations for the following: – *custPrefix*EZESYS – *custPrefix*EZEREPLY – *custPrefix*EZE_ITEMLEN – *custPrefix*EZE_WAIT_TIME • Includes an initialization statement to set the value of *custPrefix*EZESYS to the old VAGen EZESYS value. *custPrefix* is the same prefix that is used for changing part names that conflict with reserved words. Use the VAGen Migration Syntax Preferences to set its value.	The migration tool does the same things mentioned in the *Migrating with the associated part* column.
The 4 variables are used for migrating the following: • References to EZESYS in statements other than IF, WHILE, and TEST where the old VAGen value is required. • VAGen service routines if the (REPLY option is not specified. In this situation, the current value of handleSysLibErrors must be saved and restored. • The TEST nnn, +nnn, or -nnn statement which has no direct equivalent in EGL. An EGL system library function is used to determine the length of the data the user entered. • The EZEWAIT function. In this situation, the migration tool adds logic to convert the time to seconds.	The same problem listed under the *Migrating with the associated part* column can occur. You can use the same solution.
Potential Problems: None.	**Potential Problems:** None.

Handling ambiguous situations for functions, including I/O statements

DISPLAY statement for maps

VisualAge Generator: *DISPLAY* is used for both display maps and printer maps.

EGL: Two separate statements are used:

- *display form* is used for text forms.
- *print form* is used for print forms.

In VisualAge Generator Compatibility mode, *display form* is accepted if the form is a print form.

Associated part needed for migration: The map is needed to determine the device type. The first map with this map name in any available map group is the map that the migration tool uses. When migrating in program context, the migration tool only looks at the program's main map group.

Table 34. Display statement for maps

Migrating with the associated part	Migrating without the associated part
Based on the first migration of this function, if a map with this name is available, the migration tool converts to the following: • *display textForm* if the map is a display map • *print printForm* if the map is a printer map	If a map with this name is not available, the migration tool does the following: • Converts to *display form* • Issues a warning message that the map type could not be determined

Potential Problem 1: The first program that migrated used a print form so the migration tool migrated to the print statement. Another program uses the same function, but with a text form.
 	
Solution 1: Use VisualAge Generator Compatibility mode. Edit the function and change the print statement to a display statement.
 	
Potential Problem 2: A problem arises if you want to eliminate the use of VisualAge Generator Compatibility mode and two programs use the function -- one with a text form and one with a print form.
 	
Possible Solution 2A: If a specific target environment always uses display maps and other environments always use print maps, you could change the EGL function to something similar to the following:
`if (sysVar.systemType is zoscics)`
`  DISPLAY_FUNCTION();`
`else`
`  PRINT_FUNCTION();`
`end`
 	
where DISPLAY_FUNCTION and PRINT_FUNCTION use the display and print statements, respectively.
 	
Possible Solution 2B: Assuming the function migrated to a display statement, change the function from the following:
`before-logic`
`display textForm;`
`after-logic`
 	
to the following:
`before-logic-function();`
`display textForm;`
`after-logic-function();`
 	
Putting the before-logic and after-logic into separate functions enables you to keep most of the logic in common functions. Then you can make a copy of the modified display function and change it to use print map, but still use the common before-logic-function and after-logic-function. Disadvantage: This has the potential to ripple back into functions that use the original DISPLAY function.
	Potential Problem: The same potential problems and possible solutions as listed in the *Migrating with the associated part* column apply.

I/O error routine

VisualAge Generator: A function that does file or database I/O can specify an I/O error routine. The I/O error routine can be a main function or a non-main

function; the syntax is the same. VisualAge Generator determines at test or generation time whether the I/O error routine is a main function or non-main function for the program. When a main function is used as the I/O error routine, VisualAge Generator pops the function stack back to the top of the stack, starts the stack over again with only the (I/O error routine) main function on the stack, and then invokes the main function. When a non-main function is used as the I/O error routine, VisualAge Generator adds the non-main function to the current function stack and then invokes the function.

EGL: The *try* block and *onException* statement are used for error handling. The syntax for an onException statement supports the following:

- Transferring back to a main function using *exit stack functionName;*
- Invoking a non-main function using *nonmainfunctionName();*
- Invoking a main function with *mainfunctionName();* This form is not supported by VisualAge Generator. EGL adds the main function to the current function stack and then invokes the main function.

Associated part needed for migration: The program with its list of main functions.

Table 35. Ambiguous situations for functions—File and database I/O error routines

Migrating with the associated part	Migrating without the associated part

Based on the first migration of this function, if there is a program available, the migration tool does the following:
• Changes an I/O error routine that specifies a program main function to:
`try`
`  I/O-Statement;`
`  onException  exit stack functionName;`
`end`
• Changes an I/O error routine that specifies a non-main function to:
`try`
`  I/O-Statement;`
`  onException functionName();`
`end`
	If there is no program available, the migration tool does the following:
	• Assumes that the function named in an I/O error routine is a non-main function and changes it to the following:
	`try`
	`  I/O-Statement;`
	`  onException functionName();`
	`end`
	• Does not issue a warning message due to the high volume of messages that could be issued and the likelihood that messages will be ignored or hide other serious error messages.

Potential Problem: A problem arises if this function is used in a program where the I/O error routine differs in its use as a main or non-main function from the original program. **Note:** There will not be a message in the Problems view. Generation will not detect an error. *However, the program will not run the same as in VisualAge Generator.* Instead of popping the stack as in VisualAge Generator, EGL will add the main function to the stack. **Possible Solution 1:** If this situation arises, create a new version of this I/O function with the proper syntax for transferring to a main function. Note that this technique has the potential to ripple back into other functions that invoke the I/O function. **Possible Solution 2:** If this situation arises, restructure the program so it does not use the equivalent of VAGen flow statements.	**Potential Problem:** A problem arises if this function is used in a program where the I/O error routine is a main function. **Note:** There will not be a message in the Problems view. Generation will not detect an error. *However, the program will not run the same as in VisualAge Generator.* Instead of popping the stack as in VisualAge Generator, EGL will add the main function to the stack. **Possible Solutions:** The same solutions listed for *Migrating with the associated part* apply.

SQL I/O statements

VisualAge Generator: For SQL I/O, test and generation expand a single I/O option into multiple SQL statements as needed based on the record definition and the use of Execution Time Statement Build. Test and generation always create the tables clause for the I/O statement from the SQL record definition.

EGL: SQL statements must be explicitly specified in the EGL program. If an SQL statement is modified, *all* SQL clauses except the into clause are required. Execution Time Statement Build is replaced by the prepare statement followed by an open, get, or execute statement.

Associated part needed for migration: The SQL record and the record specified as the alternate specification record, if any.

Table 36. Ambiguous situations for functions—SQL I/O

Migrating with the associated part	Migrating without the associated part
Based on the first migration of this function, if the SQL record and its alternate specification record are available, the migration tool creates the corresponding EGL statement(s) based on the record definition, the SQL statement within the function, and the use of Execution Time Statement Build. If the SQL statement in the function was modified, the migration tool does the following: • Builds the EGL SQL statement with all clauses, including the *into* clause. • Creates any required tables clause from the table names in the SQL record or, if applicable, its alternate specification record. • Creates any other missing clauses that are required for this SQL I/O statement based on the record definition for the I/O object, or if applicable, the record definition for the I/O object's alternate specification record. • Converts any !itemColumnNames from the item name to the corresponding SQL column name. • Does not review the SQL statement for the SQL reserved words that require special treatment. See "SQL reserved words requiring special treatment" on page 174 for the list of reserved words and the changes you must make to your SQL statement if you use one of these reserved words as a table or column name. **Note:** • See "SQL I/O and missing required SQL clauses" on page 70 for details on problems related to missing SQL clauses. • See "SQL I/O and !itemColumnName" on page 72 for details on problems related to using !itemColumnNames.	If the SQL record or its alternate specification record are not available, the migration tool only has the SQL statement modifications and Execution Time Statement Build information to use in creating the EGL SQL statements. Because the migration tool does not have a record definition available, the migration tool does the following: • Builds the EGL SQL statement with all clauses, including the *into* clause. • Uses EZE_UNKNOWN_SQLTABLE as the table name and T1 as the table label in any tables clause. • Uses EZE_UNKNOWN_SQL_*clausename* for any missing SQL clauses, where *clausename* is the External Source Format key word for the missing SQL clause (for example, SELECT, WHERE, ORDERBY). • Uses !itemColumnNames for any column name variables. • Issues an error message that the function needs to be reviewed. • Does not review the SQL statement for the SQL reserved words that require special treatment. See "SQL reserved words requiring special treatment" on page 174 for the list of reserved words and the changes you must make to your SQL statement if you use one of these reserved words as a table or column name. **Note:** • See "SQL I/O and missing required SQL clauses" on page 70 for details on problems related to missing SQL clauses. • See "SQL I/O and !itemColumnName" on page 72 for details on problems related to using !itemColumnNames.

Table 36. Ambiguous situations for functions—SQL I/O (continued)

Migrating with the associated part	Migrating without the associated part
Potential Problem 1: A problem only arises if there are two records with the same name that have different SQL table names or table labels. This might occur in different subsystems or when generating using different tables for test and production. **Possible Solution 1A:** If the problem is due to changing the qualification for a table name between test and production, change to use unqualified table names and specify the qualification information at BIND time. **Possible Solution 1B:** If the problem is due to different table names in different subsystems, make a copy of the record and rename it. Then make a copy of the I/O function to use the new record name. Correct the new I/O function to have the proper tables clause. ***Disadvantage:*** This has the potential to ripple back into functions that use this I/O function. **Possible Solution 1C:** If the problem is due to different table names in different subsystems, change the record to use the *tableNameVariables* property and modify all functions that do I/O for this record to set the table name variable before invoking the I/O function -- possibly in each program's main function. Alternatively, make the change to table name host variables in VisualAge Generator and migrate the program, record and function again. ***Disadvantage:*** There are potential performance implications because this changes from static to dynamic SQL. **Potential Problem 2:** A problem arises if any SQL table name or column name is one of the SQL reserved words that requires special treatment. The migration tool does not enclose these SQL reserved words in double-quotes. There will be an error in the Problems view. **Solution 2A:** Edit the function and enclose the SQL table name or column name in double quotes. See "SQL reserved words requiring special treatment" on page 174 for the list of SQL reserved words and an example of the required syntax.	**Potential Problem 1:** A problem arises for any modified SQL statement or any SQL statement that uses Execution Time Statement Build. Depending on whether the record is missing and which specific SQL clauses are missing from the SQL statement, there might be errors in the Problems view. **Solution:** Review the migration log for any messages related to missing SQL clauses or table names. Alternatively, search the workspace for any occurrences of EZE_UNKNOWN. Determine the proper tables clause based on the record definition. See "SQL I/O and missing required SQL clauses" on page 70 for information about recreating the SQL clause in EGL. See "SQL I/O and !itemColumnName" on page 72 for information about correcting any !itemColumnName variables. **Other potential problems:** The same potential problems and solutions as shown for *Migrating with the associated part* apply.

SQL I/O and missing required SQL clauses

VisualAge Generator: VisualAge Generator 4.5 stored all the SQL clauses if you modified any SQL clause. However, some earlier versions of Cross System Product and VisualAge Generator only stored the clause that you modified. If a function from an earlier version was never modified in VisualAge Generator 4.5, then some of the required SQL clauses might be missing.

EGL: If any SQL clause is modified, all SQL clauses for the SQL statement must be specified.

Associated part needed for migration: The SQL record and the record specified as the alternate specification record, if any.

Table 37. SQL I/O and missing SQL clauses

Migrating with the associated part	Migrating without the associated part
If the SQL record and its alternate specification record are available, and if any SQL clause is present, but some clauses are missing, the migration tool creates the missing clauses as shown in the next rows of this table. Based on the first migration of this function, the migration tool uses the SQL record and its alternate specification record, if any, to create the missing clauses.	If the SQL record or its alternate specification record are not available, and if any SQL clause is present, but some clauses are missing, the migration tool creates the missing clauses as shown in the next rows of this table. Based on the first migration of this function, the migration tool creates intentionally invalid EGL syntax if the SQL record or its alternate specification record is not available.
Missing SELECT clause: The migration tool creates a select clause by listing all the SQL column names from the record in the same order that the items appear in the record.	**Missing SELECT clause:** The migration tool sets the SQL column names for the select clause to EZE_UNKNOWN_SQL_SELECT and issues an error message.
Missing INTO clause: The migration tool creates the into clause by listing all the item names from the record in the same order that the items appear in the record.	**Missing INTO clause:** The migration tool sets the item names for the into clause to EZE_UNKNOWN_SQL_INTO and issues an error message.
Missing INSERTCOLNAME clause: The migration tool creates the list of column names to be inserted for a VAGen ADD function by listing the SQL column names from the record in the same order that the items appear in the record. The migration tool omits the SQL column name for any item that is identified as read only.	**Missing INSERTCOLNAME clause:** The migration tool sets the SQL column names for the list to EZE_UNKNOWN_SQL_INSERTCOLNAME and issues a error message.
Missing VALUES clause: The migration tool creates the values clause for a VAGen ADD function by listing the item names from the record in the same order that the items appear in the record. The migration tool omits the item name for any item that is identified as read only.	**Missing VALUES clause:** The migration tool sets the item names for the values clause to EZE_UNKNOWN_SQL_VALUES and issues an error message.
Missing FORUPDATEOF clause: The migration tool creates the for update of clause by listing the SQL column names from the record in the same order that the items appear in the record. The migration tool omits the SQL column name for any item that is included in the EGL keyItems property or any item that is identified as read only.	**Missing FORUPDATEOF clause:** The migration tool sets the SQL column names for the for update of clause to EZE_UNKNOWN_SQL_FORUPDATEOF and issues an error message.
Missing WHERE clause: The WHERE clause is not required. The migration tool never creates a *where* clause.	**Missing WHERE clause:** The migration tool does the same thing as mentioned in the *Migrating with the associated part* column.
Missing ORDERBY clause: The ORDERBY clause is not required. The migration tool never creates an *order by* clause.	**Missing ORDERBY clause:** The migration tool does the same thing as mentioned in the *Migrating with the associated part* column.

Table 37. SQL I/O and missing SQL clauses (continued)

Migrating with the associated part	Migrating without the associated part
Potential Problem: A problem only arises if there are two records with the same name (generally in different subsystems) that have different item names or SQL column names. **Possible Solution:** Make a copy of the function for use in the second subsystem and modify the new function to use the correct item names and SQL column names. ***Disadvantage:*** This has the potential to ripple back into functions that use this I/O function.	**Potential Problem 1:** A problem arises for any modified SQL statement or any SQL statement that uses Execution Time Statement Build. **Solution 1A:** Review the list of error messages for any messages related to missing SQL clauses. Modify the SQL I/O function to include the missing clauses. The information you need to build the missing clause is in the corresponding row in the *Migrating with the associated part* column. **Solution 1B:** Edit the function in VisualAge Generator and use the SQL Editor to make a trivial change such as adding a blank at the end of a line. Save the SQL clauses and then migrate the function again. Be sure to include the record definition so that the migration tool can include the SQL table information in the EGL I/O statement. **Potential Problem 2:** The same potential problem and solution as shown for *Migrating with the associate part* apply.

SQL I/O and !itemColumnName

VisualAge Generator: For SQL I/O, VisualAge Generator permits the use of !itemColumnName in some clauses of the SQL statements. Test and generation determine the SQL column name that corresponds to the item name in the SQL row record.

EGL: The use of !itemColumnName is not supported.

Associated part needed for migration: The SQL record and the record specified as the alternate specification record, if any.

Table 38. Ambiguous situations for functions—SQL I/O and !itemColumnName

Migrating with the associated part	Migrating without the associated part
Based on the first migration of this function, if the SQL record and its alternate specification record are available, the migration tool converts any !itemColumnNames to the corresponding SQL column name based on the SQL record or, if applicable, its alternate specification record.	If the SQL record or its alternate specification record are not available, the migration tool does the following: • Uses !itemColumnNames for any column name variables. • Issues an error message that the function needs to be reviewed.
Potential Problem: A problem only arises if there are two records with the same name (generally in different subsystems) that have different SQL column names corresponding to an !itemColumnName. **Possible Solution:** Make a copy of the function for use in the second subsystem and modify the new function to use the correct SQL column names. ***Disadvantage:*** This has the potential to ripple back into functions that use this I/O function.	**Potential Problem 1:** A problem arises for any modified SQL statement or any SQL statement that uses Execution Time Statement Build. **Solution:** Review the list of error messages for any messages related to !itemColumnNames. Modify the SQL I/O function to include the correct column names based on the SQL row record. **Potential Problem 2:** The same potential problem and solution as shown for *Migrating with the associated part* apply.

SQL I/O with multiple updates

VisualAge Generator: For SQL I/O, if there are multiple UPDATE or SETUPD functions in a program, each SQL REPLACE function must specify the name of its corresponding UPDATE or SETUPD function. This is not required for non-SQL I/O. SETUPD is not supported for non-SQL I/O.

EGL: For SQL I/O, if there are multiple get for update or open for update statements, each SQL replace statement must specify the name of its corresponding get or open statement. Each get and open statement specifies a resultSetID. The replace statement specifies the resultSetID for the corresponding get or open statement. The resultSetID is not applicable for non-SQL I/O.

Associated part needed for migration: The record that is the I/O object.

Table 39. Ambiguous situations for functions—SQL I/O with multiple updates

Migrating with the associated part	Migrating without the associated part
Based on the first migration of this function, if the record is available, the migration tool creates the corresponding EGL statement(s) based on the record type. For SQL, the migration tool does the following: • Always includes a resultSetID when migrating any UPDATE or SETUPD function. The resultSetID is created using the function name and a customer-specified suffix. • Includes the resultSetID when migrating any REPLACE function that specified a corresponding UPDATE or SETUPD function name. The resultSeID is created using the corresponding UPDATE or SETUPD function name and a customer-specified suffix. For non-SQL, the migration tool always omits the resultSetID when migrating an UPDATE or REPLACE function. There are no SETUPD functions for non-SQL I/O.	When migrating an UPDATE function, if the record is not available, the migration tool does the following: • Attempts to determine if this function is for SQL I/O by checking if the function also has SQL clauses or any SQL-specific information such as Execution Time Statement Build, single row select, or cursor with hold. • If the migration tool can determine that this UPDATE statement is for an SQL record, the migration tool includes the resultSetID in the get statement. • Otherwise, the migration tool does not include the resultSetID. The migration tool issues a warning message. When migrating a SETUPD function, the migration tool always includes the resultSetID because SETUPD is only valid for SQL. When migrating a REPLACE function, the migration tool includes the resultSetID if the function specifies a corresponding UPDATE or SETUPD function name.
Potential Problem: None.	**Potential Problem:** A problem only arises if an unmodified UPDATE function really does refer to an SQL record and is used in a program where there are multiple *get* or *open for update* statements. In this case, each *replace* statement will include a resultSetID, but the *get* statement that was migrated for the VAGen UPDATE statement will not include the resultSetID. Generation for the program will fail. **Solution:** Modify the function to include the resultSetID for the *get* statement.

Handling ambiguous situations for other statements

Implicit data items in statements

VisualAge Generator: VisualAge Generator permits, but does not recommend, the use of implicit data items (items that are not explicitly defined in a record, map, table, called parameter list, function parameter list, or function local storage).

EGL: EGL does not permit implicit items.

Associated part needed for migration: Not applicable.

Table 40. Implicit data items in statements

Migrating with the associated part	Migrating without the associated part
See "Implicit data items in programs" on page 63	See "Implicit data items in programs" on page 63

Level 77 items in statements

VisualAge Generator: Only working storage records can contain level 77 items. A program can reference level 77 items only in the primary working storage record.

EGL: Level 77 items are not permitted.

Associated part needed for migration: When migrating a function, you need the working storage record.

Table 41. Level 77 items in statements

Migrating with the associated part	Migrating without the associated part
See "Level 77 items in records" on page 48	See "Level 77 items in records" on page 48

Assignment statements

VisualAge Generator: Assignment statements are permitted for records and maps and result in a "move corresponding." *MOVE* statements are permitted for items.

EGL: Assignment statements can only be used for data items or for a byte-by-byte move of a record. Assignment statements cannot be used for maps. The *move by name* statement is required for a *move corresponding* of records and maps. The *move* statement without a modifier can be used for items, but assignment statements are preferred.

Associated part needed for migration: Not applicable.

Table 42. Assignment statements

Migrating with the associated part	Migrating without the associated part
To preserve as much common code as possible, the migration tool does the following if both the source and target of an assignment or move statement are unqualified, unsubscripted names: • Checks the function's parameter list, local storage, and I/O object to try to determine whether the source or target of an assignment or MOVE statement is an item, record, or map. If the migration tool can make the determination, it migrates as follows: – To an assignment statement if the source or target is an item. – To a *move by name* statement if the source or target is a record or map. • If the migration tool cannot determine the part type, it migrates assignment and MOVE statements to a move statement without a modifier.	This is handled the same as mentioned in the *Migrating with the associated part* column.

Table 42. Assignment statements (continued)

Migrating with the associated part	Migrating without the associated part
Potential Problem: None. Test and generation convert the move statement without a modifier to a VAGen MOVE statement. This is an item to item move or a move by name (move corresponding), depending on the actual source and target of the move. Any program can use the function without modifying it.	**Potential Problem:** None. The same situation mentioned in the *Migrating with the associated part* column applies.

FIND statement

VisualAge Generator: The search column in the FIND statement is optional. The default is the first column of the table.

EGL: The FIND statement is replaced by an *if* statement. The search column is required.

Associated part needed for migration: The table.

Table 43. FIND statement

Migrating with the associated part	Migrating without the associated part
Based on the first migration of this function, if the search column is not explicitly specified and the table is available, the migration tool expands the table to get the name of search column from the first column of the table.	If the search column is not explicitly specified and the table is not available, the migration tool does the following: • Sets the search-column to `EZE_UNKNOWN_SEARCH_COLUMN` • Issues an error message that the function will need to be modified with the proper column name.
Potential Problem: A problem only arises if two tables, probably in different subsystems, have the same table name, but different search column names. **Solution:** For the second subsystem, add a data item as a substructure for the first column in the table. The name of this new data item should be the same as the search column in the first subsystem. This technique enables you to share the common function without changing any code in the second subsystem.	**Potential Problem 1:** The search column name must be provided. There will be an error in the Problems view. **Solution:** Edit the function and specify the correct column name for the table. **Potential Problem 2:** The same potential problem and solution as shown for *Migrating with the associated part* apply.

RETR statement

VisualAge Generator: The search and return columns for the RETR statement are optional. The search column defaults to the first column of the table. The return column defaults to the second.

EGL: The RETR statement is replaced by an *if* statement. The search and return columns are required.

Associated part needed for migration: The table.

Table 44. RETR statement

Migrating with the associated part	Migrating without the associated part

Based on the first migration of this function, if the search or return column is not explicitly specified and the table is available, the migration tool expands the table to get the following:
• The name of search column from the first column of the table.
• The name of the return column from the second column of the table.
	If the search column or return column is not explicitly specified and the table is not available, the migration tool does the following:
	• Sets the search column to
	`EZE_UNKNOWN_SEARCH_COLUMN`
	• Sets the return column to
	`EZE_UNKNOWN_RETURN_COLUMN`
	• Issues an error message that the function will need to be modified with the proper column names.

Potential Problem: A problem only arises if two tables, probably in different subsystems, have the same table name, but different search and/or return column names. **Solution:** For the second subsystem, add a data item as a substructure for the first column in the table. The name of this new data item should be the same as the search column in the first subsystem. Substructure the second column of the table with the name of the return column in the first subsystem. This technique enables you to share the common function without changing any code in the second subsystem.	**Potential Problem 1:** The search and return column names must be provided. There will be an error in the Problems view for each missing column. **Solution:** Edit the function and specify the correct column names for the table. **Potential Problem 2:** The same potential problem and solution as shown for *Migrating with the associated part* apply.

SET map PAGE statement

VisualAge Generator: *SET map PAGE* is used for both display and print maps.

EGL: Two separate statements are used. The map name is not specified:

- *clearScreen()* for text (display) forms
- *pageEject()* for print forms

Associated part needed for migration: The map is needed to determine the device type. The first map with this map name in any available map group is the map that the migration tool uses. When migrating in program context, the migration tool only looks at the program's main map group.

Table 45. SET map PAGE statement

Migrating with the associated part	Migrating without the associated part
Based on the first migration of this function, the migration tool converts SET map PAGE to the following: • *clearScreen()* for a text form • *pageEject()* for a print form The migration tool also includes a comment with the original map name.	If the map is not available, the migration tool does the following: • Converts SET map PAGE to EZE_SETPAGE(). • Includes a comment with the original map name. • Issues an error message that it was unable to determine the map type.

Table 45. SET map PAGE statement (continued)

Migrating with the associated part	Migrating without the associated part

Potential Problem: Any program that uses a different map type from what was determined when the function migrated *might behave differently at run time*. This is because clearScreen only applies to text forms and pageEject only applies to print forms. No error will appear in the Problems view. Generation will not fail for the program.
 	
Possible Solution: If a specific target environment does printing and other environments always use display maps, change the EGL function to something similar to the following::
`if (sysVar.systemType is zosbatch)`
`  pageEject();`
`else`
`  clearScreen();`
`end`
 	
Similar logic can be used based on transaction code, user ID, and so on, depending on the specific details of your system.
	Potential Problem 1: EGL validation results in a message in the Problems view.
	
	Solution: Edit the function and change EZE_SETPAGE() to either *clearScreen()* or *pageEject()*, depending on the map type.
	
	Potential Problem 2: The same potential problem and solution as shown for *Migrating with the associated part* apply.

SET mapItem attributes

VisualAge Generator: VisualAge Generator tolerates attributes such as protect, highlighting, and color for variables and constants on printer maps.

EGL: With the exception of underline, EGL does not support attributes for print forms.

Associated part needed for migration: Not applicable.

Table 46. SET mapItem attributes

Migrating with the associated part	Migrating without the associated part
When migrating a printer map, the migration tool omits attributes that are not supported by EGL for print forms. When migrating a function, the migration tool migrates the SET statement without regard to whether the map is a display map or printer map.	The migration tool does the same thing as mentioned in the *Migrating with the associated part* column.
Potential Problem: There is no problem for a text form. A problem only arises if the function includes logic to set attributes such as color, highlight, or protect for a print form. There will be an error in the Problems view. **Solution:** If the function is only used for print forms, modify the function to remove the set statement. If the function is used with both text and print forms, make a copy of the function for use with print forms. Modify the new function to remove the set statements and use this new function for any print forms. ***Disadvantage:*** This has the potential to ripple back into functions that use the function with the set statement.	**Potential Problem:** The same potential problem and solution as listed in the *Migrating with the associated part* column apply.

Checking for IN literal or scalar

VisualAge Generator: VisualAge Generator supports the IF or WHILE statement checking for a data item IN a literal or scalar. In this situation, VisualAge Generator sets the value of EZETST and does a comparison for equality.

EGL: EGL does not support checking a data item for IN a literal or scalar.

Associated part needed for migration: Not applicable.

Table 47. Checking for IN literal or scalar

Migrating with the associated part	Migrating without the associated part
For an IF or WHILE statement that checks a data item IN a literal, the migration tool does the following to match the VAGen behavior: • Adds a statement to initialize sysVar.arrayIndex to 0. • Changes the *if* or *while* statement to compare equal (For example, if a = ″b″). • Adds a statement immediately after the *if* or *while* to set sysVar.arrayIndex to 1. For an IF or WHILE statement that checks a data item IN another data item, the migration tool does not attempt to determine if the second data item is an array or a scalar. The migration tool migrates to an EGL *in* comparison. (For example: if a in b).	The migration tool does the same thing as mentioned in the *Migrating with the associated part* column.
Potential Problem: There is no problem if the comparison is for a literal. A problem only arises if the second data item is actually a scalar. In this case, there will be an error in the Problems view. **Solution:** Modify the function to initialize sysVar.arrayIndex to 0 before the *if* or *while* statement and to set sysVar.arrayIndex to 1 immediately after the *if* or *while* statement. Also change the *if* or *while* statement to compare using = rather than *in*.	**Potential Problem:** The same potential problem and solution as listed in the *Migrating with the associated part column* apply.

Checking SQL and map items for NULL

VisualAge Generator: IF, WHILE, and TEST support checking either an SQL item or a map item for NULL.

EGL: SQL items can be checked for null. Map items can be checked for blanks.

Associated part needed for migration: The record or map. If the item is not qualified, you need the program and all of its associates.

Table 48. Checking SQL and map items for NULL

Migrating with the associated part	Migrating without the associated part
Based on the first migration of this function, if the item is qualified, the migration tool does the following: • Checks the qualifier to determine if it is a record or map. • Converts to checking for *null* if the qualifier is an SQL record. • Converts to checking for *blanks* if the qualifier is a map.	The migration tool tries to determine the type of the item as follows: • If the item is qualified and the qualifier is not available, the migration tool does the following: – Checks if the qualifier is also the function's I/O object. If so, the CONVERSE and DISPLAY I/O options guarantee the I/O object is a map. The CLOSE I/O option is valid for either a record or map. Other I/O options guarantee the I/O object is a record. – Also checks the function's parameter list and local storage. If the qualifier is found, the qualifier is a record. • If the migration tool can determine that the item is in an SQL record or on a map, the tool migrates to the following: – *null* for an SQL record – *blanks* for a map item • If the migration tool cannot determine that the item is in an SQL record or on a map, then the tool does the following: – Converts to EZE_NULL. – Issues an error message indicating that this statement should be reviewed.
Based on the first migration of this function, if the item is not qualified, the migration tool does the following: • Checks the function's parameter list to see if the item is specified there as either an SQLITEM or a MAPITEM parameter. If so, the tool migrates on that basis. • If the program and its associates are available, the migration tool uses the VAGen qualification rules to determine which record or map contains the item and then migrates on that basis.	If the item is not qualified, the migration tool checks the function's parameter list to see if the item is specified there as either an SQLITEM or a MAPITEM. If the migration tool can determine that the item is in an SQL record or on a map, the tool migrates to the following: • *null* for an SQL record • *blanks* for a map item If the migration tool cannot determine that the item is in an SQL record or on a map, then the tool does the following: • Converts to EZE_NULL. • Issues an error message indicating that this statement should be reviewed.
Potential Problem: None.	**Potential Problem 1:** A problem arises if the migration tool uses EZE_NULL. There will be an error in the Problems view. **Solution:** Edit the function and change EZE_NULL to *null* for an SQL item or *blanks* for a form variable field.

I/O error values UNQ and DUP

VisualAge Generator: UNQ and DUP are always soft errors for non-SQL and hard errors for SQL. UNQ and DUP are always set for SQL based on the -803 SQL code. If an I/O error routine is specified for the function, the error routine gets control for the following:

- any soft error
- any hard error if EZEFEC = 1

EGL: Duplicate is always a soft error and indicates the I/O was successful. Unique is always a hard error and indicates the I/O failed. Duplicate is not supported for SQL. The *try* block and *onException* statement are used for error handling. If an onException statement is specified for the I/O statement, the onException statement gets control for the following:

- any soft error
- any hard error if handleHardIOErrors = 1

Associated part needed for migration: The record that is used in the statement.

Table 49. I/O error values UNQ and DUP

Migrating with the associated part	Migrating without the associated part
Based on the first migration of this function, if the record is available, the migration tool does the following: • If the record is non-SQL, the migration tool changes DUP to *duplicate* and UNQ to *unique*. • If the record is SQL, the migration tool changes both DUP and UNQ to *unique*.	If the record is not available, the migration tool tries to determine the type of the record as follows: • If the statement specifies the same record as the function's I/O object, the migration tool checks to see if the function also has SQL clauses, or any SQL-specific information, such as Execution Time Statement Build, single row select, cursor with hold or an UPDATE/SETUPD function. If so, the migration tool assumes that the record is SQL and converts DUP and UNQ to *unique*. • In other situations such as the following, the migration tool cannot determine the record type: – If the record is used as the I/O object of the function but the function does not have SQL-specific information. – If the record is not used as the I/O object of the function. In the previous situations, and in other situations when the migration tool cannot determine the record type, the migration tool does the following: – Converts UNQ to *unique*. – Converts DUP to *EZE_DUPLICATE* and issues an error message.

Table 49. I/O error values UNQ and DUP (continued)

Migrating with the associated part	Migrating without the associated part
Potential Problem: A problem only arises if the same record name has different definitions, one for SQL and one for non-SQL, most likely in different subsystems. If the non-SQL record is available when the function is migrated, then there will be an error if the function is used with an SQL record and checks for duplicate. If the SQL record is available when the function is migrated, then the additional information conveyed by the duplicate check will not be available for the non-SQL record. **Possible Solution:** Copy the function and use the original function for SQL and the new function for non-SQL. Disadvantage: This has the potential to ripple back into functions that use the original function that checked for UNQ or DUP. **Potential Problem for SQL:** None. DUP and UNQ were always set the same way and *unique* continues to be a hard error. **Potential Problem 1 for non-SQL:** A problem arises if you do *not* set handleHardIOErrors (EZEFEC) = 1 for the program. In this case, because *unique* is now a hard error, the onException statement will not get control and the program will end. **Solution:** Make sure your programs specify *handleHardIOErrors = 1;* **Potential Problem 2 for nonSQL:** A problem also arises if you are explicitly testing for hardIOError (HRD). In this case, because *unique* is now a hard error, hardIOError will test true in EGL in some cases, even though it did not test true in the past on VisualAge Generator. Validation and generation will not detect an error. *However, the program might not run the same as it did in VisualAge Generator.* **Possible Solution:** You might need to reorder the testing of the I/O error values in your program logic.	**Potential Problem 1:** EZE_DUPLICATE is not valid in EGL. **Solution:** Edit the function and change EZE_DUPLICATE to *duplicate* or *unique* based on the record type. **Other Potential Problems:** The same potential problems and solutions as shown for *Migrating with the associated part* apply.

I/O error value LOK

VisualAge Generator: LOK is always a soft error for OS/400®. If an I/O error routine is specified for the function, the error routine gets control for the following:

- any soft error
- any hard error if EZEFEC = 1

EGL: *LOK* is replaced by *deadlock*, but it is a hard error. The *try* block and *onException* statement are used for error handling. If an onException statement is specified for the I/O statement, the onException statement gets control for the following:

- any soft error
- any hard error if handleHardIOErrors = 1

Associated part needed for migration: Not applicable.

Table 50. I/O error value LOK

Migrating with the associated part	Migrating without the associated part
The migration tool always changes *LOK* to *deadlock*.	The migration tool does the same thing as mentioned in the *Migrating with the associated part* column.
Potential Problem 1: A problem arises if you do *not* set handleHardIOErrors (EZEFEC) = 1 for the program. In this case, because deadlock is a hard error, the *onException* statement will not get control and the program will end. **Solution:** Make sure your programs specify *handleHardIOErrors = 1;* **Potential Problem 2:** A problem also arises if you are explicitly testing for hardIOError (HRD) . In this case, because *deadlock* is a hard error, hardIOError will test true in EGL in some cases where it did not test true in VisualAge Generator. Validation and generation will not detect an error. *However, the program might not run the same as it did in VisualAge Generator.* **Possible Solution:** You might need to reorder the testing of the I/O error values in your program logic.	The same potential problems as in the *Migrating with the associated part* column can occur. You can use the same solutions.

XFER

VisualAge Generator: The XFER statement can be used with both maps and UI records to send output to the user and then transfer to another program or transaction when the user enters input data.

EGL: The *show* statement is used for sending a form to the user at a terminal and then transferring to another program or transaction when the user enters input data. This release of EGL does not support web transactions, UI records, or a replacement for XFER with a UI record. However, the *forward* statement is expected to be the replacement for XFER with a UI record. The *forward* statement is used for sending a UI record to a user in a browser window and then transferring to another program or transaction when the user enters input data.

Associated part needed for migration: Map that is used in the XFER statement. If a map with the name specified on the XFER statement is not available, the migration tool converts the XFER statement as described in the *Migrating without the associated part* column of the following table.

Table 51. XFER

Migrating with the associated part	Migrating without the associated part
Based on the first migration of this function, the tool migrates as follows: • If the second argument is a map, the tool migrates to a *show* statement. • If the second argument is a UI record, the tool migrates to a *forward* statement.	If the map or UI record is not available, the migration tool does the following: • If (NONCSP is included on the XFER statement, the tool migrates to a *show* statement. • If the target of the XFER is ' ', the tool migrates to a *forward* statement. • If the name of the second argument is longer than 8 characters, the tool migrates to a *forward* statement because only UI records can have a name longer than 8 characters. • If the name of the second argument is 8 or fewer characters, the tool migrates to the *forward* statement because XFER with a UI record is more common than XFER with a map. The tool also issues a warning message.
Potential Problem: A problem arises for any program that uses a UI record when the migration tool migrated based on a map or vice versa. Generation for the program will fail. **Solution:** If the function was migrated based on a map, create a second function that contains the forward statement for the UI record. Change any program that uses a UI record to use this new function. Disadvantage: This has the potential to ripple back into any function that invokes the original function.	The same problem listed under the *Migrating with the associated part* column can occur. You can use the same solution.

Handling ambiguous situations for EZE words

For some EZE Word replacements, an extra data item must be defined in the program. The extra data item is never defined as a local item in the function because a segmented converse cannot be done if there is any function open in the stack down to the segmented converse that has local storage, parameters or return values. Adding the extra data item to the program avoids breaking any segmented converse.

EZESYS

VisualAge Generator: *EZESYS* is generally used in IF, WHILE, and TEST statements with literal values specified by VisualAge Generator. However, EZESYS is permitted in other statements.

EGL: The EGL system variable *systemType* has different values from VisualAge Generator. When EZESYS is used in statements other than IF, WHILE, and TEST, the migration tool does not know what values the program might be expecting and so must use the original VAGen values. The EGL system library function *getVGSystemType* provides the old VAGen values.

Associated part needed for migration: Not applicable.

Table 52. EZESYS

Migrating with the associated part	Migrating without the associated part
When migrating any program, the migration tool always does the following: • Includes the declaration: `<custPrefix>EZESYS` • Includes a statement to initialize the value of `<custPrefix>EZESYS` to the old VAGen value using the system library function.	The migration tool does the same thing as mentioned in the *Migrating with the associated part* column.

Based on the first migration of the function, the migration tool does the following:
• If EZESYS is used in an IF, WHILE, or TEST statement, the migration tool converts EZESYS to
`sysVar.systemType`
The migration tool converts the EZESYS values to their EGL equivalent value. If the EZESYS value does not have an equivalent EGL value, the migration tool migrates it "as is". For example, the migration tool converts MVSBATCH to the EGL equivalent zosbatch. The migration tool migrates OS2 and NTCICS to the same value as in VisualAge Generator. See Table 101 on page 231 for specifics of which values are converted.
• If EZESYS is used in any other statement, the migration tool does the following:
– Issues a warning message that this use will result in the old VAGen EZESYS values
– Uses
`<custPrefix>EZESYS`
to replace EZESYS in the statement.
	The migration tool does the same thing as mentioned in the *Migrating with the associated part* column.

Potential Problem 1: A problem arises for EZESYS values that migrate as they are and for the EGL equivalent values (for example, imsvs and imsbmp) that are not supported in this release. There will be an error in the Problems view. **Possible Solution 1:** Modify the function and change the logic so that sysVar.systemType is not checked for values that are not valid in EGL. **Potential Problem 2:** A problem arises if you want to use the new EGL values in statements other than if and while. **Possible Solution 2:** Modify the function and change the logic to use sysVar.systemType instead of `<custPrefix>.EZESYS` Be sure to change the old VAGen values to the new EGL values in any data tables that you use for comparisons	The same potential problems mentioned in the *Migrating with the associated part* column apply. You can use the same solutions.

EZEWAIT

VisualAge Generator: EZEWAIT specifies the time to wait in hundredths of a second.

EGL: *sysLib.wait*, which is the replacement for EZEWAIT, specifes the time to wait in seconds.

Associated part needed for migration: Not applicable.

Table 53. EZEWAIT

Migrating with the associated part	Migrating without the associated part
When migrating any program, the migration tool always includes a declaration for `<custPrefix>EZE_WAIT_TIME.`	The migration tool does the same thing as mentioned in the *Migrating with the associated part* column.
When migrating a function, if EZEWAIT is used, the migration tool includes logic to calculate the time to wait in seconds and stores the result in `<custPrefix>EZE_WAIT_TIME.`	The migration tool does the same thing as mentioned in the *Migrating with the associated part* column.
Potential Problem: None. However, if you use the function in a new program, be sure to include a declaration for `<custPrefix>EZE_WAIT_TIME` in the program.	The same potential problem mentioned in the *Migrating with the associated part* column applies.

Part 2. Migrating from VisualAge Generator 4.5 on Java to EGL

Chapter 4. Stage 1 — Extracting from Java

Before you can extract your source code from VisualAge Generator, you must install the Stage 1 migration tool that runs on VisualAge for Java. You must also create the DB2 migration database that is used to store the data you are migrating from VisualAge Generator 4.5 (VAGen 4.5) to EGL.

Installing the Stage 1 migration tool on VisualAge for Java

The VisualAge Generator to EGL Stage 1 migration tool is shipped as a self-extracting file called VAGenMigJava.exe To install this file, do the following:

1. Upgrade to VisualAge Generator 4.5 with Fix pack 4. You must also install APAR PQ88461 to upgrade the VisualAge Generator Utilities feature. Contact IBM Support to obtain the fix for the APAR or check the VAGen web site at: http://www.ibm.com/software/awdtools/visgen/support/ and then follow the link in the Download section. Also review Appendix F, "Situations where incorrect External Source Format causes problems in creation of EGL," on page 307 for additional VisualAge Generator APARs that might be necessary for your specific situation.
2. On your system, determine where VisualAge for Java is installed.
3. Run the self-extracting file VAGenMigJava.exe, which is in the following subdirectory of your Rational Developer installation directory:

   ```
   \bin
   ```

   Note: If you installed and kept a Rational Developer product before installing the product that you are using now, the installation directory of interest may be the directory that was used in the earlier install.
4. When the GUI prompt appears, navigate to the drive and directory where VisualAge for Java is installed. (For example, c:\Program Files\IBM\VisualAge for Java.) Then click **Extract**.

When the self-extracting executable runs, it extracts the following files into your VisualAge for Java installation directory:

- \ide\vgmigration\MigPreferences.xml
- \ide\vgmigration\createdatabase.sql
- \ide\vgmigration\createtables.sql
- \ide\vgmigration\SetupDatabase.bat
- \ide\vgmigration\SetupTables.bat
- \ide\vgmigration\deletemigsets.bat
- \ide\features\com-ibm-vgj-mig\

This last directory contains the feature for the Stage 1 migration tool on Java. It also contains the .xml files and their corresponding .dtd files that are used by the Stage 1 migration tool on Java.

Adding the migration feature

To be able to use the Stage 1 migration tool, you must add the IBM VisualAge Generator EGL Migration feature. To do this, perform the following steps:

1. Start VisualAge Generator on Java.

2. Add the ″IBM VisualAge Generator EGL Migration″ feature as follows:
   a. From the Workbench window, press F2.
   b. Select **Features** in the left column and then **Add Feature** in the right column. Click **OK**.
   c. Select **IBM VisualAge Generator EGL Migration -** *versionNumber*. Click **OK**. The migration feature will be loaded.
   d. Click the **Projects** tab in the Workbench. You should see the ″IBM VisualAge Generator EGL Migration″ project in your workspace.

Creating the migration database

See “Creating the DB2 migration database” on page 309 for information on creating the migration database. You need to use the SetupDatabase.bat and SetupTables.bat files that were placed in your VisualAge Java installation directory, in subdirectory \ide\vgmigration directory.

Setting Stage 1 preferences

When you installed the Stage 1 migration tool on VisualAge for Java, the installation process created a sample preferences file called MigPreferences.xml in the directory *VisualAge-Java-installation-directory*\ide\vgmigration. You should make a copy of the MigPreferences.xml file for backup purposes before you modify any preferences.

You can use a text editor or the GUI editor that is provided with the Stage 1 migration tool to edit the MigPreferences.xml file. To use the GUI editor, do the following:

1. Start VisualAge Generator for Java.
2. In the Workbench window, click on the **Projects** tab.
3. Navigate to the *IBM VisualAge Generator EGL Migration* project.
4. Expand the migration project and then expand the package *com.ibm.vgj.mig*.
5. Within the package, select the **PreferencesUI** class.
6. Right-click on the **PreferencesUI** class and then click **Properties** from the context menu.
7. Select the **Program** tab.
8. On the Program page, specify the following in the Command line arguments field to point the MigPreferences.xml file you want to edit:

   ```
   -p filename
   ```

   where *filename* is the drive, directory, and file name of your MigPreferences.xml file.
9. Click on **OK** to save the properties.
10. Right-click on the **PreferencesUI** and then click **Run** or **Run main...** (Or you can pick the running man icon from the tool bar.) The Stage 1 GUI preferences editor opens and loads the file that you pointed to in the program properties.

Note:

- For preferences that require a drive and directory, you can specify the information in either of two ways:
  - An absolute path. For example: d:\tempMig\MySystem\
  - A relative path. In this case the path is relative to the working directory. For example, ...\tempMig\MySystem results in a path of :

VisualAge-Java-installation-directory\ide\project_resources
\IBM VisualAge Generator EGL Migration\tempMig\MySystem.

- If you do not specify a drive and directory for the log, debug, and report files, the files are written to the working directory which is:

  VisualAge-Java-installation-directory\ide\project_resources
  \IBM VisualAge Generator EGL Migration

The preferences you can modify are described in the following sections, based on the page within the GUI in which the preference appears:

- Build Plans
- Mapping
- Renaming
- Execution

Build Plans page

The Build Plans page identifies where the Stage 1 migration tool is to read or write the migration plan file (or files), as well as which projects and versions you want to migrate from your repository.

- **Migration Specification.** The Migration Specification identifies where the migration tool is to write the migration plan file or files that the Stage 1 tool creates based on your repository filters. Alternatively, if you have already created the migration plan file (or files), the Migration Specification identifies where the migration tool is to read the migration plan file (or files).

  Note:

  – Migration plan files have the file extension *.pln* before they are used to load the migration database and *.done* after they have been successfully processed.

  – See "Running the Stage 1 tool" on page 100 for information on setting the *-o* (override) option for the VAGenToEGLMigration class, which is the actual Stage 1 migration tool.

– *Plan directory.* This is the target directory where you want your migration plan file (or files) to be placed by the Stage 1 migration tool or in which the Stage 1 tool can find your existing migration plan file (or files).

– *Plan file name.* An optional file name of the migration plan file you are creating or using to load the migration database. When you run the Stage 1 migration tool, this file name is used in conjunction with the *-o* (override) option you specify for the VAGenToEGLMigration class as follows:

  - If you include the *-o* option in the properties for the VAGenToEGLMigration class, the Stage 1 migration tool does the following based on the file name you specify in the Migration Specifications:

    • If you do not specify a Plan file name, the migration tool deletes **all** the .pln files in the specified Plan directory before creating new plan files. The migration tool creates one plan file for each migration set. In this case, the migration Plan file names are of the form *migrationSetName_version*.pln.

    • If you specify a Plan file name, the migration tool deletes **only** the specified .pln file from the specified Plan directory before creating a new .pln file with your specified Plan file name. In this case, the single Plan file lists all the migration sets.

  Use the *-o* option if you want the Stage 1 migration tool to create the migration plan file (or files) for you based on your repository filters.

– If you omit the *-o* option from the properties for the VAGenToEGLMigration class, the Stage 1 migration tool does not create any new migration plan files. Instead, the Stage 1 migration tool runs based on the Plan directory and Plan file name you specify in the Migration Specification:
  - If you do not specify a Plan file name, the migration tool runs using **all** of the .pln files in the specified Plan directory.
  - If you specify a Plan file name, the migration tool runs using only that one .pln file in the specified Plan directory.

Omit the *-o* option if you have previously created the migration plan files and now want to run the Stage 1 migration tool to load the migration database using these files. See "Creating a migration plan file manually" on page 103 for details about creating your own migration plan files.

- **Repository filters.** The Repository Filters section enables you to control which projects and versions in your Java repository are considered by the Stage 1 migration tool. Limiting the projects and versions can greatly enhance the performance of the Stage 1 migration tool. You can specify multiple filters. The Stage 1 migration tool uses the *Projects* filter and the *Version depth* or *Version name* filters as follows:
  – The migration tool matches each VAGen project in the repository against the Projects filters.
    - If the project name does not match at least one of the Projects filters, the project is not considered for further processing.
    - If the project name matches at least one of the Projects filters, the versions of the project are processed as follows:
      • If you selected the Version depth filter, then the most recent versions of the project, up to the number specified by the Version depth filter, are considered for further processing. The default Version depth filter is 1.
      • If you selected the Version name filter, then each version name for the project is matched against the list of Version name filters. If the version name matches any of the Version name filters, then the version is considered for further processing.

        Note: Version depth and Version name are mutually exclusive. By default, the Version name filter is included in the *MigPreferences.xml* file. If you want to use the Version depth filter, select the Version depth radio button and specify the number of versions you want to migrate.
  – If the project name and version name result in the project version being considered for further processing, the Stage 1 migration tool does the following:
    - If the project version is a high-level PLP project, then the Stage 1 migration tool uses the project version as the basis for creating a migration set. Each version of the high-level PLP project results in a different migration set, assuming the version name matches the version filter.
    - If the project version is not a high-level PLP project, the project version is not considered for further processing. The project version might still be included in other migration sets; there just will not be a migration set specifically for this project version.

  Specify the Repository Filters information as follows:
  – *Projects filter.* The migration tool matches the project names in your repository to the Projects filter (or filters) that you specify. You can specify multiple Projects filters. To add or remove filters, use the Add and Remove push

buttons. To update a filter, overtype in the table. The filters are not case sensitive. You can use wildcards as follows:

- A project filter of **xyz** matches any project name in the repository that has the string ″xyz″ anywhere in its name.
- A project filter of *xyz** matches any project name in the repository that begins with ″xyz″.
- A project filter of **xyz* matches any project name in the repository that ends with ″xyz″.

– *Version depth* filter. If a project name matches one of the Projects filters and you selected the Version depth filter, the Stage 1 migration tool processes the number of versions you have specified for the Version depth. The default is 1, in which case the Stage 1 migration tool only processes the most recent version of the project.

– *Version name* filter. If a project name matches one of the Projects filters and you selected the Version name filter, the Stage 1 migration tool uses the Version name filter (or filters) to determine which, if any, of the project versions should be considered for migration. You can specify multiple version name filters. To add or remove filters, use the Add and Remove push buttons. To update a filter, overtype in the table. The filters are not case sensitive. You can use wildcards as follows:

- A version name filter of **xyz** matches any project version name that has the string ″xyz″ anywhere in the version name.
- A version name filter of *xyz** matches any project version name that begins with ″xyz″.
- A version name filter of **xyz* matches project version name that ends with ″xyz″.

Mapping page

The Mapping page enables you to control the placement of parts in EGL files and the name of some of the EGL projects, packages, and files that are created during migration.

- **File names.** The File names section enables you to control the names of two EGL files that are created during migration.
  – *Common Parts* enables you to specify the name of an EGL file to contain parts that are common to multiple unique generatable parts within the scope of the migration set. Specify the file name without an extension or path. The migration tool creates a common parts file in each EGL package that contains parts that are used by (associated with) multiple generatable parts in the migration set or which are in VAGen projects or packages that are identified as common projects or package. See “Placing parts in EGL files” on page 32 for details about whether a part is placed with a program or in the Common Parts file.
  – *Unused Parts* enables you to specify the name of an EGL file to contain parts that are not used within the scope of the migration set. Specify the file name **without** an extension or path. The migration tool creates an unused parts file in each EGL package that contains parts that are not used by (associated with) any generatable part in the migration set, provided the corresponding VAGen project and package are not identified as common projects or packages.
- **Spanning Maps.** The Spanning Maps section enables you to specify suffixes that are used in the event that one of your map groups includes maps from multiple projects or packages.

  - *Project suffix* enables you to specify a suffix that the Stage 1 migration tool concatenates to the migration set name to create a new EGL project name. The migration tool only creates this new EGL project if a map group and its maps are in multiple VAGen projects within the migration set. The new project name is *migrationSetName_ProjectSuffix*.
  - *Package suffix* enables you to specify a suffix that the Stage 1 migration tool concatenates to a project name to create a new EGL package name within an EGL project. The migration tool only creates this new EGL package if a map group and its maps are in multiple VAGen packages within a project. The new package name is *projectName_PackageSuffix*.
- **Common Identifiers.** The Common Identifiers section enables you to specify a list of strings with wildcards that the migration tool can use in determining which VAGen projects and packages contain common (shared) parts.
  - The *Projects* list enables you to specify a list of strings that identifies projects that contain common parts. The migration tool matches this list of strings to each project name in the migration set to determine if the project contains common parts. If any string matches a project name, all parts within the project are considered to be ″used.″ Each non-generatable part will either be placed in a program file or in the file specified by your Common Parts preference. The part will not be placed in the unused parts file even if the part is not used by any generatable part in the migration set. You can specify multiple Projects filters. To add or remove filters, use the Add and Remove push buttons. To update a filter, type over it in the table. The filters are not case sensitive. You can also use an * as a wildcard at the beginning or end of the string.
  - The *Packages* list enables you to specify a list of strings that identifies packages that contain common parts. The migration tool matches this list of strings to each package name in the migration set to determine if the package contains common parts. If any string matches a package name, all parts within the package are considered to be ″used.″ Each non-generatable part will either be placed in a program file or in the file specified by your Common Parts preference. The part will not be placed in the unused parts file even if the part is not used by any generatable part in the migration set. You can specify multiple Packages filters. To add or remove filters, use the Add and Remove push buttons. To update a filter, type over it in the table. The filters are not case sensitive. You can also use an * as a wildcard at the beginning or end of the string.

Renaming page

The Renaming page enables you to specify renaming rules for your projects, packages, and version names. The *Renaming Rules* section enables you to control the names of the EGL projects and packages that are derived from your VAGen project and package names. The number in the order column indicates the order in which the Stage 1 migration tool is to apply the renaming rules, with the lowest numbered rule applied first. To add or remove a renaming rule, use the Add and Remove push buttons. To update a renaming rule, overtype the contents of the cells in the table. You can double-click on any of the column headings to sort the rules based on that column. You specify a rule by specifying the following information:

- *Order* specifies the order in which the rules are to be applied.
- *From String* specifies the characters in the VAGen name that you want to change.
- *To String* specifies the characters you want to use in the resulting EGL name.

- *String Context* specifies the location in the VAGen name where the migration tool should look for the from string during renaming. The values are as follows:
  - *front* means the rule applies if the from string appears at the beginning of a project, package, or version name.
  - *back* means that the rule applies if the from string appears at the end of a project, package, or version name.
  - *any* means that the rule applies if the from string appears anywhere within a project, package, or version name.
  - *token* means that the rule applies only if the from string is an exact match for the project, package, or version name.
- *Mapping Context* indicates whether the migration tool is to apply the renaming rule to a project, package, or version name. The values for *Mapping Context* are as follows:
  - *project* means that the renaming rule only applies to VAGen project names.
  - *package* means that the renaming rule only applies to VAGen package names.
  - *both* means that the renaming rule applies to both VAGen project names and VAGen package names.
  - *version* means that the renaming rule applies to the version names for all project names. Use a version renaming rule if your version names include special characters such as a semicolon (;) that are not permitted in directory or file names. The default *MigPreferences.xml* file includes several version renaming rules to help ensure that your version names do not result in invalid directory or file names. The migration tools use the renamed versions to create the migration plan file names in Stage 1 and to create directory names in Stage 3 of migration.

Execution page

- **Execution Options.** The Execution Options section enables you to specify what you want the Stage 1 migration tool to do.
  - *Generate report* specifies that you want to create a migration report showing where each part will be placed in the EGL project, package and file structure. This report is useful for reviewing the results of preferences you specified for Common Parts and Unused Parts file names, Spanning maps suffixes, Common Identifiers for projects and packages, and your renaming rules. If you select Generate report, the migration tool creates the report in the drive, directory and file you specify for the Report file name in the Verification section.
  - *Update database* specifies that you want the Stage 1 migration tool to store the migration plan information, including the External Source Format for your parts, into the migration database.

  You might run the Stage 1 migration tool in several steps as follows:
  - Step 1 -- Deselect both Generate report and Update database. This enables you to review the migration plan files that are created an ensure that your Repository Filters are set correctly and will process the project versions that you want. If you are not satisfied with the project versions that are being selected, you can refine your Repository Filters and run this step again until you are satisfied with the project versions that the migration tool will process.
  - Step 2 -- Select only Generate report. This enables you to review the report that shows how your VAGen projects, packages, and parts will be assigned to EGL projects, packages, and files during migration. If you are not satisfied

with the placement of parts, you can refine your Mapping and Renaming rules and run the report again until you are satisfied with the placement of parts.

– Step 3 -- Select both Generate report and Update database. This provides you with a final report that records the information that is stored in the migration database.

Note:

– Generating the report can take some time. Therefore it is best to review the .pln files to be sure that the migration tool will process the project versions that you intend.

– The report files are overwritten if a report is generated. If you want to save previous report files, you must move the report files to a different directory or point to a new directory for your new report. Because the report files link to other files, renaming the report files will cause the links to be lost so the files are no longer viewable.

- **Database.** The Database section enables you to specify details about the migration database:
  – *Database driver.* This value should always be COM.ibm.db2.jdbc.**app**.DB2Driver.
  – *Database name.* This value should always be one of the following:
    - jdbc:DB2:*databaseName* if you are using a local database.
    - jdbc:*nodeName:databaseName* if you are using a remote database.

    Note: In both cases, *databaseName* is the name of the migration database into which the migration tool is to write the migration set information. By default, the databaseName is VGMIG. If you changed the database name from VGMIG when you created the migration database, you must change the database name specified by this preference to match the name you used.
  – *Schema* is the name used as the qualifier for the database tables. By default, the schema name is MIGSCHEMA. If you changed the schema name from MIGSCHEMA when you created the migration database, you must change the schema name specified by this preference to match the name you used.
  – *Userid* is the user ID needed to connect to the migration database. If you not specify the Userid, the migration tool attempts to connect using your logon user ID. If this attempt fails, the migration tool displays a dialog window asking for the information.
  – *Password* is the password needed to connect to the migration database. If you not specify the password, the migration tool attempts to connect using your logon password. If this attempt fails, the migration tool displays a dialog window asking for the information.

    Note: The password is not encrypted in the preferences file. If this is a concern, do not enter the password in the preferences file. Wait for the prompt.
- **Service.** The Service section enables you to specify details about the logging and debug information you want to capture during Stage 1. You can specify the following:
  – *Trace level* enables you to specify the level of information that you want to write to the debug file. Use the drop-down list to specify one of the following values:

1. *Fatal* error messages are logged. If any of these messages occur, the migration database might be updated, but the migration plan file (.pln file) is not changed to have the .done file extension. This enables you to reprocess the .pln file.
2. *Warning* messages, as well as fatal error messages are logged.
3. *Informational* messages, as well as warning and fatal error messages are logged.
4. *Debug* information, as well as informational, warning, and fatal error messages are logged. DEBUG is the only trace level that causes the migration tool to write information to the debug file.

The Trace level only affects the log and debug files. All the messages are written to the Console window.

– *Log file name* enables you to specify the drive, directory, and file name for a log file. You can create the log file with any file extension, but it is best viewed as an .xml file. If you omit the log file name, the migration tool writes the log information to a file named *miglog.xml* in the drive and directory that you specified in the Log file name field. If you do not specify a Log file drive and directory, the migration tool writes the log file to the working directory.

– *Debug file name* enables you to specify the drive, directory, and file name for a debug file that might be needed by IBM support. You can create the debug file with any file extension, but it is best viewed as an .xml file. Information is only written to this file if the Trace level preference is set to *Debug*. If you omit the debug file name and you specify a Trace level of *Debug*, the migration tool writes the debug file information to a file *migdebug.xml* in the drive and directory that you specified in the Debug file name field. If you do not specify a Debug file drive and directory, the migration tool writes the debug file to the working directory.

- **Verification.** The Verification section enables you to specify the drive, directory, and file name for the verification report that is produced when you select the Generate report preference in the Execution Options section. If you select Generate report, you must enter a *Report file name*. You should always specify the .htm extension. If you do not specify a drive and directory, the migration tool writes the report file to the working directory.

Sample MigPreferences.xml file

The following is a sample MigPreferences.xml filet:

```
<preferences>
  <database>
    <driver>COM.ibm.db2.jdbc.app.DB2Driver</driver>
    <uri>jdbc:DB2:VGMIG</uri>
    <schema>MIGSCHEMA</schema>
    <userid></userid>
    <password></password>
  </database>
  <migrationSpec>
    <directory>d:\tempMig\MyMigSet</directory>
    <filename></filename>
  </migrationSpec>
  <repositoryFilters>
    <projectName>MyProject*</projectName>
    <versionName></versionName>
  </repositoryFilters>
  <service>
     <tracelevel>4</tracelevel>
     <debugfile>d:\tempMig\MyMigSet\Stage1\migdebug.xml</debugfile>
     <logfile>d:\tempMig\MyMigSet\Stage1\miglog.xml</logfile>
  </service>
```

```
<eglMapping>
   <renameRule order = "1">
     <fromString> </fromString>
     <toString></toString>
     <stringContext>any</stringContext>
     <mappingContext>both</mappingContext>
   </renameRule>
   <renameRule order = "101">
     <fromString>Project</fromString>
     <toString></toString>
     <stringContext>any</stringContext>
     <mappingContext>project</mappingContext>
   </renameRule>
   <renameRule order = "301">
     <fromString>.pkg</fromString>
     <toString></toString>
     <stringContext>any</stringContext>
     <mappingContext>package</mappingContext>
   </renameRule>
   <renameRule order = "302">
     <fromString>.sql</fromString>
     <toString>sql</toString>
     <stringContext>any</stringContext>
     <mappingContext>package</mappingContext>
   </renameRule>
   <renameRule order = "501">
     <fromString>:</fromString>
     <toString>_</toString>
     <stringContext>any</stringContext>
     <mappingContext>version</mappingContext>
   </renameRule>
   <renameRule order = "502">
     <fromString>/</fromString>
     <toString>_</toString>
     <stringContext>any</stringContext>
     <mappingContext>version</mappingContext>
   </renameRule>
   <renameRule order = "503">
     <fromString>\</fromString>
     <toString>_</toString>
     <stringContext>any</stringContext>
     <mappingContext>version</mappingContext>
   </renameRule>
   <renameRule order = "504">
     <fromString>|</fromString>
     <toString>_</toString>
     <stringContext>any</stringContext>
     <mappingContext>version</mappingContext>
   </renameRule>
   <renameRule order = "505">
     <fromString>?</fromString>
     <toString>_</toString>
     <stringContext>any</stringContext>
     <mappingContext>version</mappingContext>
   </renameRule>
   <renameRule order = "506">
     <fromString>*</fromString>
     <toString>_</toString>
     <stringContext>any</stringContext>
     <mappingContext>version</mappingContext>
   </renameRule>
   <renameRule order = "507">
     <fromString>&lt;</fromString>
     <toString>_</toString>
     <stringContext>any</stringContext>
     <mappingContext>version</mappingContext>
   </renameRule>
```

```
  <renameRule order = "508">
    <fromString>&gt;</fromString>
    <toString>_</toString>
    <stringContext>any</stringContext>
    <mappingContext>version</mappingContext>
  </renameRule>
  <renameRule order = "509">
    <fromString>&quot;</fromString>
    <toString>_</toString>
    <stringContext>any</stringContext>
    <mappingContext>version</mappingContext>
  </renameRule>
  <renameRule order = "510">
    <fromString> </fromString>
    <toString>_</toString>
    <stringContext>any</stringContext>
    <mappingContext>version</mappingContext>
  </renameRule>
  <verification>
     <generateReport>true</generateReport>
     <reportName>d:\tempMig\MyMigSet\report\MyReport.htm</reportName>
  </verification>
  <dbUpdate>true</dbUpdate>
  <spanningMapsProjectSuffix>MapsProject</spanningMapsProjectSuffix>
  <spanningMapsPackageSuffix>mapspackage</spanningMapsPackageSuffix>
  <commonPartsFileName>CommonParts</commonPartsFileName>
  <unusedPartsFileName>UnusedParts</unusedPartsFileName>
  <commonParts>
         <commonProject>*Common*</commonProject>
         <commonPackage>*common*</commonPackage>
  </commonParts>
 </eglMapping>
</preferences>
```

Before you run the Stage 1 tool — hints and tips

Before you run the Stage 1 migration tool, there are some things you might want to do to improve performance. You might also want to save your existing workspace for use after migration is completed.

Improving performance

Performance measurements have shown that the performance Stage 1 migration tool can be improved dramatically by starting with a clean workspace. In one series of tests, starting with a clean workspace reduced the time for Stage 1 to 25% - 30% of the time without a clean workspace. If your existing workspace is larger than 20 megabytes, starting with a clean workspace might help the Stage 1 tool performance.

To start with a clean workspace, do the following:

1. Shut down VisualAge Generator.
2. See “Saving your workspace” on page 100 if you want to keep a backup copy of your existing workspace to use after migration has completed.
3. Obtain a copy of a clean workspace (file name ide.icx) from the VisualAge Generator download site at:

   `ftp://ftp.software.ibm.com/ps/products/visualagegen/fixes/v4.5/FixPack4/windows`
4. Delete the *features.sav* and *projects.sav* files.
5. Restart VisualAge Generator.
6. Add the VisualAge Generator features that you need.
7. Add the ″IBM VisualAge Generator EGL Migration″ feature.

8. Shut down VisualAge Generator.

To reduce the time the Stage 1 migration tool spends analyzing which projects and versions to migrate, consider creating a repository that only contains the project versions that you want to migrate. If you have ongoing maintenance in VisualAge Generator while you are migrating, a separate migration repository also has the following advantages:

- There is a stable set of project versions to migrate. This is particularly important if you use the Version Depth preference to control what is to be migrated.
- You can compare the versions in the new migration repository against your maintenance repository to determine what additional project versions still need to be migrated.

If you do create a special repository, consider using it as a local repository to improve Stage 1 migration performance.

Saving your workspace

The Stage 1 migration tool deletes all projects that contain VAGen parts from your workspace at the beginning and end of Stage 1 processing. This helps to avoid duplicate parts in the workspace and ensures that only parts in the migration set are considered for the associate parts list during Stage 1. If you have a workspace that you wish to save, you should do the following before running the Stage 1 tool:

1. Shut down VisualAge Generator.
2. Save backup copies of the following files in your *VisualAgeForJava-installation-directory*\ide\program:
   - features.sav
   - projects.sav
   - ide.icx
   - ide.ini — not necessary to save if you do not change any preferences while running Stage 1
   - hpt.ini — not necessary to save if you do not change any preferences while running Stage 1
3. Start VisualAge Generator.

When you are finished running the Stage 1 tool, do the following to restore your workspace:

1. Shut down VisualAge Generator.
2. Restore the files you backed up before running the Stage 1 tool.
3. Start VisualAge Generator.

Running the Stage 1 tool

After you have finished editing your preferences, you are ready to run the Stage 1 migration tool to extract your source code from the Java repository. To do this, perform the following steps:

1. Navigate to the *IBM VisualAge Generator EGL Migration* project.
2. Expand the migration project and then expand the package *com.ibm.vgj.mig*.
3. Within the package, select the **VAGenToEGLMigration** class.
4. Right-click on the **VAGenToEGLMigration** class and then click **Properties** from the context menu.

5. Select the **Program** tab.
6. On the Program page, specify the following in the Command line arguments field to point the MigPreferences.xml file you want to edit:

Table 54. Valid command line options for VAGentoEGLMigration class

Option	Meaning of option
—h	Display help information that shows the valid options
—p filename	Use "filename" as the name of the preferences file. You must fully qualify the file name, including the drive and directory.
—o	Overwrite the migration files if they exist and recreate them.

7. If this is the first time you are running the Stage 1 tool, do the following:
   a. In the same Properties window, select the **Class Path** tab.
   b. On the Class Path page, select the **Extra directories path** check box and then click on the **Edit** button for the Extra directories.
   c. Select the **Add Jar/Zip** button.
   d. In the File selection window, navigate to and select the db2java.zip file.
      - If you used the default install directory when you installed DB2, the file should be in the \SQLLIB\java directory.

      After you select the db2java.zip file, the file name appears in the Extra directories window. Click **OK** on the Extra Directories window.
   e. On the Class Path page, click the **Compute Now** button and then click **Yes** at the prompt.
8. Click on **OK** to save the properties.
9. Right-click on the **VAGenToEGLMigration** and then click **Run** or **Run main...** (Or you can pick the running man icon from the tool bar.) The Stage 1 migration tool starts and opens a Console window where it reports progress and any error messages. The migration tool also writes the messages to the log file you specified in your migration preferences.

When the Stage 1 migration tool finishes, if you selected the Update database preference, then your migration plan information, including your VAGen code in External Source Format, is stored in the migration database. After reviewing your report and the Stage 1 messages, you might decide to make changes to your code in VisualAge Generator and run Stage 1 again. After you are satisfied with the results of Stage 1 and have your final External Source Format code stored in the migration database, you are ready to perform Stage 2 of the migration. To run the Stage 2 migration tool, you use the Rational Developer product. See Chapter 6, "Stage 2—Conversion to EGL syntax," on page 127 for information about continuing your migration process.

Migration plans and high-level PLP projects

A migration plan file is simply an XML file that specifies the names of one or more migration sets and, for each migration set, the list of project names and versions that make up the migration set. The Stage 1 migration tool is designed to automatically create a migration plan file for you based on the repositoryFilter preferences for project and version names. The Stage 1 tool uses these filters to determine if a project version should be reviewed to determine if the project

version is a high-level PLP project. The Stage 1 tool uses each high-level PLP project version as the basis for a migration set.

If you use PLP projects when generating your VAGen source code, then these PLP projects are the same ones you should use for migration. This is because the PLP projects provide groupings of parts that are used together during generation and therefore have all the associated parts for a set of programs.

If you do not currently use PLP projects, you can do one of the following:

- Create a high-level PLP project that specifies the list of project versions that you want to migrate as a group. Then you can use the Stage 1 migration tool to automatically create the migration plan for you.
- If you prefer not to create a high-level PLP project, you can create the migration plan file yourself using one of the following techniques:
  - If you have information in a database or other system that specifies what is needed for generation in terms of Java project versions, then you can write a tool to create the migration plan file or files automatically from your database.
  - Create the migration plan file or files manually.

Creating a high-level PLP project

Note: VisualAge Generator does not support PLPs if the project names or version names include DBCS characters. If your project or version names include DBCS characters, see "Creating a migration plan file manually" on page 103 for information on how to create the migration plan file without using a PLP.

To create a high-level PLP project for use in migration do the following in VisualAge Generator:

1. From the Workbench window, select the **Projects** tab.
2. Create a new Java project to contain the Project List Part. For example, create a project called MySubsystem1.
3. Select the new project, right-click and select **Manage -> Configure VAGen Required Projects** from the context menu.
4. In the Configure VAGen Required Projects window, select each project that you want to include in your migration set. For each project you can select a specific version to include in the migration set. Alternatively, you can select *Most recent edition*, which causes the migration tool to automatically include the version that is currently at the top of the list whenever you use this project during migration.
5. After you have selected all the project versions that you require for the migration set, click **OK**.
6. Version and release the high-level PLP project, for example MySubsystem1.
7. Test that the PLP project correctly loads the project versions you want for your migration set as follows:
   a. Delete the high-level PLP project from your workspace.
   b. Click **Selected -> Add -> Project**.
   c. From the Add Project window, do the following:
      1) Select **Add projects from the repository**.
      2) Select the high-level PLP project that you just created and the version that you created.

3) Also select **Add VAGen required projects**.
4) Click **Finish**.

d. The high-level PLP project and all the project versions it specifies are added to your workspace.
e. From the VAGen Parts Browser, select **Tools -> Show Duplicate Parts**. There should not be any parts on the list. If there are, you need to change the high-level PLP project so that there are no duplicates.
f. You might also want to run validation for your programs and tables to ensure that they are valid in VAGen and that you are not missing any parts.

You can chain PLP projects. For example, create a PLP project that lists the project versions for all your common projects. Then, for each subsystem, create a high-level PLP project for that subsystem that includes all the subsystem-specific project versions and the PLP project that specifies all the common project versions. This way you do not have list each common project version in every subsystem's high-level PLP project.

When you are ready to run the Stage 1 migration tool, do the following:

- When you set your Stage 1 preferences, on the **Build plans** page, in the Repository Filters section, set the Projects list so that a filter in the list matches the high-level PLP project you created.
- When you instruct the Stage 1 tool which preferences file to use, also specify the *-o* option. The *-o* option instructs the Stage 1 migration tool to create the migration plan files for you based on your high-level PLP projects and to overwrite any existing migration plan files.

Creating a migration plan file manually

If you already have external controls that determine what project versions to add to your workspace when you generate in VisualAge Generator, you might decide to create the migration plan file manually or to develop a tool to create the migration plan file automatically from your external information. The migration plan file must have a *.pln* file extension and the following format:

```
<migrationDefinition>
    <migrationSet name="migrationSet1" version="migrationSet1Version1"
         vgName="migrationSet1" vgVersion="migrationSet1Version1">
       <project name="projectName1" version="projectName1Version1"></project>
       <project name="projectName2" version="projectName2Version1"></project>
        .
        .
        .
       <project name="projectNameN" version="projectNameNVersion1"></project>
    </migrationSet>
    <migrationSet name="migrationSet2" version="1.1"
        vgName="migrationSet2"  vgVersion="1.1">
       <project name="projectNameA" version="projectNameAVersion1"></project>
       <project name="projectNameB" version="projectNameBVersion1"></project>
        .
        .
        .
       <project name="projectNameZ" version="projectNameZVersion1"></project>
    </migrationSet>
</migrationDefinition>
```

In the previous example, the following applies:

- migrationSet1 is a name that you can use to refer to a group of projects that must be migrated together. The migration set name is stored in the migration database and is used in the later stages of migration as follows:

- In Stage 1 migration, if maps in a map group span projects, the migration set name concatenated with a suffix is used to build the name of a new EGL project that will contain the map group and all its maps. The migration set name is also used to remove information from the migration database if you change renaming rules.
- In Stage 2 migration, the migration set name specifies which group of projects in the migration database that you want to convert to EGL.
- In Stage 3, the migration set name specifies which group of projects in the migration database you want to use to create EGL projects, packages, and files in your workspace or in a temporary directory. The migration set name and the migration set version are also used to create the high-level directory name if you choose to save the outputs of Stage 3 to a temporary directory.

The migration set name is only used during migration as a way of identifying a group of projects. Other than the situation in which maps span multiple projects in VisualAge Generator, the migration set name is not used after migration.

- projectName1, projectName2, ..., projectNameN are the projects you want to migrate as a group. You must only list a projectName once within a migration set. The migration tool loads all project versions listed under the same migration set into the workspace and processes them as a group.
- projectName1Version1, projectName2Version1, ..., projectNameNVersion1 are the respective versions of each of these projects. You can only specify one version for each project within a migration set.
- **The project names and version names you specify must exactly match the project names and version names in your repository. The names are case sensitive.** The information is used to add project versions to the workspace so that the parts can be analyzed to build the Stage 1 migration report and to load the database.

You can build a migration plan file that contains just one migration set. Alternatively, you can build a migration plan file that contains several migration sets by repeating the information between the <migrationSet> and </migrationSet> tags for each migration set.

When you are ready to run the Stage 1 migration tool, do the following:

- When you set your Stage 1 preferences, on the **Build plans** tab, set the **Plan directory name** to the drive and directory where you stored your migration plan files. Specify the **Plan file name** if you want the Stage 1 migration tool to run only **one** migration plan that you have created. Leave the **Plan file name** blank if you want the Stage 1 migration tool to run using **all** the migration plan files in the specified **Plan directory**.
- When you instruct the Stage 1 tool which preferences file to use, be sure to omit the *-o* option. Omitting the *-o* option instructs the Stage 1 tool to use the existing migration plan files. That is, the tool is not to create any new migration plan files.

Part 3. Migrating from VisualAge Generator 4.5 on Smalltalk to EGL

Chapter 5. Stage 1 — Extracting from Smalltalk

Before you can extract your information from VisualAge Generator, you must install the Stage 1 migration tool that runs on VisualAge Smalltalk. You must also create the DB2 migration database that is used to store the data you are migrating from VisualAge Generator 4.5 (VAGen 4.5) to EGL.

Installing the Stage 1 migration tool on VisualAge Smalltalk

The VisualAge Generator to EGL Stage 1 migration tool is shipped as a self-extracting file called VAGenMigST.exe To install this file, do the following:

1. Upgrade to VisualAge Generator 4.5 with Fix pack 4. Also review Appendix F, "Situations where incorrect External Source Format causes problems in creation of EGL," on page 307 for additional VisualAge Generator APARs that might be necessary for your specific situation.
2. On your system, determine where VisualAge Smalltalk is installed.
3. Run the self-extracting VAGenMigST.exe file. The file is in the following subdirectory under your Rational Developer installation directory:

   ```
   \bin
   ```

   Note: If you installed and kept a Rational Developer product before installing the product that you are using now, the installation directory of interest may be the directory that was used in the earlier install.
4. When the GUI prompt appears, navigate to the drive and directory where VisualAge Smalltalk is installed. Then click **Extract**.

When the self-extracting executable runs, it extracts the following files into your VisualAge Smalltalk installation directory:

- import\vgMigSt.dat
- feature\vgMigSt.ctl
- image\Messages.properties
- image\MigPreferences.xml
- createdatabase.sql
- createtables.sql
- SetupDatabase.bat
- SetupTables.bat
- deletemigsets.bat

Loading the migration feature

To be able to use the Stage 1 migration tool, you must load the VAGen EGL Migration feature. To do this, perform the following steps:

1. Start VisualAge Generator on Smalltalk.
2. Load the VAGen EGL Migration feature by doing the following:
   a. From the System Transcript, select **Tools -> Load/Unload Features**.
   b. On the Selection Required window, do the following:
      1) Ensure the **Show other features** checkbox is selected.

2) In the **Available features** pane, select **Other: VAGen EGL Migration - *versionName*.**
3) Select the **>>** button to move **Other: VAGen EGL Migration - *versionName*** to the **Loaded features** pane.
4) Click **OK**. The VAGen EGL Migration feature will be imported and loaded into your image.

3. In the System Transcript, you should see messages that the VAGen EGL Migration feature was loaded successfully. You should also see **EGL Migration Tools** on the tool bar. In the VisualAge Organizer, you should see HptEglMigrationGuiApp in the **Applications** pane.
4. After the VAGen EGL Migration feature is loaded, you will be prompted to save your image. Click **Yes** so you do not have to load the feature again.

Note: If you have a problem loading the feature, check your abt.ini file (contained in the *VisualAge-Smalltalk-installation-directory*\image directory). Make sure the abt.ini file has the following fields filled in under the [EmLibraryInterface] heading:

- ServerAddress=*myserver.somecompay.somewhere.com*. This value should point to the server at your company that runs EMSRV. If you use a local library, set ServerAddress=127.0.0.1.
- DefaultName=*path-to-mgr50.dat**mgr50.dat*. This value must be the name of your Smalltalk library.

Creating the migration database

See "Creating the DB2 migration database" on page 309 for information on creating the migration database. You need to use the SetupDatabase.bat and the SetupTables.bat files that were placed in the VisualAge Smalltalk installation directory when you ran the self-extracting VAGenMigST.exe file.

Setting Stage 1 preferences

When you installed the Stage 1 migration tool on VisualAge Smalltalk, the installation process created a sample preferences file called MigPreferences.xml in the directory *VisualAge-Smalltalk-installation-directory*\image. You should make a copy of the MigPreferences.xml file for backup purposes before you modify any preferences.

The VisualAge Generator to EGL migration tool on Smalltalk provides a GUI editor to assist you in specifying your Stage 1 migration preferences. You can start the Stage 1 preferences editor in either of two ways:

- From the System Transcript, select **EGL Migration Tools -> Preferences Editor**. The EGL Migration Preferences Editor appears. The preferences editor defaults to the last preferences file that you modified (or to the MigPreferences.xml file that is shipped with the Stage 1 tool if you have never modified preferences before). If you need to point to a different preferences file, click the **Open...** button.
- From the System Transcript, select **EGL Migration Tools -> Migration Driver**. In the **Migration File Preference** section, specify a file name for your preferences file and then click **Edit**. The EGL Migration Preferences Editor appears. The advantage of this technique is that after you finish modifying the preferences file, you are positioned to run the Stage 1 migration tool.

Regardless of which technique you use, the EGL Migration Preferences Editor enables you to set preferences that control the Stage 1 migration tool. When you are finished editing the preferences, click the **Save** or **Save As...** button, and then close the editor.

Note: For preferences that require a drive and directory, you can specify the information in either of two ways:

- an absolute path. For example: d:\tempMig\MySystem\
- a relative path. In this case the path is relative to the working directory. For example:
  - .\tempMig\Systems results in an absolute path of *VisualAge-Smalltalk-installation-directory*\image\tempMig\MySystem
  - ..\tempMig\MySystem results in an absolute path of *VisualAge-Smalltalk-installation-directory*\tempMig\MySystem

The preferences you can modify are described in the following sections, based on the page within the GUI in which the preference appears:

- Build Plans
- Mapping
- Renaming
- Execution

Build Plans page

The Build Plans page enables you to specify information about where the migration plan is to be placed. The Build Plans page also enables you to indicate which configuration maps and versions in the library you want to consider for migration. The Build Plans page is organized in the following sections:

- *Migration Plan Specification* information identifies where the Stage 1 migration tool is to read or write the migration plan file (or files).
  - *Plan Directory*. This is the target directory where you want your migration plan file (or files) to be placed.
  - *Plan File Name*. An optional file name of the migration plan file you are creating and using to load the migration database. You can click the **Plan File Name** button to view existing plan files in your plan directory. If you need to see details within a plan file, click the **View Plans** button and expand the plan file to see the migration sets.
    - If you do not specify a Plan file name, the migration tool deletes **all** the .pln files in the specified Plan directory before creating new plan files. The migration tool creates one plan file for each migration set. In this case, the migration Plan file names are of the form *migrationSetName_version*.pln.
    - If you specify a Plan file name, the migration tool deletes **only** the specified .pln file from the specified Plan directory before creating a new .pln file with your specified Plan file name. In this case, the single Plan file lists all the migration sets.
- *Repository Filters* information enables you to control which configuration maps and versions in your Smalltalk library are considered by the Stage 1 migration tool. Limiting the configuration maps and versions can greatly enhance the performance of the Stage 1 migration tool. You can specify multiple filters. The Stage 1 migration tool uses the Configuration Maps filter and the Version Name or Version Depth filters as follows:
  - The migration tool matches each configuration map name in the library against the Configuration Maps filter.

- If the configuration map name does not match at least one of the Configuration Maps filters, the configuration map is not considered for further processing.
- If the configuration map name matches at least one of the Configuration Map filters, the versions of the configuration map are processed as follows:
  - If you specified any Version Name filters, then each version name for the configuration map is matched against the list of Version Name filters. If the version name matches any of the Version Name filters, then the version is considered for further processing.
  - If you specified the Version Depth filter and did not specify any Version Name filters, then the most recent versions of the configuration map, up to the number specified by the Version Depth filter, are considered for further processing.

    Note: Version Depth and Version Name are mutually exclusive. By default, the Version Depth filter is included in the MigPreferences.xml file.

– If the configuration map name and version name result in the configuration map version being considered for further processing, the Stage 1 migration tool does the following:
  - If the configuration map version is a high-level configuration map, then the migration tool uses the configuration map version as the basis for creating a migration set. Each version of the high-level configuration map results in a different migration set, assuming the version name matched the version filter.
  - If the configuration map version is not a high-level configuration map, the configuration map version is not considered for further processing. The configuration map version might still be included in other migration sets; there just will not be a migration set specifically for this configuration map version.

Specify the Repository Filter information as follows:

- *Configuration Maps* filter. The migration tool matches the configuration map names in your library to the Configuration Maps filter (or filters) that you specify. You can specify multiple Configuration Maps filters. To add, change, or remove filters, right-click on a filter and use the options on the context menu. The filters are not case sensitive. You can use wildcards in the filters as follows:
  – A configuration map filter of **xyz** matches any configuration map name in the library that has the string ″xyz″ anywhere in its name.
  – A configuration map filter of *xyz** matches any configuration map name in the library that begins with ″xyz″.
  – A configuration map filter of **xyz* matches any configuration map name in the library that ends with ″xyz″.
- *Version Name* filter. If a configuration map name matches the Configuration Maps filter, the migration tool uses the Version Name filter to determine which, if any, of the configuration map versions should be considered for migration. You can specify multiple Version Name filters. To add, change or remove filters, right-click on a filter and use the options on the context menu. The filters are not case sensitive. You can use wildcards in the filters as follows:
  – A version name filter of **xyz** matches any configuration map version name that has the string ″xyz″ anywhere in the version name.
  – A version name filter of *xyz** matches any configuration map version name that begins with ″xyz″.

  - A version name filter of **xyz* matches configuration map version name that ends with ″xyz″.
  - If you leave the Version Name filters field empty, the migration tool uses the Version Depth filter.
- *Version Depth* filter. You can specify the number of previous versions you want to migrate. The default is 1, in which case the migration tool only processes the most recent version of the configuration map. If any Version Name filters are specified, the Version Depth filter is ignored.

Mapping page

The Mapping page enables you to specify the following:

- EGL file names for common parts and for unused parts.
- Suffixes that are used in building EGL project and package names.
- Options that control how your application names are converted to EGL package names.
- Information about which VAGen configuration maps and applications contain common parts.

The following describes the preferences on the Mapping page in more detail:

- *File Names*. The File Names section enables you to control the names of two EGL files that are created during migration.
  - *Common Parts* enables you to specify the name of an EGL file to contain parts that are common to multiple unique generatable parts within the scope of the migration set. Specify the file name *without* an extension or path. The migration tool creates a common parts file in any EGL package that contains parts that are used by (associated with) multiple generatable parts in the migration set or which are in VAGen configuration maps or applications that are identified as common configuration maps or applications. See “Placing parts in EGL files” on page 32 for details about whether a part is placed with in a file with a program or in the Common Parts file.
  - *Unused Parts* enables you to specify the name of an EGL file to contain parts that are not used within the scope of the migration set. Specify the file name *without* an extension or path. The migration tool creates an unused parts file in any EGL package that contains parts that are not used by (associated with) any generatable part in the migration set, provided the corresponding VAGen configuration map and application are not identified as common configuration maps or applications.
- *Spanning Maps*. The Spanning Maps section enables you to specify suffixes that are used in the event that one of your map groups includes maps from multiple configuration maps or applications.
  - *Project Suffix* enables you to specify a suffix that the Stage 1 migration tool concatenates to the migration set name to create a new EGL project name. The migration tool only creates this new EGL project if a map group and its maps are spread across multiple VAGen configuration maps within the migration set. The migration tool concatenates the suffix to the migration set name **after** any Renaming rules are applied.
  - *Package Suffix* enables you to specify a suffix that the Stage 1 migration tool concatenates to a project name to create a new EGL package name within the EGL project. The migration tool only creates this new EGL package if a map group and its maps are spread across multiple VAGen applications within a configuration map. The migration tool concatenates the suffix **after** any Renaming rules are applied to create the EGL project name.

- *EGL Package Naming Options*. The EGL Package Naming Options section enables you to specify general rules about converting Smalltalk application names to Java package names.
  - *Use package naming dot notation*. If you select this option, the migration tool converts VAGen application names to EGL package names by placing a dot before each uppercase letter in the application name after the first. For example, if you select this option, the migration tool changes MyOrderEntryApp to My.Order.Entry.App.
  - *Convert package names to lowercase*. If you select this option, the migration tool converts VAGen application names to EGL package names by changing uppercase letters to lowercase. For example, if you select this option, the migration tool changes MyOrderEntryApp to myorderentryapp.

  In general, you should select both options. If both are selected, the migration tool changes MyOrderEntryApp to my.order.entry.app. The EGL Package Naming Options are applied **after** any Renaming rules.
- *Common Identifiers*. This section enables you to specify a list of strings with wildcards that the migration tool can use in determining which configuration maps and applications contain common (shared) parts. To add or delete a common identifier, right click on the field and use the options on the context menu. When you add an identifier, the editor prompts you to enter the following information:
  - *context* indicates whether the string is to be matched to the configuration map name, application name, or both.
    - *ConfigMap* enables you to specify a string that identifies configuration maps that contain common parts. The migration tool matches this string to each configuration map name in the migration set to determine if the configuration map contains common parts. If the configuration map name matches any of the strings, all parts within the configuration map are considered to be "used". Each non-generatable part will either be placed in a program file or in the file specified by your Common Parts File Name preference; the part will not be placed in the unused parts file even if the part is not used by any generatable part in the migration set. You can enter multiple *ConfigMap* strings.
    - *Application* enables you to specify a string that identifies applications that contain common parts. The migration tool matches this string to each application name in the migration set to determine if the application contains common parts. If the string matches an application name, all parts within the application are considered to be "used". Each non-generatable part will either be placed in a program file or in the file specified by your Common Parts File Name preference; the part will not be placed in the unused parts file even if the part is not used by any generatable part in the migration set. You can enter multiple *Application* strings.
    - *Both* enables you to specify a string that the migration tool matches to both configuration map names and application names within the migration set. *Both* is equivalent to specifying the same string with a context of *ConfigMap* and a context of *Application*.
  - *Pattern to identify the common code* enables you to specify the string the migration tool should match based on the context you specified. You can use the * as a wildcard at either the beginning or end of the string. The filters are not case sensitive.

Renaming page

The Renaming page enables you to control the names of the EGL projects, packages, and versions that are derived from your VAGen configuration map, application, and version names. The number in the Order column indicates the order in which the migration tool is to apply the renaming rules, with the lowest numbered rule applied first. To add or delete a renaming rule, click on a rule and use the options on the context menu. Add Rule always puts the new rule at the end of the list. When you add a rule, the editor prompts you to enter the following information:

- *from string* specifies the characters in the VAGen name that you want to change.
- *to string* specifies the characters you want to use in the resulting EGL name.
- *string context* specifies the location in the VAGen name where the migration tool should look for the from string during renaming. The values are as follows:
  - *front* means the rule applies if the from string appears at the beginning of a configuration map, application, or version name.
  - *back* means that the rule applies if the from string appears at the end of a configuration map, application, or version name.
  - *any* means that the rule applies if the from string appears anywhere within a configuration map, application, or version name.
  - *token* means that the rule applies only if the from string is an exact match for the configuration map, application, or version name.
- *mapping context* indicates whether the migration tool is to apply the renaming rule to a configuration map, application, or version name. The values for *mapping context* are as follows:
  - *configMap* means that the renaming rule only applies to VAGen configuration map names.
  - *application* means that the renaming rule only applies to VAGen application names.
  - *both* means that the renaming rule applies to both VAGen configuration map names and VAGen application names.
  - *version* means that the renaming rule applies to the version names for all configuration maps. Use a version renaming rule if your version names include special characters such as a semicolon (:) that are not permitted in directory or file names. The migration tools use the renamed versions to create the migration plan file names in Stage 1 and to create directory names in Stage 3 of migration.

Execution page

The Execution page enables you to specify information about the location of the migration database, as well as the logging, debug, and report information you want to capture during Stage 1. The following describes the preferences you can specify on the Execution page in more detail:

- *Database* information. This section enables you to specify details about the migration database:
  - *DB* is the name of the migration database into which the migration tool is to write the migration set information. If you changed the database name from VGMIG when you created the migration database, you must change the database name specified by this preference to match the name you used.
  - *Schema* is the name used as the qualifier for the database tables. If you do not specify the schema, the migration tool uses MIGSCHEMA as the default. If you changed the schema name from MIGSCHEMA when you created the

migration database, you must change the schema name specified by this preference to match the name you used.

  - *Userid* is the user ID needed to connect to the migration database. If you not specify the Userid, the migration tool attempts to connect using the user ID specified in your VAGen SQL Preferences as the default. If the connection fails, the migration tool attempts to use your logon user ID. If both attempts fail, the migration tool displays a dialog window asking for the information.
  - *Password* is the password needed to connect to the migration database. If you not specify the password, the migration tool attempts to connect using the password specified in your VAGen SQL Preferences as the default. If the connection fails, the migration tool attempts to use your logon password. If both attempts fail, the migration tool displays a dialog window asking for the information.

    Note: The password is not encrypted in the preferences file. If this is a concern, do not enter the password in the preferences file. Wait for the prompt.

- *Service* information. This section enables you to specify details about the logging and debug information you want to capture during Stage 1. You can specify the following:
  - *Trace Level* enables you to specify the level of information that you want to write to the debug file. You can specify one of the following values:
    - *FATAL* (Level 1) -- Error messages are logged.
    - *WARN* (Level 2) -- Warning messages and error messages are logged.
    - *INFO* (Level 3) -- Informational, warning, and error messages are logged.
    - *DEBUG* (Level 4) -- Debug information, as well as informational, warning, and error messages are logged. DEBUG is the only trace level that causes the migration tool to write information to the debug file.
  - *Log File Name* enables you to specify the drive, directory, and file name for a log file. You can create the log file with any file extension, but it is best viewed as an .xml file. If you omit the log file name, a file named *migLog.xml* is written to the drive and directory that you specified in the Log File Name field. If you do not specify a drive and directory, the migration tool writes the log file to the migration plan directory.
  - *Debug File Name* enables you to specify the drive, directory, and file name for a debug file that might be needed by IBM support. You can create the debug file with any file extension, but it is best viewed as an .xml file. Information is only written to this file if the Trace Level preference is set to DEBUG. If you omit the debug file name and you specify a Trace Level of DEBUG, a file named *migDebug.xml* is written to the drive and directory that you specified in the Debug File Name field. If you do not specify a drive and directory, the migration tool writes the debug file to the migration plan directory.
- *Verification* information. This section enables you to specify information about the report file that can be output from the Stage 1 migration tool. You can specify the following:
  - *Report File Name* enables you to specify the drive, directory, and file name to be used for the report file. This report contains information about how your VAGen files are going to be migrated. You should always specify the .htm extension. If you omit the report file name, a file named *report\MigrationReport.xml* is written to the drive and directory that you specified in the Report File Name field. If you do not specify a drive and directory, the migration tool writes the report file to the migration plan directory.

Sample MigPreferences.xml file

The following is a sample MigPreferences.xml file:

```
<preferences>
    <database>
        <uri>VGMIG</uri>
        <schema>MIGSCHEMA</schema>
        <userid></userid>
        <password></password>
    </database>
    <migrationSpec>
        <directory>d:\TempMig\Stage1</directory>
        <filename></filename>
    </migrationSpec>
    <service>
        <traceLevel>4</traceLevel>
        <logfile>d:\TempMig\stage1\migLog.xml</logfile>
        <debugfile>d:\TempMig\stage1\migDebug.xml</debugfile>
    </service>
    <repositoryFilters>
        <projectName>MyConfigMap*</projectName>
        <versionNumber>1</versionNumber>
    </repositoryFilters>
    <verification>
        <reportName>d:\TempMig\report\MigrationReport.htm</reportName>
    </verification>
    <eglMapping>
        <commonPartsFileName>CommonParts</commonPartsFileName>
        <unusedPartsFileName>UnusedParts</unusedPartsFileName>
        <spanningMapsProjectSuffix>MapsProject</spanningMapsProjectSuffix>
        <spanningMapsPackageSuffix>mapspackage</spanningMapsPackageSuffix>
        <packageDotNotation>true</packageDotNotation>
        <packageLowercase>true</packageLowercase>
        <commonParts>
             <commonConfigMap>*Common*</commonConfigMap>
             <commonApplication>*Common*</commonApplication>
        </commonParts>
        <renameRule order="1">
             <fromString> </fromString>
             <toString></toString>
             <stringContext>any</stringContext>
             <mappingContext>both</mappingContext>
        </renameRule>
        <renameRule order="101">
             <fromString>CM</fromString>
             <toString></toString>
             <stringContext>back</stringContext>
             <mappingContext>configMap</mappingContext>
        </renameRule>
        <renameRule order="301">
             <fromString>App</fromString>
             <toString></toString>
             <stringContext>back</stringContext>
             <mappingContext>application</mappingContext>
        </renameRule>
        <renameRule order="501">
             <fromString>:</fromString>
             <toString>_</toString>
             <stringContext>any</stringContext>
             <mappingContext>version</mappingContext>
        </renameRule>
        <renameRule order="502">
             <fromString>/</fromString>
             <toString>_</toString>
             <stringContext>any</stringContext>
             <mappingContext>version</mappingContext>
        </renameRule>
```

```
        <renameRule order="503">
             <fromString>\</fromString>
             <toString>_</toString>
             <stringContext>any</stringContext>
             <mappingContext>version</mappingContext>
        </renameRule>
        <renameRule order="504">
             <fromString>|</fromString>
             <toString>_</toString>
             <stringContext>any</stringContext>
             <mappingContext>version</mappingContext>
        </renameRule>
        <renameRule order="505">
             <fromString>?</fromString>
             <toString>_</toString>
             <stringContext>any</stringContext>
             <mappingContext>version</mappingContext>
        </renameRule>
        <renameRule order="506">
             <fromString>*</fromString>
             <toString>_</toString>
             <stringContext>any</stringContext>
             <mappingContext>version</mappingContext>
        </renameRule>
        <renameRule order="507">
             <fromString>&lt;</fromString>
             <toString>_</toString>
             <stringContext>any</stringContext>
             <mappingContext>version</mappingContext>
        </renameRule>
        <renameRule order="508">
             <fromString>&gt;</fromString>
             <toString>_</toString>
             <stringContext>any</stringContext>
             <mappingContext>version</mappingContext>
        </renameRule>
        <renameRule order="509">
             <fromString>&quot;</fromString>
             <toString>_</toString>
             <stringContext>any</stringContext>
             <mappingContext>version</mappingContext>
        </renameRule>
        <renameRule order="510">
             <fromString> </fromString>
             <toString>_</toString>
             <stringContext>any</stringContext>
             <mappingContext>version</mappingContext>
        </renameRule>
    </eglMapping>
</preferences>
```

Deriving file names from your preferences

The Stage 1 migration tool derives the file names for the log, debug, and report file names in the same way. The following table shows a name you as you might specify it in the preferences and the resulting drive, directory and path name that the migration tool uses. In this example, the Migration Plan Directory is d:\myVAGenMig.

Table 55. File name derived from preferences

Log File Name Preference	File Name used by Stage 1 Migration Tool
Preference is left blank.	d:\myVAGenMig\migLog.xml **Note:** • The default file name for the debug file is migDebug.xml. • The default file name for the report file is \report\MigrationReport.xml
mine.xml	d:\myVAGenMig\mine.xml
logs\mine.xml	d:\myVAGenMig\logs\mine.xml
.mine.xml	*VisualAge-Generator-installation-directory*\image\mine.xml

Before you run the Stage 1 tool — hints and tips

Before you run the Stage 1 migration tool, there are some things you might want to do to improve performance. You might also want to save your existing image for use after migration is completed.

Improving performance

To minimize the memory usage, it is best to clean up (or ″scrub″) the image before running the Stage 1 migration tool. To clean up (or ″scrub″) the image, do the following:

1. From the System Transcript, select **Tools —> Open VAGen Tools Workspace**.
2. Under the ″Image Management″ section, swipe through:

   ```
   System abtScrubImage
   ```

3. Then right-click and select **Execute** to run System abtScrubImage.
4. If you scrub the image, you might need to reload the VAGen EGL Migration feature. See “Loading the migration feature” on page 107 for information on how to load the feature. Alternatively, to add the EGL Migration Tools option back onto the System Transcript tool bar, do the following:
   a. Type the following into the System Transcript window:

      ```
      HptEglMigrationGuiApp loaded
      ```

   b. Swipe through the line you typed, then right-click and select **Execute**.

To reduce the time the Stage 1 migration tool spends analyzing which configuration maps and versions to migrate, consider creating a library that only contains the configuration map versions that you want to migrate. If you have ongoing maintenance in VisualAge Generator while you are migrating, a separate migration library also has the following advantages:

- There is a stable set of configuration map versions to migrate. This is particularly important if you use the Version Depth preference to control what is to be migrated.
- You can compare the versions in the new migration library against your maintenance library to determine what additional configuration map versions still need to be migrated.

If you do create a special library, consider using it as a local library to improve Stage 1 migration performance.

Saving your image

The Stage 1 migration tool unloads all applications that contain VAGen parts from your image at the beginning of Stage 1 processing. Only the last migration set to

be processed is left in your image when the Stage 1 tool finishes. Unloading all applications ensures that only parts in the migration set are considered for the associate parts list during Stage 1. If you have an image that you wish to save, you should do the following before running the Stage 1 tool:

1. Shut down VisualAge Generator.
2. Save backup copies of the following files in your *VisualAgeForSmalltalk-installation-directory*\image:
   - abt.icx
   - abt.ini — not necessary to save if you do not change any preferences while running Stage 1
   - hpt.ini — not necessary to save if you do not change any preferences while running Stage 1
3. Start VisualAge Generator.

When you are finished running the Stage 1 tool, do the following to restore your workspace:

1. Shut down VisualAge Generator.
2. Restore the files you backed up before running the Stage 1 tool.
3. Start VisualAge Generator.

Running the Stage 1 migration tool

After you have finished editing your preferences, you are ready to run the Stage 1 migration tool to extract your source code from the Smalltalk library. To do this, perform the following steps:

1. Start the EGL Migration Driver View using one of the following techniques:
   a. If you modified the preferences by starting the Preferences Editor, start the EGL Migration Driver View from the System Transcript by selecting **EGL Migration Tools -> Migration Driver**.
   b. If you modified the preferences by starting the EGL Migration Driver, then when you saved the preferences file, you are positioned back at the EGL Migration Driver View.
2. Ensure the File Name for the Migration Preference File points to the file in which you stored your preferences. Use the **Browse...** button to point to a different preferences file. Use the **Edit...** button to review or make final modifications to your preferences.
3. When you are satisfied with your preferences, select the Execution Options that you want to use. The Execution Options control the output of the Stage 1 migration tool as follows:
   - *Overwrite PLN* controls the migration plan file or files as follows:
     – If you select *Overwrite PLN*, the Stage 1 migration tool does the following to files in the Plan Directory that you specified in your preferences:
       - If your preferences file does not specify a file name for your .pln file, the migration tool deletes all the .pln files in the specified Plan Directory and creates new files.
       - If your preferences file specifies a file name for your .pln file, the migration tool only deletes a file with the same name from the specified Plan Directory before creating a new .pln file.

– If you do not select *Overwrite PLN*, the Stage 1 migration tool does not create any new migration plan files. Instead, the Stage 1 migration tool runs based on the Plan Directory and Plan File Name you specified in your preferences:
  - If your preferences file does not specify a file name for your .pln file, the migration tool runs using all of the plans in the specified Plan Directory.
  - If your preferences file does specify a file name for your .pln file, the migration tool runs using only that .pln file.

- *Generate Report* controls whether the migration tool creates the report specified in the Verification section of the preferences file. If you do not select this option, the report is not created. If you select this option, the migration tool creates the report using the Report File Name that you specified in your preferences. The report shows how your configuration maps, applications, and VAGen parts will be assigned to EGL projects, packages, and files during migration. You might deselect this option initially so that you can review the .pln files to ensure that the migration tool is planning to process the configuration map versions that you want. If you are not satisfied with the configuration map versions that are being selected, you can refine your preferences and run Overwrite PLN again. When you are satisfied with the configuration map versions that will be processed, run Stage 1 again with the Generate Report option selected.

  Note:

  – Generating the report can take some time, therefore it is best to review the .pln files to be sure that the migration tool will process the configuration map versions that you intend.
  – If you select the Generate Report option, the Stage 1 migration tool automatically deletes any existing report files from the report directory. If you want to save previous report files, you must move the report files to a different directory or point to a new directory for your new report. Because the report files link to other files, renaming the report files will cause links to be lost and the files to become unviewable.

- *Update Database* controls whether the migration tool updates the migration database with the migration plan information. If you do not select this option, the migration database will not be updated. If you select this option, the migration database you specified in your preferences will be updated with information from the migration plan, including the External Source Format for every VAGen part in the migration plan. You might deselect this option initially so that you can review the report to see how your configuration maps, applications and VAGen parts will be placed into EGL projects, packages, and files. If you are not satisfied with the placement, you can refine your preferences and run the report again. When you are satisfied with the placement, you can run Stage 1 a final time with Update Database selected so that the migration tool will put the results into the migration database.

  Note:

  – If you select the Update Database option and a migration set already exists in the database, the Stage 1 migration tool automatically deletes the old information about migration set from

the database and then adds the new information for the migration set. There is no need for you to clean up a migration set from the database.

- The migration tool does not automatically clean up an entire migration plan.

4. After you have selected your Execution Options, click **OK** to run Stage 1 of the migration tool. The migration tool writes messages to the log file you specified in your migration preferences. The tool also puts the same messages in the System Transcript.

At this point, if you ran the Stage 1 migration tool and selected the Update Database option, your migration plan information, including your VAGen code in External Source Format, is stored in the migration database. After reviewing your report and the Stage 1 messages, you might decide to make changes to your code in VisualAge Generator and run Stage 1 again. After you are satisfied with the results of Stage 1 and have your final External Source Format code stored in the migration database, you are ready to perform Stage 2 of the migration. To run the Stage 2 migration tool, you use the Rational Developer product. See Chapter 6, "Stage 2—Conversion to EGL syntax," on page 127 for information about continuing your migration process.

Migration plans and high-level configuration maps

A migration plan file is simply an XML file that specifies the names of one or more migration sets and, for each migration set, **one** high-level configuration map and version that specifies the applications and their versions for the migration set. The high-level configuration map can also specify other configuration maps and their versions as required maps. However, only one high-level configuration map version can be specified for a migration set. The Stage 1 migration tool is designed to automatically create a migration plan file for you based on your Repository Filters preferences for configuration map and version names. The Stage 1 tool uses these filters to select the configuration map versions that should be reviewed to determine which ones are high-level configuration map versions. The Stage 1 tool uses each high-level configuration map version as the basis for a migration set.

If you use high-level configuration maps when generating your VAGen source code, then these high-level configuration maps are the same ones you should use for migration. This is because each high-level configuration map provides a grouping of parts that are used together during generation and therefore has all the associated parts for a set of programs.

If you do not currently use configuration maps at all, you must create a configuration map to use for migration. In this situation, the easiest technique is to create one configuration map version that includes all the application versions, including common application versions, that you want to migrate as a group. See "Creating a high-level configuration map" on page 121 for details. After you have created the configuration map, you can use the Stage 1 migration tool to automatically create the migration plan for you.

If you currently use configuration maps, you might not have high-level configuration maps. For example, you might have a configuration map for common applications and another configuration map for a subsystem. At generation time you determine which version of each configuration map to load into your image. In this situation, you can do one of the following to specify what you want to migrate as a group:

- Create a high-level configuration map to use during migration. This high-level configuration map can specify a list of application versions, a list of configuration map versions, or a combination of application versions and configuration map versions. For example, the high-level configuration map can list the common configuration map and the subsystem configuration map so that both configuration maps will be considered as a group when migrating. See "Creating a high-level configuration map" on page 121 for details. After you have created the high-level configuration map, you can use the Stage 1 migration tool to automatically create the migration plan for you.
- If you prefer not to create a high-level configuration map, you can create the migration plan file yourself using one of the following techniques:
  - If you have information in a database or other system that specifies what is needed for generation in terms of Smalltalk configuration map versions, then you can write a tool to create the migration plan file or files automatically from your database.
  - Create the migration plan file or files manually.

Creating a high-level configuration map

To create a high-level configuration map for use in migration do the following in VisualAge Generator:

1. From the VisualAge Organizer, select **Options** and be sure that **Full Menus** is selected.
2. From the VisualAge Organizer, select **Tools -> Configuration Maps**.
3. From the Configuration Maps Browser, select **Names -> Create**.
4. In the Information Required window, enter the name of the configuration map and then click on **OK**. A new edition of the configuration map is automatically created and selected.
5. Select **Applications -> Add**. Then select each application that you want to migrate and the version of that application. You can only specify one version for each application. Click **OK** when you have selected the version for each application that you need to include. The application versions that you specify will be migrated as a group. This group determines the set of parts that the migration tools use to resolve ambiguous situations.
6. Version the configuration map by selecting **Editions -> Version**.
7. Select the configuration map version and load it into your image by selecting **Editions -> Load**.
8. After you have loaded your new high-level configuration map, you might also want to validate your programs and tables to ensure that they are valid in VAGen and that you are not missing any parts. When you validate your programs, include /GENMAPS, /GENHELPMAPS, and /NOGENTABLES. These 2 map options enable you to ensure that the maps are valid for the programs in which they are used. /NOGENTABLES enables you to validate a table just one time rather than revalidating the table with every program in which the table is used.

Chaining configuration maps

You can chain configuration maps. For example, you can create a configuration map that lists the version for each of your common applications. Then, for each subsystem, create a high-level configuration map for that subsystem that includes the version you need of each subsystem-specific application. You can include the configuration map for the common applications in the subsystem configuration map as follows:

1. From the Configuration Maps Browser, select an open edition of the subsystem configuration map.
2. Select **Expressions -> Add**.
3. In the Information Required window, click **OK** to accept *true* as the expression.
4. Select *true* in the Config. Expressions pane. Then select **Required Maps -> Add -> As First**. Then select the configuration map version that contains the common applications. Version the configuration map by selecting **Editions -> Version**.
5. Select the configuration map version and load it into your image by selecting **Editions -> Load With Required Maps**.

Using required maps provides a simple way of specifying the common application versions without having to explicitly list the common application versions in every subsystem's high-level configuration map.

Using configuration maps with the Stage 1 tool

When you are ready to run the Stage 1 migration tool, do the following:

- When you set your Stage 1 preferences, on the Build Plans page, in the Repository Filters section, set the Configuration Maps list so that a filter in the list matches the high-level configuration map you created.
- When you instruct the Stage 1 tool which preferences file to use, also select the *Overwrite PLN* option. This option instructs the Stage 1 migration tool to create the migration plan files for you based on your high-level configuration maps and to overwrite any existing migration plan files.

Creating a migration plan file manually

If you already have external controls that determine what configuration map versions to load into your image when you generate in VisualAge Generator, you might decide to create the migration plan file by hand or to develop a tool to create the migration plan file automatically from your external information. The migration plan file must have a *.pln* file extension and the following format:

```
<migrationDefinition>
    <migrationSet
       name="migrationSet1"
       version="migrationSet1Version1"
       vgName="migrationSet1"
       vgVersion="migrationSet1Version1">
       <configMap
          name="configurationMap1"
          version="configurationMap1Version1">
       </configMap>
       <configMap
          name="configurationMap2"
          version="configurationMap2Version1">
       </configMap>
           .
           .
           .
       <configMap
          name="configurationMapN"
          version="configurationMapNVersion1">
       </configMap>
     </migrationSet>
</migrationDefinition>
```

In the previous example, the following apply:

- migrationSet1 is a name that you can use to refer to a group of configuration maps that must be migrated together. The migration set name is stored in the migration database and is used in the later stages of migration as follows:
  - In Stage 1 migration, if maps in a map group span configuration maps, the migration set name concatenated with a suffix is used to build the name of a new EGL project that will contain the map group and all its maps.
  - In Stage 2 migration, the migration set name specifies which group of configuration maps in the migration database that you want to convert to EGL.
  - In Stage 3, the migration set name specifies which group of configuration maps in the migration database you want to use to create EGL projects, packages, and files in your workspace or in a temporary directory. The migration set name and the migration set version are also used to create the high-level directory name if you choose to save the outputs of Stage 3 to a temporary directory.

  The migration set name is only used during migration as a way of identifying a group of configuration maps. Other than the situation in which maps span multiple configuration maps in VisualAge Generator, the migration set name is not used after migration.
- configurationMap1, configurationMap2, ... configurationMapN are the configuration maps you want to migrate as a group. You must only list a configurationMap once within a migration set.
- configurationMap1Version1, configurationMap2Version1, ... configurationMapNVersion1 are the respective versions of each of these configuration maps. You can only specify one version for each configuration map within a migration set.
- **The configuration map names and version names you specify must exactly match the configuration map names and version names in your library. The names are case sensitive.** The information is used to add configuration map versions to the image so that the parts can be analyzed to build the Stage 1 migration report and to load the migration database.

When you are ready to run the Stage 1 migration tool, do the following:

- When you set your Stage 1 preferences, on the **Build plans** tab, set the **Plan directory name** to the drive and directory where you stored your migration plan files. Specify the **Plan file name** if you want the Stage 1 migration tool to run only **one** migration plan that you have created. Leave the **Plan file name** blank if you want the Stage 1 migration tool to run using **all** the migration plan files in the specified **Plan directory**.
- When you instruct the Stage 1 tool which outputs to produce, be sure to deselect *Overwrite PLN*. This causes the migration tool to run using the previously created .pln file based on your **Plan file name** preference.

Part 4. Stages 2 and 3— common to Java and Smalltalk migration

The remaining steps of the migration are the same whether you are coming from VisualAge Generator on Java or VisualAge Generator on Smalltalk.

Chapter 6. Stage 2—Conversion to EGL syntax

Stage 2 of migration is the same whether you are migrating from Java or Smalltalk.

You must run another migration tool to convert your source from External Source Format syntax to EGL syntax. This migration tool is a plug-in that is available after you install the Rational Developer product. You can run the tool in batch mode or interactive mode. You can optionally specify that Stage 3 is to run automatically after Stage 2 completes.

Setting your workbench preferences

Before you start to migrate you should set your workbench preferences. This includes the following:

- Start up parameters
- Required EGL preferences
- Recommended preferences
- VAGen Migration Syntax Preferences
- Other recommended settings

Start up parameters

To improve the performance of the Rational Developer product, you should set several start up parameters in the initialization file. The parameters are the same regardless of the Rational Developer product that you use. The initialization file is always in the Rational Developer product installation directory, but the name of the file varies with the product that you are using. For example, if you are using Rational Application Developer, the initialization file is named rationalsdp.ini. To set the start up parameters, do the following:

1. Using a text editor, edit the initialization file.
2. Look for the following line:

   ```
   VMArgs=-Xj9
   ```

3. Change the line to the following:

   ```
   VMArgs=-Xj9 -Xmx1024m -Xms256m
   ```

   Setting these additional values increases the available virtual memory and stack space.
4. Save the initialization file.
5. Start the Rational Developer product.

Required EGL preferences

Before you can you use the migration tool, you must enable both the EGL and the migration tool capabilities. To enable these capabilities, do the following:

- From the Workbench window, select **Window** -> **Preferences**.
- Expand the **Workbench** preference by selecting the + beside it and then select **Capabilities**.
- Select **EGL** to automatically select both **EGL Development** and **VisualAge Generator to EGL Migration**.
- Select **OK**.

You must also set the VisualAge Generator compatibility mode before you migrate. Setting the VisualAge Generator compatibility mode avoids numerous messages in the Problems view. To set this preference, do the following:

- From the Workbench window, select **Window** -> **Preferences**.
- Select **EGL**.
- Select the *VisualAge Generator Compatibility* preference.
- Select **OK**.
- When you are prompted for a full-rebuild of the workspace, select **OK**.

Recommended preferences

The following preferences are recommended to assist you in resolving any EGL validation messages in the Problems view. From the Workbench window, select **Window -> Preferences** and then the preference page indicated below as follows:

- **EGL -> Editor**. Select **Show line numbers**.
- **Workbench -> Editors -> Text Editor**. Select **Show line numbers**.
- Workbench page. Decide whether to select or deselect **Build automatically**. If you select this option, whenever you save a file, EGL rebuilds everything in the workspace and runs validation. The advantage of selecting the option is that you get immediate feedback on the changes you have made. The disadvantage is that rebuilding can take some time depending on the number of parts in your workspace. If you do not select this option, EGL does not rebuild anything when you save a file. The advantage of deselecting the option is that you avoid multiple rebuilds when you are modifying a number of files. However, you must remember to rebuild the projects (**Project -> Build Project** or **Project -> Build All**) to see the results of any changes on the messages in the Problems view. You might want to deselect **Build automatically** while you are working through the list of messages in the migration log. This enables you to control when the rebuild occurs. When you are doing normal code development, then you might want to select this option. The Stage 3 Migration Tool always turns off **Build automatically**.
- **Workbench -> Local History**. The .egl files that are produced by the migration tool can be quite large. Therefore, you should change the **Maximum file size (MB)** to a larger value (for example, 5). In addition, you might want to change the **Days to keep files** and the **Entries per file** based on your backup requirements.
- **Workbench -> Perspectives**. You might want to set either the EGL perspective or the Web perspective as your default perspective. Use the EGL perspective if you will not be developing web applications. Use the Web perspective if you plan to develop web applications. To set the default perspective, select the perspective you want to use and then select **Make Default**.
- **Logging**. Select the **General** tab and then change the **Default logging level** to **SEVERE**. Optionally, change the **Default logging level** to **INFO** or **WARNING**, but do not leave it set to **NONE**.

VAGen Migration Syntax Preferences

The preferences listed below control the migration of the VAGen syntax to the EGL syntax. Unless otherwise noted, these preferences are used in Stage 2 for both a Stage 1 – 3 migration and for single file migration. To set these preferences, from the Workbench window, select **Window -> Preferences -> VAGen Migration -> VAGen Migration Syntax Preferences.**

- Renaming preferences

- *Renaming prefix*. The migration tool uses this prefix whenever a VAGen data item, record, function or map name is an EGL or SQL reserved word or starts with the # symbol. The tool adds the Renaming prefix to the VAGen part name to create a valid EGL part name. For example, *date* is an EGL reserved word. If you have a function named DATE and use the default Renaming prefix of VAGen_, then the migration tool changes the function DATE to VAGen_DATE. The tool also changes all references to the function from DATE to VAGen_DATE. You can set the Renaming prefix to any value that you want other than blank or EZE. In addition, the Renaming prefix cannot start with the # symbol. Be sure to select something that will not cause conflicts with any of your part names.
- *Level77 suffix*. The migration tool uses this suffix whenever a VAGen working storage record contains level 77 items. EGL does not support level 77 items. The migration tool splits the VAGen working storage record into two EGL records. The first record is named the same as the original VAGen working storage record and contains all the non-level 77 items. The second record contains all the level 77 items. The migration tool names this second record based on the original working storage record name concatenated with the Level77 suffix. For example, if the original working storage record is named MYRECORD and you use the default Level77 suffix of _Level77Items, the EGL record that contains the level 77 items will be named MYRECORD_Level77Items. You can set the Level77 suffix to any value that you want other than blank. However, be sure to select something that will not cause conflicts with any of your part names.
- *Help Map suffix*. The migration tool uses this suffix whenever the main map group and the help map group for a program have maps with the same name. EGL requires that all forms in both form groups for a program have unique names. The migration tool renames maps in the help group that conflict with map names in the main map group. Consider the following example. The main map group for a program is MAPGP1 and contains MAP1. The help map group for the same program is MAPGP2 and contains MAP1. Using the default Help Map suffix of _helpmap, the migration tool renames the MAP1 in MAPGP2 to be MAP1_helpmap. You can set the Help Map suffix to any value that you want other than blank. However, be sure to select something that will not cause conflicts with any of your part names.

- SQL preferences
  - *Result Set suffix*. In VisualAge Generator, when an SQL REPLACE function needs to reference the corresponding UPDATE or SETUPD function, the REPLACE function must specify the function name. In EGL, multiple I/Os are supported within a single function. A result set ID is used to uniquely identify a get or open statement. The migration tool creates the result set ID from the function name by concatenating the Result Set suffix. For example, if a function is named MY-SETUPD and you use the default Result Set suffix of _RSI01, then the result set ID that is included in the open statement for the function will be MY-SETUPD_RSI01. You can set the Result Set suffix to any value that you want other than blank. However, be sure to select something that will not cause conflicts with any of your part names.
  - *Prepare suffix*. In VisualAge Generator, if you need to have an SQL I/O statement prepared at runtime, you select Execution Time Statement Build. The corresponding EGL statement is the *prepare* statement. The prepare statement includes a prepare statement ID so that other I/O statements such as close, get, execute, and open can specify which prepare statement they are associated with. The migration tool creates the prepare statement ID from the function name by concatenating the Prepare suffix. For example, if there is a function named MY-EXEC-TIME-BUILD and you use the default Prepare

suffix of _PREP01, then the prepare statement ID that is included in the prepare statement is MY-EXEC-TIME-BUILD_PREP01. You can set the Prepare suffix to any value that you want other than blank. However, be sure to select something that will not cause conflicts with any of your part names.

- *Omit column name*. In VisualAge Generator, the SQL column name is always specified for an item in an SQL record. In EGL, you can omit the SQL column name if it is the same as the item name. If you select Omit column name, the migration tool omits the EGL column property whenever the SQL column name is the same as the item name. Omitting the column name can make your EGL source code less cluttered.
- *Omit isNullable property*. In VisualAge Generator, the null indicator variable is always included for every item in the SQL record. To preserve the VisualAge Generator behavior, the migration tool always includes the isNullable property for every item in an SQL record. However, in EGL, you can omit the null indicator variable if the column is defined in SQL as not null. If you select Omit isNullable property, the migration tool omits the isNullable property from every item in SQL records. Omitting the isNullable property can improve runtime performance. However, you should only select Omit isNullable property if you are certain that all of your SQL columns are defined as not null. If any of your SQL columns can be null, you should not select Omit isNullable property during migration. After migration, you can remove the isNullable property for selected items in your SQL records if you want to improve performance.
- *Omit isReadOnly property*. In VisualAge Generator, the Read Only property must be explicitly set for each item in an SQL record. Read Only must always be no if there are multiple tables specified for the SQL record. By default, the migration tool includes the isReadOnly property for every item in an SQL record. However, the EGL isReadOnly property defaults to yes if there is only one SQL table and to no if there are multiple SQL tables specified for the SQL record. If you select Omit isReadOnly property, the migration tool only includes isReadOnly if there is a single table specified for the SQL record and the VisualAge Generator Read Only property is set to yes. Omitting the isReadOnly property can make your EGL source code less cluttered.

- Data Items preferences
  - *Convert shared data items to primitive item definition.* If you select this preference, then whenever shared data items are used in records, tables, function local storage, function parameter lists, or program parameter lists, the migration tool converts the shared items to primitive item definitions. If you have current organization standards that discourage the use of shared data items in new applications, this option enables you to remove the use of shared data items from existing applications as you migrate.
  - *Include shared data items in EGL file.* This option is preselected. The shared data items are always included in the EGL file.

Other recommended settings

The following additional settings are recommended:

- Optionally, close the Welcome view.
- If you are not already using the EGL perspective, switch to it by selecting **Window** -> **Open Perspective** -> **Other** -> **Resource** -> **OK**. Alternatively, if you plan to develop new EGL page handlers, you should switch to the Web perspective. You can close other perspectives by right-clicking the icon for the perspective on the tab in the upper-right corner of the window and then selecting **Close**.

- If the Navigator view is not already included with the perspective that you are using, you might want to add this view. To add the Navigator view, select **Window -> Show view -> Other**. From the Show View window, expand **Basic** and then select **Navigator**.
- The Problems view shows any validation errors. Therefore, you might want to set the Problems view to be in the foreground. In the lower right section of the Workbench window, select the Problems tab to place the Problems view in the foreground.
- In the Problems view, select the **Filters...** icon in the upper right corner. In the **Filters** window, do as follows:
  - Select the **On selected resource only** radio button. This limits the error messages in the Problems view to the messages for the currently selected file. When there are errors, this can help you focus your attention on a single file at a time. The title bar of the Problems view provides a count of the messages that matched the filter and the total number of messages for all projects in your workspace.
  - Deselect **Limit visible items to**. This enables you to see all the messages.
- When you have multiple files open for editing, you can configure the Navigator view to automatically bring an open file to the foreground (make its editor session the active editor) every time you select that open file in the Navigator view. To do this, select the **Link with Editor** icon on the tool bar of the Navigator view. You might also want to select the **Link with Editor** icon on the tool bar of the Project Explorer view.

Setting up the Stage 2 VAGen migration file

The tool that performs Stage 2 of the migration can be invoked through a wizard. To prepare for Stage 2, create a project where you can optionally save Stage 2 preferences for later use. To create the Stage 2 migration preferences, do the following:

1. Start the Rational Developer product. Be sure to set your workbench preferences as explained in section "Setting your workbench preferences" on page 127.
2. The Stage 2 wizard asks you for your database driver location. You can set a classpath variable to hold this value so that the wizard will pick it up automatically. The easiest way to do this is as follows:
   a. Under **Window->Preferences**, select **Java->Build Path->Classpath Variables**.
   b. Click the **New** button.
   c. For **Name**, enter the following: **DB2_DRIVER_PATH**
   d. For **Path**, click the **File** button and navigate to your db2java.zip file. (This is the same db2java.zip file that you used in Stage 1. By default the file is in \SQLLIB\java\db2java.zip.)
   e. Click **OK** in the New Variable Entry window, then click **OK** in the Preferences window.
3. Create a simple project that can contain your Stage 2 preferences file if you choose to save it. This is useful if you want to run Stage 2 in batch mode. Start the wizard for this by selecting **File->New->Project->Simple->Project->Next**.
4. Give the project a name. For example, VAGENMIG. Then click **Finish**.
5. In the Project Explorer view, a simple project is listed under the Other Projects folder. In the Navigator view, a simple project is listed alphabetically with all the other projects.

6. Right click on your project and then select **New->Other** from the context menu.
7. Expand **VAGen Migration to EGL** and then select **VAGen Migration File**. Click **Next**.
8. Enter the appropriate Stage 2 preferences:
   a. On the first page of the wizard, edit the preferences as described in the following table. The migration tool does not validate any of the Database Information fields until you tab out of the field. This prevents multiple attempts to connect to the database while you are entering information.

Table 56. Preferences to enter on first page of wizard

Preference	Meaning	Value
Load Existing File	This allows you to select a previously saved Stage 2 preferences file. Click the Choose File button to select an existing .vgmig file. Click the Load File button to retrieve the preferences from that file and display them in the wizard.	Optionally, choose and load an existing .vgmig file.
Database driver location	This is the location of your DB2 driver.	*path_to_db2java.zip*\db2java.zip
Database driver	This is the name of your DB2 driver.	This value must always be COM.ibm.db2.jdbc.**app**.DB2Driver. This value works for both a local database or a remote database that is cataloged locally.
Database name	This is the name of the DB2 database you used in Stage 1 of migration.	This value should be in the following format: • jdbc:DB2:*databaseName* *databaseName* is the name of the DB2 database you used in Stage 1 of migration. VGMIG is the default value for Stage 1.
Database schema	This is the name of the DB2 database schema you used in Stage 1 of migration.	This value is the name of the DB2 schema you used in Stage 1 of migration. MIGSCHEMA is the default value for Stage 1.
Database user ID	This is the database user ID you used in Stage 1 of migration.	Use the same database user ID that you used for Stage 1. (The default value is your logon ID.)
Database password	This is the database password you used in Stage 1 of migration.	Use the same database password that you used for Stage 1. (The default value is your logon password.)
Log file location	This is the location where a log file will be written.	Enter a valid location (drive and directory) in the file system.
Log file name	This is the name of the log file where you want the Stage 2 messages to be written.	Enter a valid file name.

   b. On the second page of the wizard, edit the preferences as described in the following table:

Table 57. Preferences to enter on second page of wizard

Preference	Meaning	Value
Java or COBOL radio button	This choice determines whether the migration tool creates projects that include Java Source folders.	If you are using Rational Web Developer or Rational Application Developer, you should always select the Java radio button. For other Rational Developer products, you can select COBOL if you only plan to generate COBOL.

Table 57. Preferences to enter on second page of wizard (continued)

Preference	Meaning	Value
Migrate remaining VAGen parts	This determines whether or not parts not referenced by any generatable part in the migration set will be converted to EGL.	Select the box to convert unreferenced parts to EGL. Generally, you should select Migrate remaining parts. If you do not select Migrate remaining parts, control parts and any other parts that are unused within the migration set will not be migrated to EGL source.
Import into workspace	This determines whether or not Stage 3 (importing EGL into files in the current workspace) will be automatically started after Stage 2 is complete. **Note:** If you select this box, you must select one of the radio buttons under this checkbox to specify the version to want to import (latest or oldest – see description below), because `only one version of a project can be in the workspace at a time.	Select this box to import EGL files directly after the conversion of parts to EGL. Leave this box unselected to import files later, during Stage 3. **Note:** • If you select this option, there is no need to run Stage 3 separately. The migration tool automatically starts Stage 3 (import) directly after Stage 2 (conversion) and completes the migration process.
Latest version	This specifies that the latest version of the desired migration sets should be imported.	This option can only be selected if the *Import into workspace* checkbox is selected.
Oldest version	This specifies that the oldest version of the desired migration sets should be imported.	This option can only be selected if the *Import into workspace* checkbox is selected.
Override existing files	Stage 3 (the import process) uses EGL produced by Stage 2 to create and import the EGL files specified in the Stage 1 report. If EGL files with the same names as the EGL files that Stage 3 is about to import already exist in the workspace, this option determines whether or not those files will be overwritten.	This option can only be selected if the *Import into workspace* checkbox is selected. *Override existing files* enables you to specify how you want the Stage 3 migration tool to handle the situation in which the migration set you are currently migrating contains parts that should be placed in a file that is already in your workspace. If you select *Override existing files*, the Stage 3 migration tool replaces the existing file and includes **only** those parts that are in the **current** migration set. If you do not select *Override existing files*, the Stage 3 migration tool merges any new parts into the existing file. The new parts are placed alphabetically by part type. See "Overwriting and merging files" on page 36 for a more complete discussion of the effects of this option.

Table 57. Preferences to enter on second page of wizard (continued)

Preference	Meaning	Value
Save migrated files to temporary directory	This provides the option to save EGL files to a location in the file system. This allows you to access EGL files for multiple versions of a project at the same time (whereas you can only see one version at a time in the workspace). You can move EGL files directly from here to your repository.	If you plan to migrate multiple versions of a migration set, then do the following: • Select this box so that each version can be written to a different subdirectory. • Specify the *Folder* under which the subdirectories for the versions will be placed. • Do not select *Migrate now*. Migration to a temporary directory should only be done in batch mode because of the resource requirements. If you do select *Migrate now*, the migration tool asks you to confirm that you really want to run in online mode. • Select *Save current configuration to file*. You must also specify the project and file name where the current configuration is to be saved as a .vgmig file. The .vgmig file is required if you select *Save migrated files to temporary directory* regardless of whether you migrate in online or batch mode. If you run Stage 2 in batch mode, point to the saved .vgmig file to specify your migration preferences.
Folder	This is the directory in which you want to save EGL files. Each migration set version becomes a subdirectory under the directory you specify for Folder.	Specify an existing directory in your file system.
Migrate now	This specifies that you want Stage 2 to run at this time, rather than just setting up the preferences file to migrate at a later time.	You must select either *Migrate now* to run Stage 2 in online mode or *Save current configuration to file* to save your preferences so that you can run Stage 2 in batch mode at a later time. You can select both options if you want to retain a copy of your preferences for later reference. Do not select *Migrate now* if you have already selected *Save migrated files to temporary directory*. Saving to a temporary directory can only be done in batch mode. Selecting *Migrate now* indicates that you want to migrate in online mode.
Save current configuration to file	This enables you to save the preferences you are specifying to a file. You can later run Stage 2 in one of the following ways: • In online mode, right click the saved .vgmig file and select **Start Migration** from the context menu. • In batch mode, use the -importFile option to specify the saved .vgmig file. For details, see "Running Stage 2 in batch mode" on page 136.	You must select either *Migrate now* to run Stage 2 in online mode or *Save current configuration to file* to save your preferences so that you can run Stage 2 at a later time. You can select both options if you want to retain a copy of your preferences for later reference. If you select this option you must also specify the Path and File Name where the current configuration is to be saved as a .vgmig file. When you run Stage 2 later, point to the saved .vgmig file to specify your migration preferences.
Path	This specifies the project into which the file should be saved.	*\projectName*, where projectName is the name of the project that you want to contain your saved file.
File Name	This specifies the name of the file to which preferences will be saved.	*fileName*, where fileName is the desired name for your file WITHOUT a file extension. The extension *.vgmig* is automatically appended.

c. On the third page of the wizard, the wizard lists the migration sets in the database you specified. Select the migration sets you want to migrate. If you do not select any migration sets, then the migration tool migrates all the migration sets in the database. Click **Finish**.

The combinations of the check boxes that you select determine the actions that are performed by the wizard:

- If you select *Save current configuration to file*, all the options are saved in the file you specified after you click the Finish button.
- If you select *Migrate now*, Stage 2 migration runs after you click the Finish button.
- If you also select either *Import into workspace* and/or *Save migrated files to temporary directory*, Stage 3 will start automatically after Stage 2 completes.

Here is an example of a Stage 2 preferences file, stage2.vgmig:

```
databaseDriverLocation=d:\SQLLIB\java\java\db2java.zip
databaseDriver=COM.ibm.db2.jdbc.app.DB2Driver
databaseName=jdbc:DB2:VGMIG
databaseSchema=MIGSCHEMA
databaseUserid=myuserID
databasePassword=encoded:AAEDAwQFBwYKCwo+Pw==
configurationPlan=MyMigrationSetA,1.1
migrateRemainingParts=yes
workspaceImport=latest
overrideExistingFiles=yes
tempDirectory=
logFileLocation=D:\tempmig\MyMigrationSetA\stage2\MyMigrationSetA.log
migrateNow=yes
projectType=COBOL
```

Running Stage 2

The Stage 2 migration tool can be run in batch mode or from the user interface in the Rational Developer product.

Running Stage 2 from the user interface

The wizard described in "Setting up the Stage 2 VAGen migration file" on page 131 provides the option to save your preferences in a .vgmig file. If you select the *Migrate now* box in the wizard, then Stage 2 migration starts when you click the Finish button in the wizard. If you did not select the *Migrate now* box in the wizard, when you are ready to run Stage 2 migration using your saved .vgmig file, do the following:

1. From the Navigator view, expand the project containing the Stage 2 preferences file by right-clicking on the + symbol next to the project name.
2. Right click on the .vgmig preferences file to get the context menu.
3. Click on **Start Migration**.

Stage 2 migration starts and converts the External Source Format for your specified migration sets to EGL source and stores the EGL source in the migration database. If you selected either *Import into workspace* or *Save migrated files to temporary directory*, Stage 3 automatically starts after Stage 2 completes. After Stage 3, the migration tool automatically starts a refresh of the workspace. The refresh step can take some time, particularly if there is a large number of parts. When the refresh step is complete, a pop-up window appears telling you that migration is complete. Be sure to wait for the pop-up window.

When migration and the refresh step are complete, the following outputs are available:

- The Stage 2 migration log file. The log file is in the directory you specified as the Log file location. The log file contains information about what parts were migrated and any informational, warning, or error messages that occurred during Stage 2 migration.
- The ″to do″ list log file for the migration set. This ″to do″ list file is created at the beginning of Stage 3 and contains a consolidated list of the messages produced by Stage 2 that might require you to perform additional tasks to complete the migration. The ″to do″ list is somewhat similar to the VisualAge Generator generation messages in that the messages for each generatable part and its associates are listed as a group. If a part has messages and is an associate of several programs, the messages are listed once for each program. The ″to do″ list differs from the VisualAge Generator generation messages in that messages for unused, nongeneratable parts are listed by project, package, and file name at the end of the ″to do″ list. The ″to do″ list is placed in the same directory as the Stage 2 migration log file.
- If you selected *Import into workspace*, then Stage 3 automatically starts and creates the EGL projects, source folders, packages, and EGL files that are needed for your migration set. The Stage 3 tool also imports the projects into your workspace and rebuilds the projects so that EGL validation is run.
- If you selected *Save migrated files to temporary directory*, Stage 3 automatically starts and creates the EGL projects, source folders, packages, and EGL files that are needed for your migration set. The Stage 3 tool places these projects in the temporary directory you specified. The Stage 3 migration tool appends the VAGen version name to the project name when creating the projects in the temporary directory. This enables you to migrate multiple versions of a project at one time for later import into your workspace.

Running Stage 2 in batch mode

The Stage 2 wizard enables you to select one or more migration sets for immediate migration. It also enables you to save the information in a file for later migration in batch mode. To create a file for later migration, do the following:

1. Follow the steps described in “Setting up the Stage 2 VAGen migration file” on page 131, except do the following:
   - Select *Save current configuration to file* and specify the path and file name. The file that is created automatically has the suffix *.vgmig* and is the file that you need to specify as the *-importFile* when you run Stage 2 or Stage 3 in batch mode.
   - Be sure to deselect *Migrate now*. Deselecting this option indicates that you want to save the information for migration at a later time.
   - You can define multiple .vgmig files for later migration as a single batch.
2. Create a file with a .bat file extension.

   For Windows environments, the contents of the .bat file should be the following:

```
set path=InstallDirectory\eclipse\jre\bin;%path%
set classpath=InstallDirectory\eclipse\startup.jar;
  InstallDirectory\egl\eclipse\plugins\
  com.ibm.etools.egl.vagenmigration_version\runtime\eglMigration.jar;
cd InstallDirectory
java com.ibm.etools.egl.internal.vagenmigration.batch.VGMIG
     -importFile Path\vgmigFileName.vgmig
     -data Path\workspace
     >Path\LogName.log
```

For Linux environments, the contents of the .bat file should be the following:

```
# !/bin/bash
export PATH=/InstallDirectory/eclipse/jre/bin/:$PATH
export CLASSPATH=/InstallDirectory/eclipse/startup.jar:
  /InstallDirectory/egl/eclipse/plugins/
  com.ibm.etools.egl.vagenmigration_version/runtime/eglMigration.jar:$CLASSPATH
cd /InstallDirectory/
java com.ibm.etools.egl.internal.vagenmigration.batch.VGMIG
  -importFile /Path/vgmigFileName.vgmig
  -data /Path/workspace
  >/Path/LogName.log
```

In addition, for Linux environments, you must change the permissions for the .bat file so that it is executable. To make the .bat file executable, bring up a Command Terminal and enter the following command:

```
chmod u+x mybatchfile.bat
```

mybatchfile.bat
Name of the .bat file you created.

Note:

- The following statements must be written so that the statement is all on one line:
  - For Windows environments, the **set classpath** statement.
  - For Linux environments, the **export classpath** statement.
  - The **java** statement.
- Repeat the **java** statement once for each .vgmig file that you want to migrate in batch mode. However, if you selected *Import into workspace* when you created any of your .vgmig files, then be sure that none of the .vgmig files result in the same EGL project names. **If you attempt to migrate multiple .vgmig files for the same EGL project in the same .bat file, the EGL project will only reflect the last of the .vgmig files to be migrated.**
- *InstallDirectory* is the drive and directory in which you installed the Rational Developer product. You must include the *InstallDirectory* for the path statement, the classpath statement, and the cd (change directory) statement.

  Note: If you installed and kept a Rational Developer product before installing the product that you are using now, the installation directory of interest may be the directory that was used in the earlier install.
- *version* is the plugin version number (for example, 6.0 or 6.0.0.1). In general, you should use the highest version number you see for the com.ibm.etools.egl.vagenmigration_*version* plugin.
- *Path\vgmigFileName.vgmig* refers to the drive, directory, and file name of the .vgmig file that specifies the migration sets you want to migrate from the migration database. The directory must include the workspace name. This is the .vgmig file you saved in step 1. (For example, d:\myworkspace\mySimpleProject\myMigrationInformation.vgmig.)
- *Path\workspace* is the drive, directory, and workspace name where you want to place the EGL files. (For example, the workspace could be d:\myworkspace.) Any EGL projects and packages that are used by the migration sets are created automatically by the migration tool. The *-data* parameter is optional; -data is only required if you need to

specify VAGen Migration Syntax Preferences other than the default values. If you want to set any VAGen Migration Syntax Preferences, you must specify them in the workspace specified by the -data parameter before you run the .bat file. See "VAGen Migration Syntax Preferences" on page 128 for information about how to set your preferences.

- *Path\LogName.log* points to the drive, directory, and file name of the log file you want to create for the java command. This log file lists any problems with the java command itself. Any log messages produced by Stage 2 or Stage 3 are placed in the log file that you specified on the first page of the wizard and then saved into the .vgmig file. If you include multiple java commands in the same .bat file, be sure to specify a different log file name for each java command.

For Windows environments, an example of the java command might look something like this (though it should be all on one line) :

```
java com.ibm.etools.egl.internal.vagenmigration.batch.VGMIG
  -importFile d:\myworkspace\mySimpleProject\myMigrationInformation.vgmig
  -data d:\myworkspace\
  >d:\migrationLogs\myMigrationInformationPiped.log
```

For Linux environments, an example of the java command might look something like this (though it should be all on one line):

```
java com.ibm.etools.egl.internal.vagenmigration.batch.VGMIG
  -importFile /d:/myworkspace/mySimpleProject/myMigrationInformation.vgmig
  -data /d:/myworkspace
  >/d:/migrationLogs/myMigrationInformationPiped.log
```

3. Shut down the Rational Developer product.
4. Open a Command Prompt window, navigate to the directory containing your .bat file, and run your .bat file.
5. When the process completes, your EGL project, source folders, packages, and files will be stored in the directories you specified for them respectively. The log file corresponding to each java command contains a list of the migrated parts and any error messages. The messages are the same messages that are written to the log file if you run Stage 2 using the user interface. Similarly, the ″to do″ list file contains the same messages that are written to this file if you run Stage 2 using the user interface.
6. Start the Rational Developer product.
7. You should see the EGL projects, source folders, packages, and files in your workspace.

Chapter 7. Stage 3 — Import

Stage 3 of the migration is also run with a plug-in supplied with the Rational Developer product. In this stage, you run another migration tool that builds EGL files from the EGL syntax that was stored in the migration database during Stage 2.

Running the Stage 3 tool

From the Rational Developer product Workbench, do the following:

1. From the File menu, select **Import**.
2. Select **VAGen Migration from Database** and click **Next**.
3. Specify your preferences for this stage of migration:
   a. On the first page of the wizard, edit the preferences as described in the following table. The migration tool does not validate any of the Database Information fields until you tab out of the field. This prevents multiple attempts to connect to the database while you are entering information.

Table 58. Preferences to enter on first page of wizard

Preference	Meaning	Value
Load Existing File	This allows you to select a previously saved Stage 3 preferences file. Click the Choose File button to select an existing .vgmig file. Click the Load File button to retrieve the preferences from that file and display them in the wizard.	Optionally, choose and load an existing .vgmig file.
Database driver location	This is the location of your DB2 driver.	*path_to_db2java.zip*\db2java.zip
Database driver	This is the name of your DB2 driver.	This value must always be COM.ibm.db2.jdbc.**app**.DB2Driver. This value works for both a local database or a remote database that is cataloged locally.
Database name	This is the name of the DB2 database you used in Stage 1 of migration.	This value should be in the following format: • jdbc:DB2:*databaseName* *databaseName* is the name of the DB2 database you used in Stage 1. VGMIG is the default value for Stage 1.
Database schema	This is the name of the DB2 database schema you used in Stage 1 of migration.	This value is the name of the DB2 schema you used in Stage 1. MIGSCHEMA is the default value for Stage 1.
Database user ID	This is the database user ID you used in Stage 1 of migration.	Use the same database user ID that you used for Stage 1. (The default value is your logon ID.)
Database password	This is the database password you used in Stage 1 of migration.	Use the same database password that you used for Stage 1. (The default value is your logon password.)
Log file location	This is the location where a log file will be written.	Enter a valid location (drive and directory) in the file system.

Table 58. Preferences to enter on first page of wizard (continued)

Preference	Meaning	Value
Log file name	This is the name of the log file where you want the Stage 3 messages to be written.	Enter a valid file name.

b. On the second page of the wizard, edit the preferences on the second page, which are described in the following table:

Table 59. Preferences to enter on second page of wizard

Preference	Meaning	Value
Java or COBOL radio button	This choice determines whether the migration tool creates projects that include Java Source folders.	If you are using Rational Web Developer or Rational Application Developer, you should always select the Java radio button. For other Rational Developer products, you can select COBOL if you only plan to generate COBOL.
Latest version	This specifies that the latest version of the desired migration sets should be imported.	Select one of the radio buttons.
Oldest version	This specifies that the oldest version of the desired migration sets should be imported.	Select one of the radio buttons.
Override existing files	Stage 3 (the import process) uses EGL produced by Stage 2 to create and import the EGL files specified in the Stage 1 report. If EGL files with the same names as the EGL files that Stage 3 is about to import already exist in the workspace, this option determines whether or not those files will be overwritten.	*Override existing files* enables you to specify how you want the Stage 3 migration tool to handle the situation in which the migration set you are currently migrating contains parts that should be placed in a file that is already in your workspace. If you select *Override existing files*, the Stage 3 migration tool replaces the existing file and includes **only** those parts that are in the **current** migration set. If you do not select *Override existing files*, the Stage 3 migration tool merges any new parts into the existing file. The new parts are placed alphabetically by part type. See "Overwriting and merging files" on page 36 for a more complete discussion of the effects of this option.

Table 59. Preferences to enter on second page of wizard (continued)

Preference	Meaning	Value
Save migrated files to temporary directory	This provides the option to save EGL files to a location in the file system. This allows you to access EGL files for multiple versions of a project at the same time. (You can only see one version at a time in the workspace). You can move EGL files directly from here to your repository.	If you plan to migrate multiple versions of a migration set, then do the following: • Select this box so that each version can be written to a different subdirectory. • Specify the *Folder* under which the subdirectories for the versions will be placed. • Do not select *Migrate now*. Migration to a temporary directory should only be done in batch mode because of the resource requirements. If you do select *Migrate now*, the migration tool asks you to confirm that you really want to run in online mode. • Select *Save current configuration to file*. You must also specify the project and file name where the current configuration is to be saved as a .vgmig file. The .vgmig file is required if you select *Save migrated files to temporary directory* regardless of whether you migrate in online or batch mode. If you run Stage 3 in batch mode, point to the saved .vgmig file to specify your migration preferences.
Folder	This is the directory in which you want to save the EGL files. Each migration set version becomes a subdirectory under the directory you specify for Folder.	Specify an existing directory in your file system.
Migrate now	This specifies that you want Stage 3 to run at this time, rather than just setting up the preferences file to migrate at a later time.	You must select either *Migrate now* to run Stage 3 in online mode or *Save current configuration to file* to save your preferences so that you can run Stage 3 in batch mode at a later time. You can select both options if you want to retain a copy of your preferences for later reference. Do not select *Migrate now* if you have already selected *Save migrated files to temporary directory*. Saving to a temporary directory can only be done in batch mode. Selecting *Migrate now* indicates that you want to migrate in online mode.
Save current configuration to file	This enables you to save the preferences you are specifying to a file. You can later run Stage 3 in one of the following ways: • In online mode, right click the saved .vgmig file and select **Start Migration** from the context menu. • In batch mode, use the -importFile option to specify the saved .vgmig file. For details, see "Running Stage 2 in batch mode" on page 136.	You must select either *Migrate now* to run Stage 3 in online mode or *Save current configuration to file* to save your preferences so that you can run Stage 3 at a later time. You can select both options if you want to retain a copy of your preferences for later reference. If you select this option, you must also specify the Path and File Name where the current configuration is to be saved as a .vgmig file. When you run Stage 3 later, point to the saved .vgmig file to specify your migration preferences.
Path	Specifies the project into which the file should be saved.	*projectName*, where *projectName* is the name of the project that you want to contain your saved file.
File Name	This specifies the name of the file to which preferences will be saved.	*fileName*, where *fileName* is the desired name for your file WITHOUT a file extension. The extension *.vgmig* is automatically appended.

c. On the third page of the wizard, select the migration sets to import.

   Note: The VAGen Migration Import from Database wizard only lists migration sets that have been migrated to EGL. This ensures that you run Stage 2 to convert the migration set to EGL source and store the EGL into the migration database before you run Stage 3.

4. Click **Finish**.
5. The migration tool creates the EGL projects, EGL source folder, and EGL packages based on the migration set you selected. The tool extracts the EGL source from the migration database and creates the EGL files based on the migration set. The migration tool also includes import statements and updates the project's EGL build path so that the part references can be resolved.

Running Stage 3 in batch mode

The only difference between running Stage 2 and Stage 3 in batch mode is the wizard that you use to create the .vgmig file. See "Running the Stage 3 tool" on page 139 for details on setting up the .vgmig file to run just Stage 3. See "Running Stage 2 in batch mode" on page 136 for details of the commands to include in your .bat file and descriptions of the options you can specify for batch mode.

Using the migration sets written to temporary directories

If you direct the output of stage 3 to a temporary directory, the migration tool creates one subdirectory for each migration set version. The subdirectory name is of the form *migrationSetName_versionName*. You can use either of the following techniques to bring the projects into a workspace:

- If you only have a few projects in the subdirectory, you can point an existing workspace to each of the projects. This technique does not copy the projects into your workspace; it merely makes the projects available to your workspace. When you modify or delete files in the projects, the change is made on the file system in the directory to which the workspace points.
  1. From an existing workspace, select **Window -> Preferences -> Workbench** and deselect **Build automatically**. This avoids multiple rebuilds while you are bringing in each of the projects.
  2. From the Navigator or Project Explorer view, select **File -> New -> Project -> EGL -> EGL Project** (or **EGL Web Project**).
  3. Select the Browse button for the **Project location** and point to the first project. Type (or copy and paste) the project name in the **Project name** field. You might also want to select **Use build descriptor specified in EGL preference**. Select **Finish**.
  4. Repeat steps 2 and 3 for each of the projects within the subdirectory for the migration set.
  5. Select **Project -> Rebuild All** to rebuild the workspace with the new projects.
- If there are a number of projects in the subdirectory, you might find it more convenient to bring up a workspace for the subdirectory as follows:
  1. Start the Rational Developer product.
  2. When you are prompted for the workspace name, point to the subdirectory containing a migration set version and then select **OK**.
  3. If you have errors in the Problems view indicating that projects could not be built, use the Navigator view to locate any closed projects. Closed projects do not have the plus (+) symbol to the left of the project name. Select the closed

projects by selecting **Open Project** from the context menu. The projects should now be visible on the Project Navigator view.

Chapter 8. Running migration in single file mode

An alternative to running migration using Stages 1 – 3 is running migration in Single File Mode. This process allows you to migrate one External Source Format file directly to an EGL file. To run migration in this mode, you must first export VisualAge Generator parts to an External Source Format file, and then import that External Source Format file into the Rational Developer product. During the import process, the External Source Format file is migrated into one or more EGL files, depending on your preferences.

To export parts from VisualAge Generator, do the following:

1. Start VisualAge Generator on Java (or VisualAge Generator on Smalltalk) and open the VAGen Parts Browser.
2. Select the parts you want to export and right-click the selection.
3. From the context menu, select **Import/Export —> VAGen Export** (or **VAGen Export with Associates**).
4. Type a name for the External Source Format file in the box and click the **Save** button. (If you type the name of an existing file, you will be asked if you would like to add parts to the file or overwrite it. Choose whichever is appropriate for you.)

To set up the Rational Developer product for single file mode, do the following:

1. Start the Rational Developer product and point to your workspace. (For example, d:\workspaces\myworkspace)
2. Set your migration preferences. See "VAGen Migration Syntax Preferences" on page 128 for information on how to do this.
3. From the Workbench window, select **Window -> Preferences -> VAGen Migration**. In general, you should always ensure that the **Separate parts into EGL files** preference is selected. When you select this preference, each program, map group, and table is placed in its own file. This adheres to the EGL requirement of one generatable part per file. If you do not select **Separate parts into EGL files**, all the parts are placed in the same EGL file. See "Overview of Single File Migration" on page 20 for specifics of the parts placement algorithm for single file mode.
4. Create a new EGL project. (For example, MyProject.) Alternatively, you might want to create an EGL Web project if you are planning to develop new EGL page handlers.
5. Under the EGLSource directory for the EGL project, create a new EGL package. (For example, my.pkg)
6. See the section "Running single file migration using the user interface" on page 145 for details on running in online mode or "Running single file migration using batch mode" on page 146 for details about creating a batch command file to process multiple External Source Format files with a single command file.

Running single file migration using the user interface

To import the External Source Format file into the Rational Developer product, do the following:

1. From the Navigator view, select the EGL package in which to put the resulting EGL file.

2. Right-click the package. From the context menu, select **Import**.
3. Select **VAGen External Source Format File** and select **Next**.
4. In the **Input file name** field, enter the name of the External Source Format file you want to import.
5. In the **Source folder** field, enter the name of the project and source folder in which to put the EGL file. (For example, MyProject\EGLSource)
6. In the **Package name** field, enter the name of the package in which to put the EGL file. The migration tool also uses the package name you specify for the package statement within the EGL file. (For example, my.pkg)
7. In the **EGL file name** field, enter the name of the EGL file that will be created from your External Source Format file. By default, the EGL file name will be the same as the External Source Format file, but with the .egl file extension. See "Overview of Single File Migration" on page 20 for information about how the migration tool uses the *Separate parts into EGL files* preference and the type of parts in the External Source Format file to determine what files to create during migration in single file mode.
8. In the **Log file location** field, enter the drive and directory where the migration log file is to be placed. In the **Log file name** field, enter the name for the migration log file. The Log file name defaults to match the name of the External Source Format file that you specified. The migration log file contains any messages written during migration.
9. Click **Finish**. If the file name you specified in the EGL file name field already exists in the container you specified in the Container field, you are prompted to append or overwrite to the file.
10. When the migration completes, you should notice the following:
    - One or more EGL files should be listed in the project, source folder, and package that you specified. See "Overview of Single File Migration" on page 20 for information about how migration tool uses the *Separate parts into EGL files* preference and the type of parts in the External Source Format file to determine what files to create during migration in single file mode.
    - Any error messages appear in a pop-up window. If you did not specify a Log file location, you can use the **Save to File** button to save the messages in a file. Be sure to close the pop-up window.
11. Select the project and then select **Project —> Build Project**. This causes validation to run so that the Problems view reflects the most current messages for all files in the project.

Running single file migration using batch mode

The user interface enables you to migrate one External Source Format file at a time. With batch mode, you can migrate multiple External Source Format files in a single command file. To use batch mode do the following:

1. Create a file with a .bat file extension. For Windows environments, the contents of the .bat file should be:

```
set path=InstallDirectory\eclipse\jre\bin;%path%
set classpath=InstallDirectory\eclipse\startup.jar;
        InstallDirectory\egl\eclipse\plugins\
        com.ibm.etools.egl.vagenmigration_version\runtime\eglMigration.jar;
cd InstallDirectory
java com.ibm.etools.egl.internal.vagenmigration.batch.VGMIG
    -importFile Path\ExternalSourceFormatFile.esf
    -eglFile Path\EGLFile.egl
```

```
    -data Path\workspace
    -package packageName
    -overwrite
    >Path\LogName.log
```

For Linux environments, the contents of the .bat file should be the following:

```
# !/bin/bash
export PATH=/InstallDirectory/eclipse/jre/bin/:$PATH
export CLASSPATH=/InstallDirectory/eclipse/startup.jar:
  /InstallDirectory/egl/eclipse/plugins/
  com.ibm.etools.egl.vagenmigration_version/runtime/eglMigration.jar:$CLASSPATH
cd /InstallDirectory/
java com.ibm.etools.egl.internal.vagenmigration.batch.VGMIG
  -importFile /Path/ExternalSourceFormatFile.esf
  -eglFile /Path/EGLFile.egl
  -data /Path/workspace
  -package packageName
  -overwrite
  >/Path/LogName.log
```

In addition, for Linux environments, you must change the permissions for the .bat file so that it is executable. To make the .bat file executable, bring up a Command Terminal and enter the following command, where *mybatchfile.bat* is the name of the .bat file you created:

```
chmod u+x mybatchfile.bat
```

Note:

- The following statements must be written so that the statement is all on one line:
  - For Windows environments, the **set classpath** statement.
  - For Linux environments, the **export classpath** statement.
  - The **java** statement.
- Repeat the **java** statement once for each External Source Format file you want to migrate.
- *InstallDirectory* is the drive and directory in which you installed the Rational Developer product. You must include the *InstallDirectory* for the path statement, the classpath statement, and the cd (change directory) statement.

  Note: If you installed and kept a Rational Developer product before installing the product that you are using now, the installation directory of interest may be the directory that was used in the earlier install.
- *version* is the plugin version number (for example, 6.0 or 6.0.0.1). In general, you should use the highest version number you see for the com.ibm.etools.egl.vagenmigration_*version* plugin.
- *Path**ExternalSourceFormatFile.esf* refers to the drive, directory, and file name of the External Source Format file you want to migrate. (For example, d:\temp\VAGenFiles\PROG1.esf.)
- *Path**EGLFile.egl* refers to the drive, directory, and file name of the EGL file you want to create. The directory must include the workspace, EGL source folder, and package where you want to place the EGL source file. (For example, d:\myworkspace\MyProject\EGLSource\my\pkg\prog1.egl.) *EGLFile.egl* is used in the same way as the EGL file name field you specify when you use the Import VAGen External Source Format File wizard. See "Overview of Single File Migration" on page 20 for information about how migration tool uses the *Separate parts into EGL*

files preference and the type of parts in the External Source Format file to determine what files to create during migration in single file mode.

- *Path\workspace* is the drive and directory for your workspace. (For example, d:\workspaces\myworkspace) If you do not specify the *-data* option, anything you specified in the VAGen Migration Syntax Preferences is ignored and the migration tool uses the default VAGen Migration Syntax Preferences. If you want to specify VAGen Migration Syntax Preferences, you must specify the *-data* option and point to the workspace in which you set the preferences.
- *packageName* is the name of the package with which you want to associate the EGL file. (For example, my.pkg.) The package name is also used in the package statement of the .egl files that the migration tool creates.
- The *-overwrite* parameter is optional. This parameter tells the migration tool whether or not to overwrite an existing EGL file in the specified directory with the specified name.
- *Path\LogName* refers to the location and file name of the log file you want to create for the migration of the corresponding External Source Format file. Sending your migration messages to a log file is also optional, but it is highly recommended.

For Windows environments, an example of the java command might look something like this (though it should be all on **one** line):

```
java com.ibm.etools.egl.internal.vagenmigration.batch.VGMIG
   -importFile d:\temp\VAGenFiles\prog1.esf
   -eglFile d:\workspaces\myworkspace\MyEGLProject\EGLSource\my\pkg\prog1.egl
   -data d:\workspaces\myworkspace
   -package my.pkg -overwrite >d:\temp\EGLLogs\prog1.log
```

For Linux environments, an example of the java command might look something like this (though it should be all on **one** line):

```
java com.ibm.etools.egl.internal.vagenmigration.batch.VGMIG
  -importFile /d:/temp/VAGenFiles/prog1.esf
  -eglFile /d:/workspaces/myworkspace\MyEGLProject/EGLSource/my/pkg/prog1.egl
  -data /d:/workspaces/myworkspace
  -package my.pkg  -overwrite  >/d:/temp/EGLLogs/prog1.log
```

2. Shut down the Rational Developer product.
3. Open a command prompt window, navigate to the directory containing your .bat file, and run your .bat file.
4. When the process completes, your EGL files and log files will be stored in the directories you specified for them, respectively. The log file contains a list of the migrated parts and any error messages. The messages are the same messages that are listed in the pop-up window when you use the Import Wizard in online mode.
5. Start the Rational Developer product.
6. Select the project into which you imported External Source Format files, then right-click and select **Refresh**. This refreshes the project from the file system so that the EGL files that were created, appended, or overwritten during migration in batch mode are recognized by the Rational Developer product. This in turn causes validation to run so that the Problems view reflects the most current messages for all files in the project. Then you can expand the package you created to see your EGL files.

Part 5. Completing the migration

Chapter 9. Completing your migration

After you have migrated your source code using Stages 1 – 3 migration or single file migration, there are some additional tasks you should do. This includes the following:

- Export your preferences.
- Save a baseline for the EGL projects and packages in your source code repository.
- Preliminary steps for completing single file migration.
- Review your EGL source code.
- Review your EGL build descriptor parts.
- Review your EGL linkage option parts.
- Review your EGL resource associations parts.
- Prepare for debugging.
- Generate and test with COBOL generation.
- Generate and test with Java generation.
- Review your standards

Exporting your preferences

After you have worked with EGL during your pilot project, you might have set additional preferences beyond those that are required or recommended in "Required EGL preferences" on page 127, "Recommended preferences" on page 128, and "VAGen Migration Syntax Preferences" on page 128. For example, you might have set other preferences related to your source code repository and the library management process you have chosen. You can export your preferences to a file so that other developers can import the preferences to have as a starting base for their own preferences. The export technique is also an easy way to move preferences from one workspace to another. To export your preferences, do the following:

- Select **Window -> Preferences**.
- Select **Export** in the lower left corner of the Preferences window.
- Point to a file where you want to save your preferences.

Other developers can import your preferences into a workspace by doing the following:

- Select **Window -> Preferences**.
- Select the EGL capability as described in "Required EGL preferences" on page 127.
- Select **Window -> Preferences**.
- Select **Import** in the lower left corner of the Preferences window.
- Point to the file where you saved your preferences.

Note: This technique does not have any effect on the settings for perspectives and views. It only changes the preferences.

Saving a baseline for EGL projects and packages

Before you attempt to resolve any messages in the Problems view or modify any migrated EGL code, you might want to create a version of the EGL projects and packages in your source code repository. Storing and versioning the EGL projects and packages immediately after migration provides a baseline so that you know exactly what source code was produced by the migration tool. This baseline also provides a way of tracking any code changes you have to make by hand. This is especially useful during a pilot project as a way of capturing all the changes so that you can document the types of changes that were necessary. This documentation can serve as an aid in migrating additional subsystems.

Preliminary steps for completing single file migration

Single file migration does not do everything that Stage 1 – 3 migration does. You must do the following steps manually:

- Nest any forms within their corresponding form group. This is required if you migrate two form groups to the same package and the two form groups contain the same form name. (For example, MAP1).
- Resolve any duplicate parts within the same EGL package. This can occur if you migrate two programs with their associates to the same EGL package and the two programs share some common parts. You can split the common part definitions into a centralized common file or you can remove the duplicate parts from one of the files. If all the files are in the same package, you do not need to modify the EGL build path property or add import statements.
- Update the EGL build path property for the current project to list all the projects that the current project needs to reference to resolve any part names. To update the EGL build path, in the Project Navigator view, select the current project, right-click the selection, and then select **Properties**. On the EGL Build Path page, on the Projects tab, select the additional projects that the current project needs to reference. Be sure to include in the EGL build path any projects that contain packages that files in the current project need to import. For example, if FileA is in ProjectB and FileA needs to import packageC, be sure to include the project where packageC is located in the EGL build path for ProjectB.
- Add any import statements to your EGL file to point to the common packages that your file needs to reference. The packages that you specify for the import statement must exist in projects that are specified for the EGL build path of the project in which the current EGL file is located. For example, if FileA is in ProjectB, then the import statements in FileA can only reference packages that are located in projects specified for the EGL build path of ProjectB.

Common steps for both Stage 1 — 3 and single file migration

Reviewing your EGL source code

You need to perform the following steps regardless of whether you used Stage 1 – 3 migration or single file migration:

- Review and resolve the errors in the migration log or the ″TODO″ list log. These errors reflect ambiguous situations that the migration tool was not able to resolve. Modify your EGL source code to resolve these errors. For example, if you used the VAGen RETR statement and did not explicitly specify a search column, then if the table was not available during migration, the EGL syntax includes EZE_UNKNOWN_SEARCH_COLUMN. You must update your EGL source code with the correct search column name based on the table definition.

See Appendix D, "Messages in the Problems view," on page 299 for help with resolving specific strings that the migration tool uses when it creates intentionally invalid EGL syntax.

- If you have program, table, or form group names that are reserved words, you must change the part name. If you generate COBOL, you might want to set the *alias* property for the part to specify the original part name. That will help you avoid having to change any external references to the program, table, or form group.
- Review and resolve any additional errors in the Problems view. See Appendix E, "IWN.xxx messages in the Problems view," on page 303 for help in resolving common messages in the Problems view that are a result of the migration process.
- Determine if you need to set the *containerContextDependent* property for any records or functions. For more details, see "containerContextDependent Property" on page 30.

Reviewing your EGL build descriptor parts

The migration tool converts VAGen generation options parts to EGL build descriptor parts. However, some VAGen options have no EGL equivalent. In addition, EGL has several new build descriptor options that you might need to set. You might see errors in the Problems view due to either of these changes. See Appendix E, "IWN.xxx messages in the Problems view," on page 303 for help in resolving common messages in the Problems view that are a result of the migration process. You might need a text editor to resolve some of the problems. You need to perform the following steps regardless of whether you used Stage 1 – 3 migration or single file migration:

- Review general build descriptor options.
- Review COBOL generation build descriptor options.
- Review Java generation build descriptor options.
- Establish a debug build descriptor part.

Reviewing general build descriptor options

You need to review the following build descriptor options regardless of whether you plan to generate COBOL or Java:

- If you generate for a national language in which the decimal point is the comma symbol, you must include the *decimalSymbol* build descriptor option.
- If you use the VAGen EZESYS special function word to determine your runtime environment, you might want to set the EGL build descriptor option *eliminateSystemDependentCode*. Refer to the online helps for more information about this option.
- Refer to the online helps for information about master build descriptors. This technique is a replacement for the VAGen preference for the Default generation options part. The migration tool automatically splits any generation option part that contains the NOOVERRIDE attribute into two build descriptor parts. One of the build descriptor parts is named *xxxxx*, and the other is named *xxxxx_NOOVERRIDE*, where *xxxxx* is the original VAGen generation options part name. The part named xxxxx contains the EGL replacement for all the VAGen generation options that did not specify the NOOVERRIDE attribute. The part name xxxxx_NOOVERRIDE contains the EGL replacement for all the VAGen generation options that specified the NOOVERRIDE attribute. This split into two parts is required if you decide to use master build descriptors.
- If you used the VAGen /OPTIONS generation option to chain generation options parts, review how your EGL build descriptor parts chain using the

nextBuildDescriptor option. You might need to modify this chaining to obtain the same set of build descriptor options that you had in VisualAge Generator.

- The migration tool does not create a default build descriptor for you when it creates the EGL projects. You can establish a default build descriptor for a file, package, EGL source folder, project, or workbench levels. The default build descriptor that is closest to the generatable part is the one that is used. For example, you can specify a default build descriptor for just one file and specify a different default build descriptor for the workbench. In this situation, if you generate the program contained in the file, the default build descriptor for the file is used. When you generate any other program, then the workbench default build descriptor is used.
  - To set a preference for a particular file, package, EGL source folder, or project, select the resource (file, package, folder, or project), then right-click and select **Properties** from the context menu. Select *EGL Default Build Descriptor* in the left pane. Select the build descriptor that you want to use as the default for all generatable parts in this resource. Assuming there is no closer EGL default build descriptor, the *Target system build descriptor* is the default that will be used whenever you generate anything in this resource. The *Debug build descriptor* is the default that will be used when you use the debugging tool.
  - To set a workbench preference for a build descriptor part, select **Window -> Preferences -> EGL -> Default Build Descriptor**. This preference applies to all projects, packages, source folders, and files if you do not specifically override it. You can set both a *Target system build descriptor* to use for generation and a *Debug build descriptor* to use with the debugging tool.
- If your control parts (build descriptor, linkage options, resource association, bind control, and linkedit parts) are not all in the same file, you must modify the current file to include import statements for the files that contain other build parts that you want to reference from the current file. For example, if buildDescriptorPartA references a linkageTableB that is in a different file, the file containing buildDescriptorPartA must include an import statement for the file that contains linkageTableB. Use the EGL Build Parts Editor to add the import statement.

Reviewing COBOL generation build descriptor options

If you plan to generate COBOL, you need to review or set the following build descriptor options:

- For VisualAge Generator, the outputs of COBOL generation are transferred to the host to run the preparation steps. EGL uses a build server to handle the preparation steps. For the EGL build server, there is a port number that must be specified to transfer the outputs of generation. Contact the person who installed and configured the Enterprise Developer build server to determine the port number on which the remote build server is listening for build requests. If the port is other than 5555, you must add the *destPort* build descriptor option to set the value for your organization.
- If you generate COBOL for the zOS environment, you must also do the following:
  - Establish a bind control part to use as a template.
  - Establish a program-specific bind control part.
  - Review linkedit commands.

Establishing a bind control part to use as a template: VisualAge Generator uses a bind control template to create default bind control commands. The default VAGen template binds a DB2 plan, but you might have modified the template so that it binds a package or made other changes to conform to the standards of your

organization. The VAGen templates are stored outside the workspace in files named EFK2MBDA.tpl and EGK2MBDD.tpl. Bind control parts are only required if you need to do a special bind for a particular program.

EGL does not use bind control templates. Instead EGL requires a bind control part. If you bind packages, you can achieve an effect similar to VisualAge Generator templates by creating an EGL bind control part that contains a template to use for all the binds and store this part in your workspace. Note: The technique described in this section does not work if you bind plans. See "Establishing a program-specific bind control part" on page 156 if you bind plans.

If you modified the VAGen bind control template so that you bind a package for each program, you can adapt that template for use as an EGL bind control part. It is recommended that you put this bind control part in the same file with other control parts. For example, you might have a VAGen bind control template that looks like the following:

```
DSN SYSTEM(%MYDB2SUBSYSTEM%)
BIND PACKAGE(%MYCOLLECTIONNAME%) -
     MEMBER(%EZEMBR%) -
     .
     .
     .
```

In the previous example, MYDB2SUBSYSTEM and MYCOLLECTIONNAME are symbolic parameters you set in your VAGen generation options and EZEMBR is set automatically with the name of the program currently being generated.

For binding packages, the EGL bind control part that you need to create is very similar to the VAGen template, but requires 3 additional lines and a change to the EZEMBR symbolic parameter. The corresponding EGL bind control part looks like the following:

```
TSOLIB ACTIVATE DA('%DSNLOAD%')
ALLOC FI(DBRMLIB) SHR DA('%EZEPID%.%SYSTEM%.DBRMLIB' +
'%ELA%.SELADBRM')
DSN SYSTEM(%MYDB2SUBSYSTEM%)
BIND PACKAGE(%MYCOLLECTIONNAME%) -
     MEMBER(%EZEALIAS%) -
     .
     .
     .
```

DSNLOAD, EZEPID, and ELA all have the same meaning as they did in VisualAge Generator. EZEALIAS is the EGL replacement for EZEMBR when you need the name of the program being generated in a bind control part. SYSTEM is the EGL replacement for EZEENV. You might need to modify the first 3 lines of the bind control part if you use different data set naming conventions on your EGL build server. Contact the person who installed and configured the Enterprise Developer build server to determine what the additional 3 lines need to be based on the naming conventions for your organization. You might also need to modify your EGL build descriptor options to set the projectID build descriptor option and the DSNLOAD and ELA symbolic parameters if you did not set these values in VisualAge Generator. See "Symbolic parameters" on page 273 for changes to the names of the symbolic parameters. Also see the online helps for more information about using a template for the EGL bind control part and setting the values of EGL symbolic parameters.

In addition to creating the EGL bind control part to serve as a template, you must also modify your build descriptor parts to include the *bind* build descriptor option

to point to your bind control part. It is recommended that you add the *bind* build descriptor option to one of your existing, common build descriptor parts to minimize the number of build descriptor parts you need to modify.

Establishing a program-specific bind control part: If you bind plans in VisualAge Generator, then generally each program requires a different bind command. In this case, you need a program-specific bind command to bind a plan for the program with all the other programs that are in the same run unit. The typical way to do this is to create a bind control part called *xxxxx.BND*, where *xxxxx* is the name of the program. You then set the VAGen generation option /BIND=BND to specify the suffix that you want VisualAge Generator to use when searching for a program-specific bind command. The .BND suffix can also be used if you bind packages for the rare situations in which one program requires something different from what the template provides.

The EGL *bind* build descriptor option does not permit you to specify a suffix. Instead, the *bind* build descriptor option must specify the name of a specific bind control part. In EGL, if you do not specify the *bind* build descriptor option, then EGL looks for a bind control part with the same name as the program. In general, the easiest technique is to bind packages and follow the process described in "Establishing a bind control part to use as a template" on page 154. However, if you want to bind plans or if you have the situation in which one program requires something other than what is provided by the bind control part template, you can create program-specific bind control parts.

If you have VAGen program-specific bind control parts that used the default .BND suffix, then the migration tool automatically removes the .BND suffix for you and adds the additional statements required for an EGL bind control part. Assuming that your naming convention was programName.BND and you always have program-specific bind command parts, then you do not need to specify the EGL *bind* build descriptor option for this program. However, if you are using the EGL *bind* build descriptor to specify a bind control part for most programs to use as a template and you need to provide a program-specific bind control part for a program, then you must create a build descriptor part for this specific program and set the *bind* build descriptor option to point to the program-specific bind control part. Otherwise, EGL will pick up the bind control part that is the template because that is what your normal *bind* build descriptor option specifies.

Review linkedit commands: VisualAge Generator provides default linkedit commands based on the target environment and database access. However, in some cases, you might have a program that requires specific linkedit commands. (For example, to link in a PL/I program for the MVS Batch environment.) The typical way to do this is to create a linkedit part called *xxxxx.LKG*, where *xxxxx* is the name of the program. You then set the VAGen generation option /LINKEDIT=LKG to specify the suffix that you want VisualAge Generator to use when searching for the program-specific linkedit command.

The EGL *linkEdit* build descriptor option does not permit you to specify a suffix. Instead, the *linkEdit* build descriptor option must specify the name of a specific linkedit part. In EGL, if you do not specify the *linkEdit* build descriptor option, then EGL looks for a linkedit part with the same name as the program. If EGL does not find a linkedit part with the same name as the program, then EGL creates default linkedit commands based on the target environment and database access similar to what VisualAge Generator does. Therefore, the only time you need to specify the *linkEdit* build descriptor option is if you create a linkedit part with a

different name from the program. You might need to do this if you generate the same program for both the zOS Batch and zOS CICS environments.

If you have VAGen program-specific linkedit parts that used the default .LKG suffix, then the migration tool automatically removes the .LKG suffix for you. Assuming that your naming convention was programName.LKG, then you do not need to specify the EGL *linkedit* build descriptor option for this program. EGL will find the program-specific part first, before it attempts to create a default linkedit command.

Reviewing Java generation build descriptor options

If you plan to generate Java, you need to review or set the following build descriptor options:

- Be sure to add the *genProject* build descriptor option to specify where the outputs of Java generation are to be placed. There is no VAGen generation option that migrates to the EGL *genProject* build descriptor option.
- There are some EGL build descriptor options that have somewhat different behavior from their corresponding VAGen generation option. Refer to the online helps for information about the following build descriptor options to determine whether you need to set or modify them for your environment:
  - genProperties, which is set by the migration tool based on the VAGen /genproperties option.
  - enableJavaWrapperGen, which is set by the migration tool based on the VAGen /system=JAVAWRAPPER option.
- There are some new EGL build descriptor options that have no corresponding VAGen generation option. Refer to the online helps for information about the following build descriptor options to determine whether you need to set them for your environment:
  - dateMask
  - cicsj2cTimeout
  - sessionBeanID
  - sqlJDBCDriverClass
  - sqlJNDIName
  - sqlValidationConnectionURL

Establishing a debug build descriptor part

Create a build descriptor part that contains the build descriptor options that you want to use during debug. See the online helps for guidance on creating a debug build descriptor part.

Reviewing your EGL linkage option parts

The migration tool converts VAGen linkage table parts to EGL linkage options parts. However, some VAGen options have no EGL equivalent. In addition, EGL has several new linkage options that you might need to set. You might see errors in the Problems view due to either of these changes. See Appendix E, “IWN.xxx messages in the Problems view,” on page 303 for help in resolving common messages in the Problems view that are a result of the migration process. Also refer to the online helps for details about the linkage options that are supported for your environment. You need to perform the following steps regardless of whether you used Stage 1 – 3 migration or single file migration:

- Review and resolve the messages in the migration log and in the Problems view. You might need a text editor to resolve some of the problems.

- For callLink, consider the following:
  - Not all of the link types from VisualAge Generator are supported in EGL. For example, CSOCALL is no longer supported. The migration tool converts CSOCALL to a remoteCall. However, the attributes you must specify for an EGL remoteCall differ from those for CSOCALL.
  - Not all of the remoteComType values from VisualAge Generator are supported in EGL. For example, DCE and DCESECURE are no longer supported. The migration tool converts these unsupported values exactly as they are, which results in an invalid EGL linkage options part. This ensures that there is an error in the Problems view as a reminder that you must modify the linkage options part to specify the option you want to use with EGL.
  - If you change to use remoteComType=CICSECI, you must add the *ctgPort* and *ctgLocation* attributes. This does require the configuration and setup of a CICS Transaction Gateway Server for the invocation of remote CICS transactions.
  - If you change to use remoteComType=CICSSSL, you must add the *ctgKeyStore* and *ctgKeyStorePassword* attributes. In addition, if you have not already included the *ctgPort* and *ctgLocation* attributes in your VAGen linkage table, you must include them for the EGL remoteComType=CICSSSL.
  - conversionTable=BINARY is not supported in EGL. The migration tool converts this value exactly as it is so that there is a place holder in the EGL linkage part. However, you must modify the value.
- For fileLink, conversionTable=BINARY is not supported in EGL. The migration tool converts this value exactly as it is so that there is a place holder in the EGL linkage part. However, you must modify the value.
- For asynchLink (VAGen crtxlink), conversionTable=BINARY is not supported in EGL. The migration tool converts this value exactly as it is so that there is a place holder in the EGL linkage part. However, you must modify the value.
- The EGL Transfer To Program linkage information is the equivalent of the VAGen dxfrlink entry. If you generate for Java and use the VAGen XFER statement, you might need to add EGL Transfer to Transaction entries. Refer to the online helps for information about this new linkage entry.

Reviewing your EGL resource association parts

The migration tool converts VAGen resource association parts to EGL resource associations parts. However, some VAGen options have no EGL equivalent. In addition, EGL has several new resource association options that you might need to set. You might see errors in the Problems view due to either of these changes. See Appendix E, “IWN.xxx messages in the Problems view,” on page 303 for help in resolving common messages in the Problems view that are a result of the migration process. Also refer to the online helps for details about the resource association options that are supported for your environment. You need to perform the following steps regardless of whether you used Stage 1 – 3 migration or single file migration:

- Review and resolve the messages in the migration log and in the Problems view. You might need a text editor to resolve some of the problems.
- Not all of the file types from VisualAge Generator are supported in EGL. For example, BTRIEVE and MFCOBOL are no longer supported. The migration tool converts these unsupported options exactly as they are so that there is a place holder in the resource association part. This ensures that there is an error in the Problems view as a reminder that you must modify the resource associations

part to specify the option you want to use with EGL. Depending on the EGL file type option you select, there might be additional attributes you must set for the resource association entry.

- Review the online helps for the *FormFeedOnClose* and *text* attributes to determine if you need to set these values for your environment. In VisualAge Generator, the equivalent options, /noff and /text respectively, can only be specified in a resource association file for the workstation environment. Therefore, these options are not set by the migration tool because the tool only processes resource association parts.

Preparing for debugging

You should do the following to prepare for debugging:

- From the workbench window, select **Window -> Preferences -> EGL -> Debug**. Refer to the online helps to determine which, if any, of these preferences you need to set for your environment.
- Also review the **EGL-> Default Build Descriptor** preferences. You might want to set the default Debug build descriptor for your entire workspace. Alternatively, you can set the default Debug build descriptor for a project, EGL source folder, package, or file.
- If you are calling generated EGL or non-EGL programs on a remote CICS system from the debugger, you need to configure and use a CICS Transaction Gateway Server. Direct calls to CICS using CICS Client or CICS Transaction Gateway are no longer supported.

Generating and testing with COBOL generation

You should do the following to prepare for COBOL generation:

- You might want to create another version of the EGL projects and packages in your source code repository at this time. This provides another baseline of code that reflects the changes you made manually and before generation.
- If you are generating for zOS, be sure that the Enterprise Developer Server for zOS Version 5.0 is installed with all of the latest PTFs.
- If you are generating for iSeries, be sure that the EGL runtime for iSeries has been installed in your host environment with all the latest PTFs.
- Be sure the EGL build server is installed and configured in your host environment. In VisualAge Generator, customization was done to the preparation process for zOS and iSeries by changing preparation templates on the workstation. In EGL, this customization is done on the host machine. See the online helps for more information regarding if customization is still needed and how the customization is done for each target host environment.
- Contact the person who installed and configured the Enterprise Developer build server. Be sure you understand any changes to the naming conventions for the host data sets that contain the outputs of generation and preparation. For example, in VisualAge Generator when you generate for MVS Batch, the default name of the data sets is *xxxx.MVSBATCH.yyyy*, where *xxxx* is the high-level qualifier you specify in the */projectid* generation option and *yyyy* is the type of code. (For example, EZESRC for the COBOL source.) With EGL, because the target environment names have changed, the corresponding data set names are *xxxx.ZOSBATCH.yyyy*. This means that you might need to define a new group of data sets on the host.
- Generate your programs and tables. When you generate the programs, use the following build descriptor options:
  - genFormGroup="YES"

- genHelpFormGroup="YES"
- genDataTables="NO"

This enables you to generate the form groups with the programs that use them, but to only generate the tables one time regardless of the number of programs that use the table. Resolve any validation errors that are caught during generation.

- Test the generated code.
- You might want to create another version of the EGL projects and packages in your source code repository at this time. This provides another baseline of code that reflects the changes you made as a result of problems found during generation and testing.

Generating and testing with Java generation

You should do the following to prepare for Java generation:

- You might want to create another version of the EGL projects and packages in your source code repository at this time. This provides another baseline of code that reflects the changes you made by hand and before generation.
- Generate your programs and tables. When you generate the programs, use the following build descriptor options:
  - genFormGroup="YES"
  - genHelpFormGroup="YES"
  - genDataTables="NO"

  This enables you to generate the form groups with the programs that use them, but to only generate the tables one time regardless of the number of programs that use the table. Resolve any validation errors that are caught during generation.
- Test the generated code.
- You might want to create another version of the EGL projects and packages in your source code repository at this time. This provides another baseline of code that reflects the changes you made as a result of problems found during generation and testing.

Reviewing your standards

You might want to review your current coding standards and set new standards to be used for any new code that is written. For example, if you generate COBOL, some standards that you might want to consider:

- Use underscore rather than hyphens in your part names. COBOL programs do not permit the use of underscore for names. However, COBOL generation automatically changes the underscore to a hyphen, so the generated COBOL will still be readable. The only time an alias name is assigned is if there are duplicate part names after changing the underscore to a hyphen.
- To improve the readability of generated COBOL code, avoid the use of alias name. You can do this by using the following naming conventions:
  - Record and function names should be 18 or fewer characters. This is the same limit as in VisualAge Generator.
  - Data item names should be 27 or fewer characters. This is slightly less than the recommended limit of 30 characters and the maximum of 32 characters for VisualAge Generator.
- Program and table names can now be 8 characters.

Part 6. Language and runtime differences

There are various language and runtime differences between VisualAge Generator and EGL.

Chapter 10. Language and runtime differences

Language differences

Refer to "Determining whether you can migrate to EGL" on page 6 for information about areas in which the Rational Developer product is not a complete replacement for VisualAge Generator Developer.

Refer to Chapter 3, "Handling ambiguous situations," on page 43 for details about VAGen language elements or migration strategies that do not allow a precise migration to the EGL language.

Refer to Appendix B, "Relationship of VisualAge Generator and EGL Language Elements," on page 177 for details about how each VAGen language element is migrated to EGL.

Runtime differences

After you have migrated your source code to EGL, you should generate and thoroughly test your code to ensure that it runs the same as in VisualAge Generator. The specific runtime differences vary depending on the target environment as follows:

- General differences.
- Differences in debug.
- Differences in generated COBOL.
- Differences in generated Java.
- Differences between distributed CICS and native workstation environments.
- Differences between generated C++ and generated Java.

General differences

The following runtime behavioral differences can occur without any messages in the migration log or the Problems view. The problems can occur during debug or when running the generated Java or COBOL code:

- The following apply to text programs:
  - A runtime error occurs if a form field is not long enough to contain all the digits and formatting information (sign, decimal point, currency symbol, and numeric separator).
  - Non-default fill characters are always honored, even if the program does not issue a SET formItem FULL statement.
  - Arrays on forms always use the validation and formatting properties of the first element of the array. This might result in slightly different behavior from VisualAge Generator, which allowed some of these properties to vary for the elements of the array. For details, see "Map arrays and attributes" on page 59.
  - If any maps contained fields at row=0, column=0, be sure to test the programs that use the corresponding forms for any differences in appearance or behavior. For details, see "Fields at row=0, column=0" on page 61.
- If a record that is a VAGen REDEFINED record is not available when migrating a program, the migration tool does not include the EGL *redefines* property in the data declarations. This results in two separate record areas, rather than a single

area with two definitions as in VisualAge Generator. Errors, including abends, can result due to uninitialized or invalid data. See "Redefined records" on page 47 for details.

- Hard I/O errors occur in more situations in EGL than in VisualAge Generator:
  - In VisualAge Generator, UNQ for non-SQL records is a soft error so the HRD I/O error state is not set. In EGL, *unique* is a hard error so *hardIOError* also tests true. See "I/O error values UNQ and DUP" on page 79 for details.
  - For iSeries, the VAGen I/O error value LOK is migrated to the EGL *deadlock* I/O error state. In VisualAge Generator, LOK is a soft error so the HRD I/O error state is not set. In EGL, *deadlock* is a hard error so *hardIOError* also tests true. See "I/O error value LOK" on page 81 for details.
- If the I/O error routine is not available when migrating a function, the migration tool assumes that the I/O error routine is not a main function and converts to a function invocation. In VisualAge Generator, if the I/O error routine is a main function and an error occurs at runtime, VisualAge Generator clears the current execution stack of functions and starts a new stack with just the main function that is specified as the I/O error routine. This also clears out any storage for the execution stack. In EGL, because the migration tool converted to a function invocation, if an error occurs at runtime, EGL adds the main function to the current execution stack rather than cleaning out the stack and starting a new stack with just the main function. This has the potential for an infinite loop or a large use of resources if functions have local storage or parameter lists. See "I/O error routine" on page 67 for details.

Differences in debug

There are some differences in debug that might affect your testing. If you generate for COBOL environments, you need to be particularly aware of these differences because debug does not provide the same support as generated COBOL in the following areas:

- Differences for maps are as follows:
  - Blink is not supported for text forms.
  - The *isDecimalDigit* property is only supported for character fields. It is implemented as a software edit, not as a hardware attribute. Numeric fields also have a software edit. See "Map fields and the numeric hardware attribute" on page 58 for details.
- For indexed records that have an alternate index record defined, the setting for the DUP I/O error value differs from VisualAge Generator. For VisualAge Generator, for a SET record SCAN followed by a SCAN or SCANBACK I/O option, the DUP I/O error value is not set for the SET record SCAN statement. The DUP I/O error value is set for each of the duplicate-keyed records other than the last record retrieved with a duplicate key. For Java generation, a *set* record *position* followed by a *get next* or *get previous* statement results in the *duplicate* I/O error state being set on the *set* record *position* rather than on the first duplicate-keyed record retrieved. The remaining duplicate-keyed records result in the *duplicate* I/O error state being set the same as in VisualAge Generator. The EGL duplicate state is set on all records other than the first and last of the duplicate-keyed records. See the online helps for more information about indexed records and alternate index records and their use with *set* record *position*, *get next* and *get previous*.
- Differences for SQL are as follows:
  - ODBC is not supported in EGL. If you use a SQL database manager other than DB2, you must obtain a JDBC driver for your database manager.

- JDBC does not support two-phase commit. Therefore, there are the following differences:
  - There are separate calls to the SQL manager and MQ series manager for commit and rollback. Therefore, if a problem occurs, it is possible for resource to commit or rollback without the corresponding commit or rollback for the other resource.
  - EZECONCT (EGL sysLib.connectionService). In VisualAge Generator, the R option for *unit of work* argument changed the connection to another database without ending the current connection. This permitted you to update multiple databases within the same unit of work. In EGL, the R option, as well as the D1C, D2A, D2C, and D2E options for the *unit of work* argument are all treated as though you specified D1E. D1E is a one-phase commit, but does not automatically release the database connection. You must explicitly request the DISC, DCURRENT, or DALL option to disconnect the database. See the online helps for the sysLib.connectionService for details.
- EZESQISL (EGL sysVar.sqlIsolationLevel). In VisualAge Generator, a value of 1 means you want cursor stability. In EGL a value of 1 means you want serializable transactions.
- EZESQRRM (EGL sysVar.sqlerrmc) is not supported.
- EZESQWN6 (EGL sysVar.sqlwarn[7]) is not supported.
- EZESQLCA (EGL sysVar.sqlca) fields are limited. They do not include values for EZESQRRM and EZESQWN6.

Differences in generated COBOL

The following differences occur for generated COBOL code:

- EGL generated COBOL text and basic programs are fully compatible with VisualAge Generator programs. You do not have to regenerate or recompile a VAGen program for either of the following situations:
  - A VAGen program other than a web transaction uses CALL, DXFR, or XFER as a way of transferring to an EGL text or basic program.
  - An EGL program uses *call, transfer to program, transfer to transaction*, or *show* as a way of transferring to a VAGen program.

  The restrictions on calling or transferring between EGL and VAGen programs are similar to those for calling or transferring between two VAGen programs. For example, a VAGen called program cannot use the DXFR or XFER statements to transfer to other programs. Similarly, an EGL called program cannot use *transfer to program, transfer to transaction*, or *show* to transfer to other programs.
- Differences for maps are as follows:
  - The *isDecimalDigit* property is only supported for character fields. It is implemented as the numeric hardware attribute. Numeric fields have a software edit. See “Map fields and the numeric hardware attribute” on page 58 for details.
  - The following device sizes are no longer supported: 6x40, 12x40, 16x64, and 255x60. See “Map groups, maps, and device sizes” on page 53 for details.

Differences in generated Java

The following differences occur for generated Java code:

- EGL generated Java text and basic programs are fully compatible with VisualAge Generator programs. You do not have to regenerate a VAGen program for either of the following situations:

- A VAGen program other than a web transaction uses CALL, DXFR, or XFER as a way of transferring to an EGL text or basic program.
- An EGL program uses *call*, *transfer to program*, *transfer to transaction*, or *show* as a way of transferring to a VAGen program.

The restrictions on calling or transferring between EGL and VAGen programs are similar to those for calling or transferring between two VAGen programs. For example, a VAGen called program cannot use the DXFR or XFER statements to transfer to other programs. Similarly, an EGL called program cannot use *transfer to program*, *transfer to transaction*, or *show* to transfer to other programs.

Differences between distributed CICS and native workstation environments

To run generated EGL code in a distributed environment, you must change to run as a native process instead of having the option to run under Transaction Series (TX Series or CICS). The following list outlines the differences or changes that are necessary to move from a CICS environment to a native environment. The list uses VAGen terminology, but you must make the changes in the corresponding EGL language elements. Refer to Appendix B, "Relationship of VisualAge Generator and EGL Language Elements," on page 177 to determine the corresponding EGL language element.

- General differences are as follows:
  - There is a change from C++ generation to Java generation. Be sure to review the section on "Differences between generated C++ and generated Java" on page 169.
  - Be sure to test performance and scalability when migrating from CICS to native environments.
  - Communication protocols are different between CICS and native environments. You must determine which protocol you plan to use and then change your EGL linkage options parts and resource associations parts accordingly.
- The following CICS-specific special function words and service routines are not supported in native environments:
  - AUDIT (EGL sysLib.audit) for writing a CICS journal entry. You can create your own non-EGL program named AUDIT to write similar information to a file for the native environment.
  - EZEPURGE (EGL sysLib.purge) for deleting a temporary storage queue. You must remove references to sysLib.purge. Alternatively, you can check sysVar.systemType and only use sysLib.purge when you are running in the ZOSCICS environment. If you use this technique, be sure to include the EGL build descriptor option eliminateSystemDependentCode="YES"
  - EZELOC (EGL sysLib.remoteSystemID) for setting the location of a remote file, remote program, or the location at which a remote transaction is to be started using CREATX (EGL sysLib.startTransaction).
- CICS-specific resource associations are not supported in native environments. You must change your resource associations part to use options that are supported for EGL native environments. The following are CICS-specific resource associations that are not supported for generation for a native environment:
  - CICS spool file.
  - Transient data queue, including transient data queue with a trigger level of 1.
  - Temporary storage queue.

  - Local VSAM files, except for the AIX environment.
- The following CICS-supplied features are not supported in native environments:
  - Security services.
  - Database connection and retention.
  - CICS file management, including the use of recoverable files.
  - True segmentation support.
  - Program management.
- Differences when transferring between programs are as follows:
  - For main programs other than web transactions, the XFER statement in CICS transfers to the next transaction ID. For native environments, the XFER statement transfers to a program name. Therefore, all the EGL *transfer to transaction* and *show* statements must be modified to specify the program name.
  - XFER or DXFR to non-VAGen programs is supported in the CICS environment. For native environments, *transfer to program, transfer to transaction*, and *show* are not supported to non-EGL programs.
- Commit and rollback differences are as follows:
  - CICS supports a two-phase commit. Native environments support only a single-phase commit.
  - Files can be defined to CICS as recoverable files. This is not possible for native environments.
  - Message queues are committed or rolled back at the same time as other resources in a CICS environment. Native environments support only a single-phase commit so the message queues might not be committed or rolled back simultaneously with SQL resources if a problem occurs during commit or rollback.
- CALL CREATX (EGL sysLib.startTransaction) differences are as follows:
  - CALL CREATX starts another transaction in CICS and honors the parameters *prid* and *recip*. EGL sysLib.startTransaction starts another program for a native environment and ignores the parameters *prid* and *recip*. As a minimum, you must change sysLib.startTransaction to specify a program name if you generate for a native environment.
  - CICS supports both local and remote CALL CREATX. EGL sysLib.startTransaction only supports starting a local program.
- SQL connection services using EZECONCT (EGL sysLib.connectionService). In VisualAge Generator, for the CICS environment, EZECONCT ignores the password. In EGL, for native environments, sysLib.connectionService uses the password. See “Differences between generated C++ and generated Java” on page 169 for aditional differences due to changing to Java generation.
- EZE special data word differences are as follows:
  - EZEAPP (EGL sysVar.transferName). In VisualAge Generator, for the CICS environment, when EZEAPP is used with an XFER statement, EZEAPP contains the name of the new transaction to be started. In EGL, for native environments, when sysVar.transferName is used with a transfer to transaction or show statement, sysVar.transferName contains the name of the new program to be started.
  - EZEDEST (EGL sysVar.resourceAssociation). In VisualAge Generator, for the CICS environment, EZEDEST contains the system resource name associated with a record while the program is running. In EGL, for native environments, sysVar.resourceAssociation also contains the system resource name associated with a record. However, the format of the information varies depending on

the system and the file type. Therefore, because you are changing both the runtime environment and the file type, you must review any use of sysVar.resourceAssociation to ensure that the information provided by your program is correct for your native environment and file type.

– EZEDESTP (EGL sysVar.printerAssociation). In VisualAge Generator, for the CICS environment, EZEDESTP contains the destination associated with the print file. In EGL, for native environments, sysVar.printerAssociation also contains the file name associated with the print file. However, the format of the information varies depending on the system and the file type. Therefore, because you are changing both the runtime environment and the file type, you must review any use of sysVar.printerAssociation to ensure that the information provided by your program is correct for your native environment and file type.
– EZELTERM (EGL sysVar.terminalID). In VisualAge Generator, for the CICS environment, EZELTERM contains the CICS terminal identifier and is equivalent to EZEUSR. In EGL, for native environments, sysVar.terminalID is initialized from the Java Virtual Machine system property user.name. If this property cannot be retrieved, sysVar.terminalID contains blanks. sysVar.terminalID is not equivalent to sysVar.sessionID (which is the EGL replacement for EZEUSR).
– EZERCODE (EGL sysVar.returnCode). In VisualAge Generator, for the CICS environment, the value in EZERCODE is not passed back to the system or calling program. In EGL, for native environments, the value in EZERCODE is ignored.
– EZERT8 (EGL sysVar.errorCode). In VisualAge Generator, for the CICS environment, EZERT8 is in one of two forms:
  - *RSnnnnnn*, where *nnnnnn* is a VAGen return code based on file access and the problem that occurred.
  - *nnnnnnnn*, where the first two characters are the hexadecimal representation of the first byte of the EIBFN from the CICS EXEC interface block. The remaining 6 characters contain the hexadecimal representation of bytes 0-2 of the EIBRCODE from the CICS EXEC interface block.

  In EGL, for native environments, the return code information varies based on the file type. You should review your use of sysVar.errorCode to ensure that the values you are checking are correct for your environment and file type.
– EZESEGTR (EGL sysVar.transactionID). In VisualAge Generator, for the CICS environment, EZESEGTR is initialized to the current transaction ID and also used to set a new transaction ID to take effect after a CONVERSE. EZESEGTR can be used to control program logic. In EGL, for native environments, EZESEGTR is ignored and cannot be used to control program logic.
– EZEUSR (EGL sysVar.sessionID). In VisualAge Generator for the CICS environment, EZEUSR contains the CICS terminal identifier and is equivalent to EZELTERM. In EGL, for native environments, sysVar.sessionID is initialized from the Java Virtual Machine system property user.name. If this property cannot be retrieved, sysVar.userID contains blanks.
– EZEUSRID (EGL sysVar.userID). In VisualAge Generator, for the CICS environment, EZEUSRID contains the CICS user ID if the user is signed on to the system; otherwise it contains blanks. In EGL, for native environments, sysVar.userID is initialized from the Java Virtual Machine system property *user.name*. If this property cannot be retrieved, sysVar.userID contains blanks.

Differences between generated C++ and generated Java

The following differences occur if you change from generated C++ to generated Java:

- Generated Java is not interoperable with VAGen generated C++ programs. An EGL program that is generated for Java cannot transfer to or from a VAGen program that is generated for C++. An EGL program that is generated for Java can call a VAGen called batch program that is generated for C++.
- In VAGen generated C++ programs, binary data is stored in Intel format (reversed byte order). In EGL generated Java programs, binary data is not stored in Intel format. The EGL conversion tables convert binary data that is passed on calls between generated Java and generated C++ programs. However, you must write a program to convert any binary data in files that were written by C++ programs before using the file in a generated Java program.
- For generated Java programs, the transfer and show statements require one of the following:
  - Both programs must be in the same EGL package.
  - The transfer or show statement explicitly specifies the package name and program name (for example: transfer to program mypackage.program1).
  - The transfer or show statement explicitly specifies the program name and the file containing the transfer statement explicitly imports the package that contains the program (for example: import mypackage.* and transfer to program program1).

  You cannot use sysVar.transferName if you need to transfer across package boundaries using either the transfer or show statement.
- General differences are as follows:
  - Resource association is done at runtime when using VisualAge Generator generated C++ code. In EGL, you have the option to specify resource association information at generation time and have it generated into the properties file for you. Set the resource associations build descriptor to point to your resource associations part. Also set the genProperties build descriptor to GLOBAL or PROGRAM. Refer to the online helps for details of these build descriptor options.
- Differences for maps are as follows:
  - Blink is not supported for text forms.
  - The *isDecimalDigit* property is only supported for character fields. It is implemented as a software edit, not as a hardware attribute. Numeric fields also have a software edit. See "Map fields and the numeric hardware attribute" on page 58 for details.
- For indexed records that have an alternate index record defined, the setting for the DUP I/O error value differs from VisualAge Generator. For VisualAge Generator, for a SET record SCAN followed by a SCAN or SCANBACK I/O option, the DUP I/O error value is not set for the SET record SCAN statement. The DUP I/O error value is set for each of the duplicate-keyed records other than the last record retrieved with a duplicate key. For Java generation, a *set* record *position* followed by a *get next* or *get previous* statement results in the *duplicate* I/O error state being set on the *set* record *position* rather than on the first duplicate-keyed record retrieved. The remaining duplicate-keyed records result in the *duplicate* I/O error state being set the same as in VisualAge Generator. The EGL duplicate state is set on all records other than the first and last of the duplicate-keyed records. See the online helps for more information about indexed records and alternate index records and their use with *set* record *position*, *get next* and *get previous*.

- Differences for SQL are as follows:
  - ODBC is not supported in EGL. If you use a SQL database manager other than DB2, you must obtain a JDBC driver for your database manager.
  - JDBC does not support two-phase commit. Therefore, there are the following differences:
    - There are separate calls to the SQL manager and MQ series manager for commit and rollback. Therefore, if a problem occurs, it is possible for resource to commit or rollback without the corresponding commit or rollback for the other resource.
    - EZECONCT (EGL sysLib.connectionService). In VisualAge Generator, the R option for *unit of work* argument changed the connection to another database without ending the current connection. This permitted you to update multiple databases within the same unit of work. In EGL, the R option, as well as the D1C, D2A, D2C, and D2E options for the *unit of work* argument are all treated as though you specified D1E. D1E is a one-phase commit, but does not automatically release the database connection. You must explicitly request the DISC, DCURRENT, or DALL option to disconnect the database. See the online helps for the sysLib.connectionService for details.
  - EZESQISL (EGL sysVar.sqlIsolationLevel). In VisualAge Generator, a value of 1 means you want cursor stability. In EGL a value of 1 means you want serializable transactions.
  - EZESQRRM (EGL sysVar.sqlerrmc) is not supported.
  - EZESQWN6 (EGL sysVar.sqlwarn[7]) is not supported.
  - EZESQLCA (EGL sysVar.sqlca) fields are limited. They do not include values for EZESQRRM and EZESQWN6.
- EZE special data word differences are as follows:
  - EZECONVT (EGL sysVar.callConversionTable). In VisualAge Generator for C++ generation, the conversion table names are in the format ELAxxyyy, where xx indicates the system and yyy indicates the language. In EGL, for Java generation, the conversion table names provided by EGL are in the format CSOBxxxx, where CSO is a fixed prefix, B indicates the byte order of the target system, and xxxx indicates the code page of the target system. Valid values for B are X for Unix systems, I for Intel systems, and E for EBCDIC systems. EGL automatically translates the EGL table names to CSO table names for you so you do not need to change any code. However, EGL does not provide the ability for you to create your own CSO conversion table.
  - EZERCODE (EGL sysVar.returnCode). In VisualAge Generator, for C++ generation, EZERCODE is passed back to the system or calling program. If the program ends abnormally, a VAGen return code is passed back rather than the value in EZERCODE. In EGL, for Java generation, EZERCODE is ignored.

Part 7. Appendixes

Appendix A. Reserved words

EGL reserved words

There are a large number of reserved words in the EGL. The reserved words cannot be used as part names. The migration tool renames functions, data items, records, and maps if the part name is an EGL reserved word. The migration tool does not rename tables, map groups, or programs. The EGL reserved words are as follows:

Table 60. EGL reserved words

Letter	Reserved words
A	absolute, add, all, as
B	bigInt, bin, blob, by, byName, byPosition
C	call, case, char, clob, close, continue, converse, current
D	dataItem, dataTable, date, dbChar, decimal, decrement, delete, display, dliCall
E	else, embed, end, escape, execute, exit, externallyDefined
F	field, first, float, for, forEach, form, formGroup, forUpdate, forward, freeSql, from, function
G	get, goto
H	hex, hold
I	if, import, in, inOut, insert, int, interval, into, is, isa
L	label, languageBundle, last, library, like
M	matches, mathLib, mbChar, money, move
N	next, noRefresh, not, nullable, num, number, numc
O	onException, open, openUI, otherwise, out
P	pacf, package, pageHandler, passing, prepare, previous, print, private, program, psb
R	record, ref, relative, replace, report, return, returning, returns
S	scroll, self, set, show, singleRow, smallFloat, smallInt, sql, stack, string, strLib, sysLib, sysVar
T	this, time, timeStamp, to, transaction, transfer, try, type
U	unicode, update, url, use, using, usingKeys
W	when, while, with, withinParent

Note: EGL part names cannot start with EZE or the # symbol.

SQL reserved words

There are a large number of SQL reserved words that cannot be used in SQL clauses. The migration tool renames functions, data items, records, and maps if the part name is an SQL reserved word. The migration tool does not rename tables, map groups, or programs. The SQL reserved words are as follows:

Letter	Reserved words
A	absolute, action, add, alias, all, allocate, alter, and, any, are, as, asc, assertion, at, authorization, avg
B	begin, between, bigint, binaryLargeObject, bit, bit_length, blob, boolean, both, by

Letter	Reserved words
C	call, cascade, cascaded, case, cast, catalog, char, char_length, character, character_length, characterLargeObject, characterVarying, charLargeObject, charVarying, check, clob, close, coalesce, collate, collation, column, comment, commit, connect, connection, constraint, constraints, continue, convert, copy, corresponding, count, create, cross, current, current_date, current_time, current_timestamp, current_user, cursor
D	data, database, date, dateTime, day, deallocate, dec, decimal, declare, default, deferrable, deferred, delete, desc, describe, diagnostics, disconnect, distinct, domain, double, doublePrecision, drop
E	else, end, endExec, escape, except, exception, exec, execute, exists, explain, external, extract
F	false, fetch, first, float, for, foreign, found, from, full
G	get, getCurrentConnection, global, go, goto, grant, group
H	having, hour
I	identity, image, immediate, in, index, indicator, initially, inner, input, insensitive, insert, int, integer, intersect, into, is, isolation
J	join
K	key
L	language, last, leading, left, level, like, local, long, longint, lower, ltrim
M	match, max, min, minute, module, month
N	national, nationalCharacter, nationalCharacterLargeObject, nationalCharacterVarying, nationalCharLargeObject, nationalCharVarying, natural, nchar, ncharVarying, nclob, next, no, not, null, nullIf, number, numeric
O	octet_length, of, on, only, open, option, or, order, outer, output, overlaps
P	pad, partial, position, prepare, preserve, primary, prior, privileges, procedure, public
R	raw, read, real, references, relative, restrict, revoke, right, rollback, rows, rtrim, runtimeStatistics
S	schema, scroll, second, section, select, session, session_user, set, signal, size, smallint, some, space, sql, sqlcode, sqlerror, sqlstate, substr, substring, sum, system_user
T	table, tablespace, temporary, terminate, then, time, timestamp, timezone_hour, timezone_minute, tinyint, to, trailing, transaction, translate, translation, trim, true
U	uncatalog, union, unique, unknown, update, upper, usage, user, using
V	values, varbinary, varchar, varchar2, varying, view
W	when, whenever, where, with, work, write
Y	year
Z	zone

SQL reserved words requiring special treatment

The following SQL reserved words require special treatment in EGL if they are used as SQL table names or column names:

```
call, from, group, having, insert, order, select, set, union,
update, values, where
```

To use these SQL reserved words, do the following:

- To specify the column property for an item in an sqlRecord, specify:

  ```
  column = "\"reservedWord\""
  ```

  For example:

  ```
  column = "\"FROM\""
  ```

- To specify the tableNames property for an sqlRecord, specify:

```
tableNames = "\"reservedWord2\""
```

For example:

```
tableNames = "\"ORDER\""
```

- To use one of the reserved words as an SQL column name in the defaultSelectionCondition for a record or in an SQL I/O statement, specify:

```
...   #sql{ select "reservedWord"  from "reservedWord2" ....  }  ...
```

For example:

```
...   #sql{ select "FROM"  from "ORDER" ....  }  ...
```

Java reserved words

Java has reserved words that cannot be used for the package names. If you are generating Java, you may want to avoid using these names:

Letter	Reserved word
A	abstract
B	boolean, break, byte
C	case, catch, char, class, const, continue
D	default, do, double
E	else, extends
F	false, final, finally, float, for
G	goto
I	if, implements, import, instanceof, int, interface
L	long
N	native, new, null
P	package, private, protected, public
R	return
S	short, static, super, switch, synchronized
T	this, throw, throws, transient, true
V	void, volatile
W	while

Appendix B. Relationship of VisualAge Generator and EGL Language Elements

The tables in this appendix have 3 columns:

- VisualAge Generator 4.5 -- this column shows the VAGen language element. In the sections related to part type, the organization of the tables and the terminology used correspond to the VAGen user interface. The tables for statements, EZE words, and service routines are organized based on the type of statement, EZE word, or service routine.
- EGL produced by the migration tool -- this column shows the corresponding EGL language element. This column only shows the information needed for migration and is not intended to be the complete EGL syntax. Additional properties, values, and options might be available for certain EGL language elements. For example, the EGL set statement provides additional options that are not available in VisualAge Generator. The only set statement options listed in these tables are the ones that correspond to VAGen language elements. Use this column as a guide for finding more detailed information in the EGL documentation.
- Migration tool considerations -- this column contains additional information about how the migration tool handles the conversion from VisualAge Generator to EGL.It also provides references to the sections on ambiguous situations, where necessary, to provide details about migration with and without the associated part and the potential problems that can occur when migrating the VAGen language element.

For each part type, the first row in the first table of the section provides:

- VisualAge Generator 4.5 - an overview of the information you can specify in various windows for the part type in VisualAge Generator.
- EGL — the overall EGL syntax for the corresponding EGL part type, using the syntax that the migration tool uses. Other variations of the syntax might be possible. For example, when migrating a VAGen table, the migration tool always places the table contents after the table structure so that is the syntax shown in the Tables section. EGL syntax also permits the table contents to be placed before the table structure.

The following syntax is used in the tables:

- | - choice of a few options. The order of the choices is the same in both the VisualAge Generator 4.5 and the EGL columns.
- bullet list - choice of a longer list of options or values. The order of the choices is the same in both the VisualAge Generator 4.5 and the EGL columns.
- *italics* - values that the migration tool fills in when migrating from VisualAge Generator or that you fill in when writing new EGL statements.
- **bold** - EGL key words and symbols that must be specified as shown.
- { } - encloses information that can be repeated 0 to n times.
- **{ }** - encloses an EGL property list; properties are always separated by commas.
- [ ] - encloses optional information.

General syntax conventions

There are some differences in the overall syntax of VisualAge Generator and EGL.

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
Comments are specified in the following formats: • Prologs for programs, tables, and records. • Descriptions for items and functions. • Comments within functions are indicated by: – a semicolon (;). Everything on the same line after the semicolon is treated as a comment. – /*. Everything on the same line after the /* is treated as a comment.	Comments are specified in the following formats: • // indicates a line comment. Everything on the same line after the // is treated as a comment. The comment is only for one line. • /* comment */. Everything after the /* is treated as a comment until the next */. The comment can span multiple lines.	The migration tool converts as follows: • Prologs and part descriptions are converted to EGL // line comments. • Comments within functions are converted to /* comment */
Decimal point can be either a period or a comma depending on your locale.	Decimal point during development is always the period. Generation and runtime use either a period or a comma depending on the runtime locale.	During migration, if your locale uses the comma for the decimal point, the migration tool converts the comma to a period.
Properties are entered in specialized editors using check boxes, drop down lists, and so on.	Properties are entered in a text editor and must be separated by a comma.	No special considerations.

Data items

The data items section is organized into the following tables:

- Data items - general syntax, data type, length, decimals, and description, Table 61 on page 179
- Default map properties and User Interface properties - general information, Table 62 on page 181
- Default map properties and User Interface properties - general edits, Table 63 on page 181
- Default map properties and User Interface properties - numeric edits, Table 64 on page 183
- Default map properties and User Interface properties - error messages, Table 65 on page 184
- User Interface properties - label and help, Table 66 on page 184

Note: There is only one set of edit and message properties for a data item. Even though this release of EGL does not support UI records, the migration tool merges the map and UI properties for the data items. The migration tool also converts the label and help UI properties to their EGL equivalent properties. This preserves as much of your data item information as possible. You can use this information if you develop any new EGL forms or page handlers.

Table 61. Data items — general syntax, data type, length, decimals, and description

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations

VAGen data item part:
• itemName
• Basic information:
– Data type
– Length
– Decimals
– Description
• Default Map Properties
• User Interface (UI) Properties
	EGL syntax example:
	`// Description`
	`DataItem itemName`
	`dataType(lengthInformation)`
	`{ [ { formattingProperties } ]`
	`[ { validationProperties } ]`
	`[ { pageItemProperties } ]`
	`}`
	`end`
		The migration tool uses the VAGen data type, length, and decimals to determine the EGL dataType and lengthInformation.
		The migration tool merges the VAGen default map properties and the UI properties into the single set of EGL formatting, validation, and pageItem properties.

Character item types: • Char • Hex • DBCS • Mixed • Unicode (VisualAge for Java only) Length is the number of characters. In the record editor you can also show the number of bytes.	Corresponding character item types: • char • hex • dbchar • mbchar • unicode Length is the number of characters.	The migration tool converts character data items to the corresponding type and length.
Numeric character (zoned decimal) types: • Num • Numc Length is the total number of digits, with a maximum of 18. Decimals is the number of digits to the right of the decimal point. In the record editor, you can also show the number of bytes.	Corresponding numeric types: • num • numc Precision is the total number of digits, with a maximum of 18. Scale is the number of digits to the right of the decimal point.	The migration tool converts to the corresponding type, precision, and scale. The migration tool omits the scale if decimals is 0.

Table 61. Data items — general syntax, data type, length, decimals, and description (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
Packed decimal types: • Pacf • Pack Length is the total number of digits, with a maximum of 18. Decimals is the number of digits to the right of the decimal point. The length for Pacf must be odd or 18. The length for Pack can be odd or even. Except for a length of 18, even lengths are recorded within the data item definition, but are treated as the next higher odd length for test, generation, and in the Data Item and Record editors. Only the SQL Record editor shows the even lengths and only SQL records support even length for test and generation. The even length is only used in SQL *where* clauses and in SQL functions that use execution time statement build. In the record editor you can also show the number of bytes.	Corresponding numeric types: • pacf • decimal Precision is the total number of digits, with a maximum of 18. Scale is the number of digits to the right of the decimal point. The length for pacf must be odd or 18. The length for decimal can be odd or even. Even lengths are supported for data item definitions and all record types. At test and generation, if you use VisualAge Generator Compatibility mode, EGL does the following for decimal items with even precision: • Increases the precision by one in all records. • EGL uses a temporary variable with the even precision in SQL *where* clauses or *prepare* statements.	The migration tool converts to the corresponding type, precision, and scale. The migration tool omits the scale if decimals is 0. For a Pack item, if an even length was recorded in the data item definition, the migration tool migrates it as the even length.
Binary item types: • Bin, length 4, no decimals • Bin, length 9, no decimals • Bin, length 18, no decimals • Bin, length 4, 9, or 18 with decimals	Corresponding binary types: • smallint (no precision or scale) • int (no precision or scale) • bigint (no precision or scale) • bin with precision and scale	The migration tool converts binary data items to the corresponding type based on the length and number of decimals. The bin type is only used if decimals (scale) is specified.
Description	Not applicable.	The migration tool converts the item description to a comment that precedes the DataItem definition.

Table 62. Default map properties and User Interface properties - general information

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
Data items can have both default map properties and user interface (UI) properties specified. The properties include the following: • formatting edits • validation edits • error messages UI properties also include a label and help text. Explicitly setting some properties in VisualAge Generator automatically causes other properties to be set. For example, setting numeric separator also explicitly sets fill character, input required, justify, currency symbol, and sign.	Data items can have the following properties: • formatting properties • validation properties • page item properties The categories for some properties are changed from VisualAge Generator. For example, error messages are grouped with the validation properties. Page item properties include the UI label and help text. The EGL column in the following tables shows the category for the EGL property.	The migration tool merges the default map properties and UI properties, giving precedence to the UI properties. Validation edits and their associated error messages are migrated as a pair. The migration tool only migrates properties that were explicitly set in VisualAge Generator. The tool does not automatically insert default values for EGL properties. See information about Merging map and UI edits in “Shared edits and messages” on page 44 for details and potential problems. Also see information about map item edits for shared data items in “Map item edit routine for shared data items” on page 45 for details and potential problems.

Table 63. Default map properties and User Interface properties - general edits

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
Edit type (UI only) - values: • None • Boolean • Date • Time	EGL supports multiple properties: • not applicable • boolean = yes • dateFormat = locale • timeFormat = ″hh:mm:ss″ (formatting properties)	No special considerations.
Edit function (UI only)	validator (validation property)	No special considerations.
Edit table (UI only)	validatorTable (validation property)	No special considerations.
Run edit function on web (UI only)	Not supported.	The migration tool converts to a comment. The comment includes the EGL runValidatorFromProgram property, which is expected to be the eventual replacement. The EGL property is the reverse of the VAGen property. The migration tool converts *yes* to *no* and *no* to *yes* when creating the comment.

Table 63. Default map properties and User Interface properties - general edits (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
Edit routine (map only)	validator OR validatorTable (validation property)	If the UI edit function and edit table are not specified, the migration tool does the following: • Sets the validator property if the map edit routine is EZEC10 or EZEC11. • Sets the validator property if the edit routine is a function. • Sets the validatorTable property if the edit routine is a table. If the UI edit function or edit table are specified, the migration tool does not migrate the map Edit routine. Special considerations apply if the edit routine is not available during migration. See information about map edit routines in "Map item edit routine for shared data items" on page 45 for additional details and potential problems.
Justify - Left | Right | None (map only) **Note:** • For map items, the default is right for numeric fields and left for all other fields. • For UI items, justify is not supported.	align = left | right | none (formatting property) **Note:** • For form fields, the default is right for numeric fields and left for all other fields • For page items, align is not supported.	No special considerations.
Date edit mask (map only) Valid values are as follows: • SYSGREGRN • SYSJULIAN • dateEditPattern	dateFormat = *value* Valid values are as follows: • systemGregorian • systemJulian • "dateEditPattern" (formatting property)	If the UI edit type does not specify Date, the migration tool sets the dateFormat based on the Date edit mask specified in VisualAge Generator, if any. If the UI edit type specifies Date, the migration tool does not migrate the map Date edit mask.
Minimum input	minimumInput (validation property)	No special considerations.

Table 63. Default map properties and User Interface properties - general edits (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
Fill character **Note:** • The default fill character for items used in a UI record is blank for character, MIXED, and numeric fields. The default fill character is zero for hex fields. Blank is the required fill character for DBCS and Unicode fields. Null is not a valid fill character. • The default fill character for items used on a map is null for character, DBCS, or MIXED fields. The default fill character is blank for numeric fields and zero for hex fields.	fillCharacter (formatting property) **Note:** The same default fillCharacter is used for both page items and form fields unless overridden in the specific page or form.	Special considerations apply because there is only one default fill character in EGL. See information about ambiguous data items and fill characters in "Fill characters for shared data items" on page 46 for details and potential problems.
Fold	upperCase (formatting property)	No special considerations.
Hex edit (map only)	isHexDigit (validation property)	No special considerations.
Input required	inputRequired (validation property)	No special considerations.
Check SO/SI space	needsSOSI (validation property)	No special considerations.

Table 64. Default map properties and User Interface properties - numeric edits

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
Minimum value and Maximum value **Note:** If either Minimum value or Maximum value is specified, both must be specified.	range = (*minimumValue, maximumValue*) (validation property)	The migration tool combines the minimum and maximum value into the EGL range property.
Sign - None | Leading | Trailing **Note:** The default value is None.	sign = none | leading | trailing (formatting property) **Note:** The default value is none.	No special considerations.
Currency (both map and UI) Currency symbol (UI only)	`currency = yes | no | "currencySymbol"` (formatting property) **Note:** • The currencySymbol also applies to forms. • If currency = yes, the actual currency symbol used at runtime is set the same way it is in VisualAge Generator.	The migration tool migrates the first of the following that applies: • If the UI Currency symbol is specified, the tool migrates to currency = "currencySymbol". • If the UI Currency edit is set to yes or no, the tool sets the currency property to yes or no, respectively. • If the map Currency edit is set to yes or no, the tool sets the currency property to yes or no, respectively.
Separator	numericSeparator (formatting property)	No special considerations.
Zero edit	zeroFormat (formatting property)	No special considerations.

Table 65. Default map properties and User Interface properties - error messages

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
Edit table (UI only)	validatorTableMsgKey (validation property)	No special considerations.
EZE function (UI only)	validatorMsgKey (validation property)	No special considerations.
Edit routine (map only)	validatorTableMsgKey OR validatorMsgKey (validation properties)	The migration tool only migrates the map edit routine message if the UI edit table, UI edit function, UI edit table message and UI EZE function messages are not specified. If the migration tool migrates the map edit routine message, the tool does the following: • Sets validatorMsgKey if the edit routine is EZEC10 or EZEC11. • Sets validatorTableMsgKey if the edit routine is a table. • Does not migrate the edit routine message if the edit routine is a function because the message is not used in this situation in VisualAge Generator. Special considerations apply. See information about ambiguous data items and map edit routines in "Map item edit routine for shared data items" on page 45 for additional details and potential problems.
Minimum input	minimumInputMsgKey (validation property)	No special considerations.
Input required	inputRequiredMsgKey (validation property)	No special considerations.
Data type	typeChkMsgKey (validation property)	No special considerations.
Numeric range	rangeMsgKey (validation property)	No special considerations.

Table 66. User Interface properties - label and help

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
UI label	displayName (page item property)	No special considerations.
Help text	help (page item property)	No special considerations.

Records

The records section is organized into the following tables:

Table 67. Records - general syntax, record type, properties, and prolog

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations

VAGen record part:
• recordName
• Basic information
– Record type
– Record structure (item list)
• Properties (vary based on record type)
• Prolog
 	
The record structure can be given by specifying an alternate specification record or by including the item list.
	EGL record example:
	`//*** Record=recordName***`
	`// prolog`
	`//******************`
	`Record recordName`
	`type recordType`
	`{ [ recordProperties ] }`
	`recordStructure`
	`end // end recordName`
	
	Note: The record structure can be given by specifying an embed statement or by including the item list.
		No special considerations.

Record types: • Working Storage • Redefined • Serial • Indexed • Relative • Message Queue • SQL Row • User Interface • DL/I Segment	EGL Record types: • basicRecord • basicRecord • serialRecord • indexedRecord • relativeRecord • mqRecord • sqlRecord • Not supported in this release. • Not supported in this release.	The migration tool migrates a redefined record to a basicRecord. The tool includes a comment with the record definition to provide the name of the record that was redefined. Special considerations apply for redefined records. See the information in "Redefined records" on page 47 for details and potential problems.
Working storage record properties: • Alternate specification	basicRecord properties: • embed statement	The migration tool migrates an alternate specification to the embed statement.
Redefined record properties: • Redefinition **Note:** The Redefinition property specifies the name of another record that provides the physical storage. The current record provides a different data item layout of the same physical storage.	basicRecord properties: • Not applicable. Redefinition information is only specified in programs that use the record. The same record can be used as a redefinition of another record or as a normal record.	The migration tool includes a comment with the record definition to provide the name of the record that was redefined. The migration tool also includes the redefines property on the declaration statement for the record in programs that use the record. Special considerations apply depending on how the record is used in the program and on whether the record is available during migration. See the information in "Redefined records" on page 47 for details and potential problems.
Serial record properties: • File name • Alternate specification • Variable length item • Occurrences item	serialRecord properties: • fileName • embed statement • lengthItem • numElementsItem	The migration tool migrates an alternate specification to the embed statement.

Table 67. Records - general syntax, record type, properties, and prolog (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
Indexed record properties: • File name • Record ID • Alternate specification • Variable length item • Occurrences item	indexedRecord properties: • fileName • keyItem • embed statement • lengthItem • numElementsItem	The migration tool migrates an alternate specification to the embed statement.
Relative record properties: • File name • Record ID • Alternate specification	relativeRecord properties: • fileName • keyItem • embed statement	The migration tool migrates an alternate specification to the embed statement.
Message Queue record properties: • File name • Alternate specification • Include message in transaction • Open queue for exclusive use on input • Record length item • Occurrences item • Queue descriptor record • Open options record • Message descriptor record • Get options record • Put options record	mqRecord properties: • fileName • embed statement • includeMsgInTransaction • openQueueExclusive • lengthItem • numElementsItem • queueDescriptor • openOptions • msgDescriptor • getOptions • putOptions	The migration tool migrates an alternate specification to the embed statement.
SQL row record properties: • See Table 69 on page 188.	SQL row record properties: • See Table 69 on page 188.	No special considerations.
Prolog	Not applicable.	The migration tool converts the prolog to a comment that precedes the record definition.

Table 68. Records - record structure for most record types

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
Record structure - variation 1: Alternate specification. If RecordA specifies an alternate specification of RecordB, RecordB provides all the items for RecordA. There is no item structure in RecordA. If RecordB contains level 77 items, RecordA only contains the non-level 77 items from RecordB.	Record structure - variation 1: The EGL embed statement specifies the record that provides the item structure for the current record. RecordA embeds RecordB. For example: `embed RecordB;`	The migration tool migrates an alternate specification to the embed statement. Special considerations apply for level 77 items in working storage records. See information in "Level 77 items in records" on page 48 for details and potential problems.

Table 68. Records - record structure for most record types (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations

Record structure - variation 2 with Shared Items:
• itemName
• Occurs
• Shared
• levelNumber
Note: levelNumber is hidden, but it is based on the data item hierarchy within the record.
Note: Type, Length, Decimals and Description are visible in the record editor, but are not stored in the record.
	Record structure - variation 2 with EGL type definitions example:
	`levelNumber itemname`
	`  itemname [occurs];`
	Note: Note: Type, Length, Decimals and Description are **not** visible in the editor.
		The migration tool migrates shared items to an EGL variable that is defined using a type definition. For migration, the type definition is always the same as the item name.
		The migration tool omits the occurs information if occurs is 1.
		Special considerations apply for level 77 items in working storage records. See information in "Level 77 items in records" on page 48 for details and potential problems.

Record structure - variation 2 with Nonshared Items:
• itemName
• Occurs
• Type
• Length
• Decimals
• Nonshared
• Description
• levelNumber is hidden, but is based on the data item hierarchy within the record
Note: Type, Length, Decimals and Description are stored with the item in the record.
	Record structure - variation 2 with EGL primitive types example:
	`levelNumber itemName`
	`  dataType(lengthInformation)`
	`  [occurs];`
	`  // Description`
	Note: Type, Length, Decimals and Description are visible in the editor.
		The migration tool migrates nonshared items to an EGL variable that is defined using a primitive type. Migration of type, length, and decimals information is the same as described in Table 61 on page 179, Data items - general syntax, data type, length, decimals, and description.
		The migration tool omits the occurs information if occurs is 1.
		Special considerations apply for level 77 items in working storage records. See information in "Level 77 items in records" on page 48 for details and potential problems.

Table 69. Records - SQL properties and SQL record structure

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
SQL record properties: • Default key item • Alternate specification • SQL tables: – Label – Name **Note:** • If a record does not specify an alternate specification, the key items are the items in the record structure that specify key=yes. The Default key item is ignored. • If a record specifies an alternate specification, the key items are the Default key item in the current record merged with the items in the alternate specification record that specify key=yes. The keys are merged in the order in which the items appear in the record structure. If the Default key item in the current record is also specified as key=yes in the alternate specification record, the item is only included once in the merged list of keys. • SQL table names can be an actual table name (normal situation) or a table name host variable that will be substituted at run time. Table name host variables start with a semicolon (:).	sqlRecord properties: • keyItems • embed statement • tableNames and / or tableNameVariables **Note:** • keyItems is a list of all keys for the record. key=yes is not specified for items in the record structure. • The tableNames property is a list of the table names and table labels when the table name is not a host variable. The tableNameVariables property is a list of the table names and table labels when the table name is a host variable that will be substituted at run time. The table names in the tableNameVariables property do not start with a semicolon. tableNames and tableNameVariables can both be used in the same record definition.	The migration tool builds the keyItems property as follows: • If the VAGen alternate specification is not included, the tool uses any items from the record structure that specify key=yes, but does not include the VAGen default key item. • If the VAGen alternate specification is included, the tool merges any items from the alternate specification record that specify key=yes and the default key item from the current record. The keys are listed in the same order as the items appear in the record structure. If the default key item from the current record is the same as one of the key items from the alternate specification record, the item is only included once in the keyItems list. The migration tool builds the lists for tableNames and tableNameVariables as follows: • tableNames is built from the table names and table labels when the table name is not a host variable. • tableNameVariables is built from the table names and table labels when the table name is a host variable. Special considerations apply. See information about SQL alternate specification in “Alternate specification records” on page 49 for details and potential problems.

Table 69. Records - SQL properties and SQL record structure (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations

SQL Default Conditions:
• whereClauseText
Note:
• The SQL default conditions enable you to specify a where clause, most typically for join conditions when multiple tables are used in the SQL row record. The syntax is SQL syntax.
• !itemColumnName variables are permitted. These variables specify the name of an item in the SQL row record. At test or generation time, VisualAge Generator substitutes the corresponding SQL column name.
	Example of default selection conditions:
	`defaultSelectCondition =`
	`  #sql{`
	`  whereClauseText`
	`  }`
	Note:
	• The defaultSelectCondition is used for the same purpose as in VisualAge Generator.
	• !itemColumnName variables are not supported. Actual SQL column names must be used.
		The migration tool converts any !itemColumnName variables to their corresponding SQL column name.
		Special considerations apply. See information about SQL alternate specification in "Alternate specification records" on page 49 for details and potential problems.

Record structure - variation 1: Alternate specification. If RecordA specifies an alternate specification of RecordB, RecordB provides all the items for RecordA. There is no item structure in RecordA.	Record structure - variation 1: The EGL embed statement specifies the record that provides the item structure for the current record. RecordA embeds RecordB. For example: `embed RecordB;`	The migration tool migrates an alternate specification to the embed statement.

Table 69. Records - SQL properties and SQL record structure (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
Record structure - variation 2 with Shared Items: • itemName • Read Only • Key • SQL Column Name • SQL Code • Shared **Note:** • Type, Length, Decimals and Description are visible in the record editor, but are not stored in the record. • VisualAge Generator always includes the null indicator variable for SQL items. • The SQL Code is not included in the External Source Format for pack and binary fields. If the SQL Code is not included in the External Source Format for char, dbchar, or unicode fields, the field is treated as a fixed length field. This only occurs for records that were migrated from earlier releases of VisualAge Generator and never modified using VisualAge Generator 4.5.	Record structure - variation 2 with EGL type definitions example: *itemName itemName* **{** [**sqlDataCode=***sqlCodeNumber*] **column="***SQLColumnName***"** [**isReadOnly=yes**] **isNullable = yes** [**sqlVar = yes**] **};** **Note:** • Type, Length, Decimals and Description are **not** visible in the editor. • isNullable = no is supported.	If you selected the Migration Syntax Preference *Convert shared data items to primitive item definitions* and the data item part is available, the migration tool does the following: • Converts the shared item to an EGL variable that is defined using a primitive definition based on the type, length, and decimals specified for the data item part. • Includes the sqlDataCode property for hex items. • Sets the sqlVar=yes property for char, dbchar, or unicode fields if the VAGen SQL data code indicates the item is variable length. The migration tool omits the sqlVar property if the VAGen SQL data code indicates the item is fixed length. If you did not select the Migration Syntax Preference to *Convert shared data items to primitive item definitions* or the data item part is not available, the migration tool does the following: • Converts the shared item to an EGL variable that is defined using a type definition. For migration, the type definition is always the same as the item name. • Includes the sqlDataCode property if it is included in the External Source Format and is not one of the values for VAGen binary or packed fields. • Sets the sqlVar=yes property if the VAGen SQL data code indicates the item is variable length. The migration tool omits the sqlVar property if the VAGen SQL data code indicates the item is fixed length. The migration tool always does the following: • Includes any key=yes items in the EGL keyItems property for the sqlRecord. • Always sets isNullable=yes because VisualAge Generator always includes the null indicator variable. • Only Includes isReadOnly if the value is yes.

Table 69. Records - SQL properties and SQL record structure (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
Record structure - variation 2 with Nonshared Items: • itemName • Type • Length • Decimals • Read Only • Key • SQL Column Name • SQL Code • Nonshared • Description **Note:** • Type, Length, Decimals and Description are stored with the item in the record. • VisualAge Generator always includes the null indicator variable for SQL items. • The SQL Code is not included in the External Source Format for pack and binary fields. If the SQL Code is not included in the External Source Format for char, dbchar, or unicode fields, the field is treated as a fixed length field. This only occurs for records that were migrated from earlier releases of VisualAge Generator and never modified using VisualAge Generator 4.5.	Record structure - variation 2 with EGL primitive types example: *itemName* *dataType(lengthInformation)* *// Description* **{** [**sqlDataCode=***sqlCodeNumber*] **column="***SQLColumnName***"** [**isReadOnly=yes**] **isNullable = yes** [**sqlVar = yes**] **};** **Note:** • Type, Length, Decimals and Description are visible in the editor. • isNullable = no is supported.	The migration tool migrates nonshared items to an EGL variable that is defined using a primitive type. Migration of type, length, and decimals information is the similar to what is described in Table 61 on page 179. The migration tool includes the sqlDataCode property only for hex items. The migration tool sets sqlVar=yes for char, dbchar, and unicode data items if the VAGen SQL data code indicates the item is variable length. The migration tool omits the sqlVar property if the VAGen SQL data code indicates that the item is fixed length. The migration tool includes any key=yes items in the EGL keyItems property for the sqlRecord. The migration tool always sets isNullable=yes because VisualAge Generator always includes the null indicator variable. The migration tool only includes isReadOnly if the value is yes.
VAGen data type - Char • data code - 453 • data code - 449 or 457	EGL data type: • char; omit sqlVar • varchar, sqlVar = yes	No special considerations.
VAGen data type - DBCS • data code - 469 • data code - 465 or 473	EGL data type: • dbchar; omit sqlVar • vardbchar, sqlVar = yes	No special considerations.
VAGen data type - Unicode • data code - 469 • data code - 465 or 473	EGL data type: • unicode; omit sqlVar • varunicode, sqlVar = yes	No special considerations.

Tables

The VAGen tables section is organized into the following tables:

- VAGen tables - general syntax, table type, properties, and prolog, Table 70 on page 192
- VAGen tables - table structure, Table 71 on page 192
- VAGen tables — table contents, Table 72 on page 193

Table 70. Tables — general syntax, table type, properties, and prolog

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations

VAGen table part:
• tableName
• Basic information
– Table type
– Table structure (item list)
• Properties
• Prolog
• Table Contents
	EGL syntax example:
	`//*** DataTable=tableName***`
	`// prolog`
	`//******************`
	`DataTable tableName`
	`type tableType`
	`{ [otherTableProperties]`
	`[alias =`
	`"originalTableName"] }`
	`tableStructure`
	`[{ contents =`
	`[{rowContents}] }]`
	`end // end tableName`
		The migration tool does not rename tables for you even if they conflict with the EGL reserved word list. The migration tool does not set the alias property. If you must rename a table, you can use the alias property to specify the original name of the VAGen table. See information about table names in "Reserved words and table names" on page 51 for details.

Table types: • Unspecified • Match Invalid • Match Valid • Range Match Valid • Message	DataTable types: • basicTable • matchInvalidTable • matchValidTable • rangeChkTable • msgTable	No special considerations.
Properties — Runtime attributes: • Resident • Shared	DataTable properties: • resident • shared	No special considerations.
Properties - Fold table contents	Not applicable. If you want the table contents to be folded, you must enter the contents in upper case.	If the VAGen table specifies that the table contents should be folded, the migration tool ensures that the char, hex, and mixed data in the table contents is converted to upper case.
Prolog	Not applicable.	The migration tool converts the prolog to a comment that precedes the DataTable definition.

Table 71. Tables — Table structure

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations

VAGen Table structure - with Shared Items:
• itemName
• Shared
• levelNumber
Note: levelNumber is hidden, but it is based on the data item hierarchy within the record.
Note: Type, Length, Decimals and Description are visible in the table editor, but are not stored in the table.
	DataTable structure - with EGL type definitions:
	`levelNumber itemName`
	`itemName ;`
	Note: Type, Length, Decimals and Description are **not** visible in the editor.
		The migration tool migrates shared items to an EGL variable that is defined using a type definition. For migration, the type definition is always the same as the item name.

Table 71. Tables — Table structure (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations

VAGen Table structure — with Nonshared Items:
• itemName
• Type
• Length
• Decimals
• Nonshared
• Description
• levelNumber
Note: levelNumber is hidden, but it is based on the data item hierarchy within the table.
Note: Type, length, decimals, and description are stored with the item in the table.
	DataTable structure — with EGL primitive types:
	`levelNumber itemName`
	`  dataType(lengthInformation) ;`
	`  // Description`
	Note: Type, length, decimals and description are visible in the editor.
		The migration tool migrates nonshared items to an EGL variable that is defined using a primitive type. Migration of type, length, and decimals information is the same as described in Table 61 on page 179.

Table 72. Tables — table contents

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations

Table contents:
• Table contents are entered in a formatted editor. Table contents are entered for the top level (parent) items in the table structure.
• Character and hex data is not enclosed in quotes.
	DataTable contents:
	• Each row's contents is enclosed in square brackets. There is an outer set of square brackets that encloses the entire set of rows.
	• Values within the row contents must be separated by commas.
	• Character data including hex data must be enclosed in double-quotes.
	Example:
	`contents = [ [ rowContents ]`
	`  { , [ rowContents ] }`
	`  ]`
	`where`
	`  rowContents = value { , value}`
		If the VAGen table specifies that the table contents should be folded, the migration tool ensures that the char, hex, and mixed data in the table contents is converted to upper case.
		The migration tool also encloses character data, including hex data, in double-quotes.

Map groups

The map groups section is organized into the following tables.

- Map Groups — general information, Table 73 on page 194
- Map Groups — general syntax and floating areas, Table 74 on page 195
- Map Groups — device names, types, and sizes, Table 75 on page 196

Table 73. Map Groups — general information

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
The map group part is only required if there are floating areas. If there is no map group part, VisualAge Generator automatically generates all maps with the same map group name as though the map group part did exist.	The formGroup is required.	The migration tool creates a formGroup part if one does not exist in the migration set.
Map names consist of a map group name and a map name.	The form name does not include the formGroup name. A form can be defined (nested) within a form group. Alternatively, a form can be outside the formGroup part. In this case, the formGroup part must include a use statement to specify the form name and an import statement import the package in which the form located. This technique enables you to have one definition of a common form (for example, a pop-up list form) and make it available in many different form groups.	The migration tool migrates all maps to forms. The tool does not attempt to identify common, identical map definitions across multiple map groups. If you migrate in single file mode, the migration tool includes a use statement for each form within a form group. You should move the forms so that they are nested within their formGroup part. If you migrate using Stage 1 – 3 migration, the migration tool automatically nests all forms within the formGroup part.
When a program specifies a map group, the program can use any map within the map group just by referencing the map name.	When a program includes a *use* statement to indicate which formGroup it is using, the program can reference any map within the formGroup just by referencing the form name.	No special considerations.

Table 74. Map Groups — general syntax and floating areas

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations

The map group part can contain the following:
• Map group name
• Floating area information
– Device name
– Device size
– Size
- Lines
- Columns
– Position
- Starting line
- Starting column
	The formGroup part can contain the following:
	• Form group name
	• Form group properties
	• Screen floating area information
	• Print floating area information
	• Use statements for the forms that are included in the form group.
	
	An example of the format of a form group part is as follows:
	`FormGroup groupName {`
	`[ alias="generationName"]`
	`[ screenFloatingArea`
	`{screenFloatingAreaInformation}]`
	`[printFloatingArea`
	`{printFloatingAreaInformation}]`
	`Form formName type textForm`
	`{formProperties}`
	`[variableFields]`
	`[constantFields]`
	`end // end formName`
	`use formName2;`
	`end // end groupName`
		The migration tool uses the VAGen device type to determine whether the floating area information is for a Display map (screenFloatingArea) or a Printer map (printFloatingArea).
		
		See Table 75 on page 196 about setting deviceType.

Not applicable.	alias	The migration tool does not rename map groups even if they conflict with an EGL reserved word. Special considerations apply. See “Reserved words and formGroup names” on page 51 for details and potential problems.

Table 74. Map Groups — general syntax and floating areas (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations

Floating area information includes the following:
• Device name
• Device size (rows x columns)
• Floating area specification
– Size
- Lines
- Columns
– Position
- Starting line
- Starting column
Note:
• In VisualAge Generator, you define the size and starting position of the floating area.
• Different floating area specifications are permitted, but not recommended, for devices that have the same size.
	Floating area information includes the following:
	• Device size
	• Margin information
	Print floating area information also includes the device type.
	Here is an example of the screen floating area that is used for text forms:
	`screenFloatingArea { screenSize=(lines,columns) topMargin=nn, bottomMargin=nn, leftMargin=nn, rightMargin=nn }`
	Here is an example of the print floating area that is used for print forms:
	`printFloatingArea { deviceType=singleByte, pageSize=(lines,columns) topMargin=nn, bottomMargin=nn, leftMargin=nn, rightMargin=nn }`
	Note: Only one floating area specification is permitted for a screenSize or pageSize.
		The migration tool uses the VAGen device type to determine whether the floating area specification is for display maps (screenFloatingArea) or print maps (printFloatingArea).
		The migration tool computes the margin information as follows:
		• The topMargin is set to the VAGen floatingAreaStartingLine - 1.
		• The bottomMargin is set to the VAGen deviceRows - (floatingAreaStartingLine + floatingAreaLines) + 1.
		• The leftMargin is set to the VAGen floatingAreaStartingColumn - 1.
		• The rightMargin is set to the VAGen deviceColumns - (floatingAreaStartingColumn + floatingAreaColumns) + 1.
		See Table 75 on page 196 for information about setting the deviceType.

Printer type can be one of the following: • Printer • DBCS printer	deviceType=singleByte | doubleByte **Note:** The deviceType property is only specifed for print forms.	The migration tool sets the EGLdeviceType property based on the VAGen printer type.

Table 75. Map Groups — device names, types, and sizes

VisualAge Generator Device Name	Device Size (lines x columns)	Device Type	Migration tool considerations
3643–2	6 x 40	Display	This device size is not supported for COBOL generation.
3277–1	12 x 40	Display	This device size is not supported for COBOL generation.
3643–4	16 x 64	Display	This device size is not supported for COBOL generation.
3278–1, 3278–1B, ANY–1D	12 x 80	Display	No special considerations.
3278–2, 3278–2B, ANY–2D	24 x 80	Display	No special considerations.
3278–3, 3278–3B, ANY-3D	32 x 80	Display	No special considerations.

Table 75. Map Groups — device names, types, and sizes (continued)

VisualAge Generator Device Name	Device Size (lines x columns)	Device Type	Migration tool considerations
3278–4, 3278–4B, ANY-4D	43 x 80	Display	No special considerations.
3278–5, 3278–5B, ANY-5D	27 x 132	Display	No special considerations.
ANY-D (3290 configured as 62x160)	255 x 160	Display	This device size is not supported for COBOL generation.
5550D	24 x 80	DBCS Display	No special considerations.
3767, PRINT-B, PRINTER	255 x 132	Printer	For the printFloatingArea, the EGL deviceType=singleByte
5550P	255 x 158	DBCS Printer	For the printFloatingArea, the EGL deviceType=doubleByte

Maps

The maps section is organized into the following tables.

Table 76. Maps — general information

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
There are two types of maps: • Display maps • Printer maps	There are two types of forms: • Text forms • Print forms	No special considerations.

Table 76. Maps — general information (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
Map names consist of a map group name and a map name.	The form name does not include the formGroup name. A form can be defined (nested) within a form group. Alternatively, a form can be outside the formGroup part. In this case, the formGroup part must include a use statement to specify the form name and an import statement import the package in which the form located. This technique enables you to have one definition of a common form (for example, a pop-up list form) and make it available in many different form groups.	The migration tool migrates all maps to forms. The tool does not attempt to identify common, identical map definitions across multiple map groups. If you migrate in single file mode, the migration tool includes a use statement for each form within a form group. You should move the forms so that they are nested within their formGroup part. If you migrate using Stage 1 – 3 migration, the migration tool automatically nests all forms within the formGroup part.
When a program specifies a map group, the program can use any map within the map group just by referencing the map name.	When a program includes a *use* statement to indicate which formGroup it is using, the program can reference any map within the formGroup just by referencing the form name.	No special considerations.

Table 77. Display maps — general syntax, map type, and properties

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations

Display maps can contain the following:
• Map group name and map name
• Map properties
– General properties
- Help map name
- Help key
- Bypass keys
- Variable field folding
– Layout properties
- Map size
- Starting position
- Floating map
– Devices
- Type (Display or Print)
- Supported devices
• Constant fields
• Variable fields
• Field edit order for variable fields
	Text form parts can contain the following:
	• Form name
	• Form type
	• Form properties
	• Constant fields
	• Variable fields
	• Validation order for variable fields
	
	An example of the format of a text form created by the migration tool is as follows:
	`Form mapName type textForm`
	`{ screenSizes=(sizeList),`
	`formSize=(24,80), position=(1,1),`
	`helpForm=helpFormName,`
	`helpKey=pf1,`
	`validationBypassKeys=(pf3),`
	`msgField=VAGen_EZEMSG }`
	`[ variableFields ]`
	`[ constantFields ]`
	`end // end mapName`
		The migration tool uses the VAGen device type to determine whether the map is a Display map (text form) or a Printer map (print form).
		
		See Table 75 on page 196 for information about determing whether the device is a display or printer.

Help map name	helpForm	No special considerations.
Help key	helpKey	No special considerations.

Table 77. Display maps — general syntax, map type, and properties (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
Bypass keys You can specify a maximum of 5 Bypass keys for a map.	validationBypassKeys You can specify a maximum of 5 validationBypassKeys for a form.	No special considerations.
Variable field folding	Not supported for a form. Each char or mbchar variable field on the form must specify whether the data the user enters is to be automatically converted to upper case.	The migration tool does the following: • If Variable field folding is specified for the entire map, the migration tool includes *upperCase=yes* for every character and mixed field. • If Variable field folding is not specified for the entire map, the migration tool uses the Fold information specified for each character or mixed field to determine whether to set the upperCase property for that field.
Map size — Lines and Columns	size = (Lines, Columns)	No special considerations.
Starting position - Line and Column NEXT,SAME is required if the map ia a floating map.	position = (Line, Column) If the position information is omitted, the form is a floating form	If Floating map is selected, the migration tool omits the position information.
Floating map	Not applicable. If the position information is omitted, the form is a floating form.	If Floating map is selected, the migration tool omits the position information.
Device Type - Display or DBCS Display	type textForm	The migration tool uses the Device Type information to determine whether to migrate the map to a text or print form.

Supported devices
Note: Supported devices shows the device type, number of lines, and number of columns
	`screenSizes = ((Lines, Columns),`
	`(Lines, Columns))`
	
	Note: Include a (Lines, Columns) pair for each screen size that you want to have supported for the form.
		The migration tool uses the device type information to determine the corresponding screenSizes property. If several VAGen devices have the same screen Size, the migration tool only includes the screen size once.
		
		Special considerations apply because not all of the devices supported by VAGen are supported for COBOL generation in EGL. See "Map groups, maps, and device sizes" on page 53 for details.

Not applicable. In VisualAge Generator, the message field is always named EZEMSG.	msgField This is the name of the field that is to contain any EGL error messages.	The migration tool sets the msgField property if EZEMSG is anywhere on the map.

Table 77. Display maps — general syntax, map type, and properties (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
Not applicable.	alias	The migration tool includes the alias property if the map has to be renamed due to a conflict with an EGL reserved word or because the map name starts with the # symbol. The migration tool also includes the alias property for a map in a program's help map group if the map has to be renamed due to a conflict with the name of a map in the program's main map group. Special considerations apply. See "Map names and help map names" on page 54 for details.

Table 78. Printer maps — general syntax, map type, and properties

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations

Printer maps can contain the following:
• Map group name and map name
• Map properties
– General properties
- Help map name
- Help key
- Bypass keys
- Variable field folding
- SO/SI take position
– Layout properties
- Map size
- Starting position
- Floating map
– Devices
- Type (Display or print)
- Supported devices
• Constant fields
• Variable fields
• Field edit order for variable fields
	Print forms can contain the following:
	• Form name
	• Form properties
	• Constant fields
	• Variable fields
	
	An example of the format of a text form created by the migration tool is as follows:
	`Form mapName type printForm`
	`{size=(255,158), position=(1,1),`
	`addSpaceForSOSI=yes }`
	`[ variableFields ]`
	`[ constantFields ]`
	`end // end mapName`
		The migration tool uses the VAGen device type to determine whether the map is a Display map (text form) or a Printer map (print form).
		
		The migration tool always omits the following properties for print forms:
		• General properties
		– Help map name
		– Help key
		– Bypass keys
		– Variable field folding
		• Devices
		– Supported devices
		• Field edit order for variable fields
		
		See Table 74 on page 195 for information about determining whether the device is a display or printer.

Help map name	Not applicable for a print form.	The migration tool omits this property for a print form.
Help key	Not applicable for a print form.	The migration tool omits this property for a print form.
Bypass keys	Not applicable for a print form.	The migration tool omits this property for a print form.
Variable field folding	Not applicable for a print form.	The migration tool omits this property for a print form.
SO/SI take position	addSpaceForSOSI	No special considerations.
Map size — Lines and Columns	size = (Lines, Columns)	No special considerations.

Table 78. Printer maps — general syntax, map type, and properties (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
Starting position - Line and Column NEXT,SAME is required if the map is a floating map.	position = (Line, Column) If the position information is omitted, the form is a floating form.	If Floating map is selected, the migration tool omits the position information.
Floating map	Not applicable. If the position information is omitted, the form is a floating form.	If Floating map is selected, the migration tool omits the position information.
Device Type - Printer or DBCS Printer	type printForm	The migration tool uses the Device Type information to determine whether to migrate the map to a text or print form.
Supported devices	Not applicable for a print form.	The migration tool omits this property for a print form.
Not applicable. In VisualAge Generator, the message field is always named EZEMSG.	msgField This is the name of the field that is to contain any EGL error messages.	The migration tool sets the msgField property if EZEMSG is anywhere on the map.
Not applicable.	alias	The migration tool includes the *alias* property if the map has to be renamed due to a conflict with an EGL reserved word or because the map name starts with the # symbol. The migration tool also includes the alias property for a map in a program's help map group if the map has to be renamed due to a conflict with the name of a map in the program's main map group. Special considerations apply. See "Map names and help map names" on page 54

Table 79. Map constant and variable fields — general information

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
All positions on a map must be accounted for as one of the following: • a variable field • a constant field • an attribute byte at the beginning of a constant or variable field	All positions on a form do not have to be accounted for. Blank constants that have the default properties (noHighLight, normalIntensity, protect=skip, defaultColor, no outlining, and no cursor) do not need to be specified.	The migration tool omits blank constants that have the default properties.
Constant fields on display maps can have attributes specified that do not really apply to constants. For example: • Unprotected • Input required • Require fill on input • Numeric attribute • Modified data tag	Constant fields on text forms cannot specify properties that do not make sense for a constant.	The migration tool omits properties for constants on text form if the properties are not supported.

Table 79. Map constant and variable fields — general information (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
Constant field on printer maps can have attributes that do not really apply to printers. For example: • Color • Intensity • Highlighting other than underscore • Protection • Initial cursor field • Light pen detect	Constant fields on print forms cannot specify properties that do not make sense for a constant.	The migration tool omits properties for constants on print forms if the properties are not supported.
Variable fields on printer maps can specify attributes that do not really apply to printers. For example: • Color • Intensity • Highlighting other than underscore • Protection • Initial cursor field • Input required • Require fill on input • Numeric attribute • Modified data tag • Light pen detect	Variable fields on print forms cannot specify properties that to not make sense for a print form.	The migration tool omits properties for variable fields on print forms if the properties are not supported.
Variable fields on printer maps can specify edits that do not really apply to printers. For example: • Minimum input • Fold • Hex edit • Input required • Minimum value • Maximum value • Edit messages	Variable field on print forms cannot specify properties that to not make sense for a print form.	The migration tool omits properties for variable fields on print forms if the properties are not supported.

Table 80. Map constant and variable fields — general syntax, data type, length, decimals, and description

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations

A variable on a map has the following information:
• Name
• Information based on what you dropped on the map:
– Data type
– Position
• Basic information:
– Descripton
– Initial value
– Length in bytes
– Array index
– Numeric edit
• Attributes
• Edits, including number of decimals
• Error messages
Note: *position* is the position of the attribute byte. The length in bytes is the length of the field in bytes, excluding the attribute byte. The length in bytes is also used for the length of the data value.
	The information for a variable field on a map includes the following:
	• Name
	• Type and length in characters for character fields
	• Type, precision, and scale for numeric fields
	• Position
	• Field length in bytes
	• Presentation properties
	• Formatting properties
	• Validation properties
	• Value
	In general, the following is true:
	• VAGen attributes correspond to EGL presentation properties.
	• VAGen edits and messages correspond to EGL formatting properties or validation properties.
	• However, some of the VAGen attributes and edits are merged into a single EGL property or moved to a different category.
	Here is an example of an EGL variable field:
	`itemName`
	`  dataType(lengthInformation)`
	`  // description`
	`  { position=(row,column),`
	`    fieldLen=length`
	`    validationOrder=n`
	`    [presentationProperties]`
	`    [formattingProperties]`
	`    [value="initialValue"] }`
	`   [arrayInformation]`
	Note: *position* is the position of the attribute byte. fieldLen is the length of the field in bytes, excluding the attribute byte. The primitive type information given in dataType(lengthInformation) is the length of the data value.
		The migration tool sets the EGL fieldLen property to the VAGen Length in bytes. The tool sets the lengthInformation for the dataType as follows:
		• For char, dbchar, and mbchar fields, migration tool sets the lengthInformation to the number of characters, not the number of bytes.
		• For VAGen char fields that specify the Numeric edit, the migration tool does the following:
		– Converts the field to the EGL num type.
		– Sets the precision to the VAGen length in bytes and then reduces the precision by one if decimals are specified for the field in VisualAge Generator.
		– Sets the scale to the number of decimals specified in VisualAge Generator.
		Special considerations apply. See "Numeric variable fields" on page 56 for details.

Table 80. Map constant and variable fields — general syntax, data type, length, decimals, and description (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
A constant on a map has the following information: • Information based on what you dropped on the map: – Data type – Position • Basic information: – Initial value – Length in bytes • Attributes **Note:** *position* is the position of the attribute byte. The *length in bytes* is the length of the field in bytes, excluding the attribute byte.	The information for a constant field on a map includes the following: • Position • Field length • Presentation properties • Value In general, the following is true: • VAGen attributes correspond to EGL presentation properties. • Attributes that apply only to input editing are not supported for EGL constant fields. The data type for a constant is determined based on the *value* property. Here is an example of an EGL constant field: `{ position=(row,column), fieldLen=length, [presentationProperties] [value="initialValue"] }` **Note:** *position* is the position of the attribute byte. *fieldLen* is the length of the field in bytes, excluding the attribute byte.	The migration tool sets the EGL fieldLen property to the VisualAge Generator Length.
Data type: • Character constant • Character variable • DBCS constant • DBCS variable • Mixed constant • Mixed variable • Character variable with the Numeric edit selected **Note:** The type is determined based on the type of field you drop on the map and whether you select the Numeric edit box.	EGL data type: • Not applicable • char • Not applicable • dbchar • Not applicable • mbchar • num	No special considerations.
Description	Not applicable.	The migration tool converts the description to a comment that follows the data type and length information.

Table 80. Map constant and variable fields — general syntax, data type, length, decimals, and description (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
Initial value	value **Note:** • In VisualAge Generator Compatibility mode, the *value* property is only used when displaying a field on the screen that has not had a value assigned to it. The *value* property is not used to set the initial value of the field in storage. • When VisualAge Generator Compatibility mode is not specified, the *value* property provides the initial value of field in the program when the program starts.	No special considerations.
Length	An EGL variable has the following: • A length, which is the number of characters or digits in the field. • A fieldLen, which is the space the field occupies on the map, excluding the attribute byte.	The migration tool uses the VAGen length to set both the EGL length and the EGL fieldLen properties. Special considerations apply for numeric fields. See "Numeric variable fields" on page 56 for details.

Array index
Note:
• The array size is determined based on the highest array index for the variable field.
• You can override some attributes such as cursor position, color, highlighting, intensity, protection, and cursor position for elements of the array.
• You can also override the initial value for elements of the array.
	`itemName`
	`  datatype(lengthInfo)`
	`  [ arraySize ]`
	`  { properties for index 1 }`
	`  itemName[n]`
	`  { properties for index n }`
	
	Note:
	• The array size is specified immediately after the datatype and length information.
	• You can override cursor location, and presentation properties such as color, highlighting, intensity, and protect.
	• You can also override the value property.
	• You can specify the position of each element by doing the following:
	– Specifying the explicit position of each element with *position(row,column)*.
	– Specifying the following additional properties for index 1: *columns*, *linesBetweenRows*, *spacesBetweenColumns*, and *indexOrientation*.
		The migration tool always explicitly sets the position for each element of the array.

Table 80. Map constant and variable fields — general syntax, data type, length, decimals, and description (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
Field Edit Order **Note:** • Field Edit Order is specified from the Define pulldown. • The default Field Edit Order is based on the position of the variable fields on the map, left to right, then top to bottom. • Some versions of Cross System Product and VisualAge Generator did not record the field edit order in the External Source Format.	validationOrder **Note:** The default validationOrder is based on the position of the variable fields on the map, left to right, then top to bottom.	The migration tool omits the validationOrder if it is not included in the External Source Format for the map.

Table 81. Map constant and variable fields — attributes

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
Intensity: • Normal • Dark • Bright	intensity: • normalIntensity • invisible • bold (presentation property)	No special considerations.
Highlight: • No highlight • Blink • Reverse video • Underscore	highlight: • noHighlight • blink • reverse • underline (presentation property)	No special considerations.
Protection: • Unprotected • Protected • Autoskip	protect: • no • yes • skip (presentation property)	No special considerations.
Color: • Mono • Blue • Red • Pink • Green • Turquoise • Yellow • White	color: • defaultColor • blue • red • magenta • green • cyan • yellow • white (presentation property)	No special considerations.
Initial cursor field	`cursor = yes | no` (form field property)	No special considerations.

Table 81. Map constant and variable fields — attributes (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
Input required	inputRequired (validation property)	The migration tool merges the VAGen Input required attribute and the Input required edit as follows: • If either the Input required attribute or the Input required edit is selected, the migration tool includes inputRequired. • If neither is selected, the migration tool omits inputRequired.
Require fill on input	fill (validation property)	No special considerations.
Numeric attribute **Note:** This property is supported for CHA fields, including CHA fields that have *Numeric edit* selected.	isDecimalDigit (validation property) **Note:** This property is only supported for char fields.	If the Numeric attribute is selected, the migration tool does the following: • Includes isDecimalDigit for char fields. • Omits isDecimalDigit for numeric fields. EGL provides a software edit for numeric fields to maintain compatibility with VAGen. See "Map fields and the numeric hardware attribute" on page 58 for additional details.
Modified data tag	modified (presentation property)	No special considerations.
Light pen detect	detectable (presentation property)	No special considerations.
Outlining: • left • right • over • under • box	outline: • left • right • over • under • box (presentation property)	No special considerations.

Table 82. Map variable fields — general edits

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
Edit routine	validator OR validatorTable (validation property)	The migration tool does the following: • Sets the validator property if the map edit routine is EZEC10 or EZEC11. • Sets the validator property if the edit routine is a function. • Sets the validatorTable property if the edit routine is a table. **Note:** Special considerations apply if the edit routine is not available during migration. See "Variable map fields and edit routines" on page 57 for additional details and potential problems.

Table 82. Map variable fields — general edits (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
Justify - Left | Right | None **Note:** For map items, the default is right for numeric fields and left for all other fields.	align = left | right | none (formatting property) **Note:** For form fields, the default is right for numeric fields and left for all other fields.	No special considerations.
Date edit mask Values are as follows: • SYSGREGRN • SYSJULIAN • dateEditPattern	dateFormat = *value* Values are as follows: • systemGregorian • systemJulian • ″dateEditPattern″ (formatting property)	No special considerations.
Minimum input	minimumInput (validation property)	No special considerations.
Fill character **Note:** The default fill character for items used on a map is null for character, DBCS, or MIXED fields; blank for numeric fields; and 0 for hex fields.	fillCharacter (formatting property) **Note:** The default fill character for items used on a map is null for character, DBCS, or MIXED fields; blank for numeric fields; and 0 for hex fields.	No special considerations.
Fold	upperCase (formatting property)	The migration tool does the following: • If Variable field folding is specified for the entire map, the migration tool includes upperCase=yes for every character and mixed field. • If Variable field folding is not specified for the entire map, the migration tool uses the Fold information specified for each character or mixed field to determine whether to set the upperCase property for that field.
Hex edit	isHexEdit (validation property)	No special considerations.
Input required	inputRequired (validation property)	The migration tool merges the VAGen Input required attribute and the Input required edit as follows: • If either the Input required attribute or the Input required edit is selected, the migration tool includes inputRequired. • If neither is selected, the migration tool omits inputRequired.
Check SO/SI space	needsSOSI (validation property)	No special considerations.

Table 83. Map variable fields — numeric edits

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
Minimum value and Maximum value **Note:** If either Minimum value or Maximum value is specified, both must be specified.	range = (minimumValue, maximumValue) (validation property)	The migration tool combines the Minimum value and Maximum value into the EGL *range* property.

Table 83. Map variable fields — numeric edits (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
Sign: • None • Leading • Trailing **Note:** The default is None.	sign: • none • leading • trailing (validation property) **Note:** The default is none.	No special considerations.
Currency	currency = yes | no (formatting property) **Note:** • If currency = yes, the actual currency symbol used at runtime is set the same way it is in VAGen. • Alternatively, you can specify currency="symbol" for a form variable field.	The migration tool only sets currency to *yes* or *no*. The tool never sets currency="symbol" for form variable fields.
Separator	numericSeparator (formatting property)	No special considerations.
Zero edit	zeroFormat (formatting property)	No special considerations.

Table 84. Map variable fields — error messages

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
Edit routine	validatorMsgKey OR validatorTableMsgKey (validation properties)	The migration tool migrates the edit routine message as follows: • Sets validatorMsgKey if the edit routine is EZEC10 or EZEC11. • Sets validatorTableMsgKey if the edit routine is a table. • Does not migrate the edit routine message if the edit routine is a function because the message is not used in this situation in VisualAge Generator. See "Variable map fields and edit routines" on page 57 for additional details and potential problems.
Minimum input	minimumInputMsgKey (validation property)	No special considerations.
Input required	inputRequiredMsgKey (validation property)	No special considerations.
Data type	typeChkMsgKey (validation property)	No special considerations.
Numeric range	rangeMsgKey (validation property)	No special considerations.

Programs

The programs section is organized into the following tables:

- Programs - general syntax, program type, called parameters, and prolog Table 85 on page 210
- Programs - program specifications, properties, tables and additional records list Table 86 on page 211
- Programs - main functions and flow statements Table 87 on page 213

Table 85. Programs — general syntax, program type, called parameters, and prolog

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations

Programs part:
• programName
• Program type
• Specifications (vary based on program type):
– Working Storage record
– PSB
– Firstmap
– First UI Record
– Map Group
– Help Map Group
• Tables and Additional Records
• Called Parameters
• Prolog
• Properties (vary based on program type)
• Structure diagram
– Main Functions
– Flow Statements (hidden in the structure diagram, but can be specified for any main function)
	EGL syntax sample:
	`//*** Program=programName`
	`// prolog`
	`//******************`
	`Program programName`
	`type eglProgramType`
	`//vagenProgramType`
	`[ ( calledParameters ) ]`
	`{`
	`[alias= "originalProgramName"]`
	`includeReferencedFunctions`
	`=yes,`
	`allowUnqualifiedItemReferences`
	`=yes`
	`[ propertiesBasedOnType ]`
	`}`
	`[ dataDeclarations ]`
	`[ useDeclarations ]`
	`function main ( )`
	`{ functionLabel:`
	`functionName( ) ;`
	`[{functionFlowStatements}]}`
	`end // end main`
	`end // end programName`
		The migration tool does not rename programs for you even if they conflict with the EGL reserved word list. The migration tool does not set the alias property. If you must rename a program, you can use the alias property to specify the original name of the VAGen program. See "Program names and reserved words" on page 62.
		
		The migration tool includes the VAGen program type as a comment in the program definition.
		
		The migration tool migrates the Tables and Additional Records list as follows:
		• Records migrate to dataDeclarations.
		• Tables migrate to useDeclarations.
		
		The migration tool always includes the following properties to preserve VAGen behavior:
		• includeReferencedFunctions
		• allowUnqualifiedItemReferences

Programs types: • Main transaction • Called Transaction • Main Batch • Called Batch • Web Transaction **Note:** See later row on "Main Transaction Exection Mode Values" for additional details.	EGL program types: • textUIProgram • textUIProgram • basicProgram • basicProgram • Not supported in this release.	The migration tool includes the VAGen program type as a comment in the program definition. See Table 86 on page 211 for information on how the segmentation values correspond to EGL properties.

Called Parameters
Note:
• Called parameters are entered in a special window.
• The parameter type indicates whether the parameter is an item, record, or map.
• The parameter name is always the name of another VAGen part. There is no equivalent of an EGL type definition or primitive type.
	EGL called parameters example:
	`( parameterName typeInfo`
	`{ , parameterName typeInfo } )`
	• Parameters must be separated by commas.
	• A parameter can be an item, record, or form. There is no direct correspondence to the VAGen parameter types.
	• The EGL typeInfo can be:
	– a primitive type for an item
	– a type definition for an item, record, or form.
		The migration tool includes the original VAGen parameter type as a comment. The migration tool always migrates called parameters with type definitions. The migration tool does not use primitive types for item parameters.
		
		Special considerations apply. See "Redefined records" on page 47 for details and potential problems.

Prolog	Not applicable.	The migration tool converts the prolog to a comment that precedes the program definition.

Table 86. Programs — program specifications, properties, tables and additional records list

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
General information: • Some VAGen specifications and properties information migrates to EGL properties, data declarations, or use declarations.	General information: • The rows that follow indicate whether the corresponding EGL language element is a program property, data declaration, or use declaration.	No special considerations.
Main Transaction Execution Mode values: • Nonsegmented • Segmented • Single segment **Note:** Called transactions always run in nonsegmented mode.	segmented — values: • segmented = no • segmented = yes • segmented = yes (program property) **Note:** The segmented property is not specified for called programs.	If the segmented information is not in the External Source Format file, the migration tool defaults to *segmented = no*.
Working Storage record (Specifications) • The Working Storage record can be specified for both main and called programs. It is sometimes referred to as the program's primary working storage record. • The primary working storage record is always initialized.	inputRecord (program property) • The inputRecord property can only be specified for main programs. • The inputRecord is always initialized. • A data declaration is also required.	The migration tool converts the primary working storage record to the inputRecord property for main programs. The migration tool also includes a data declaration for the primary working storage record for both main and called programs. The tool includes the *initialized = yes* property for the data declaration in called programs. If the primary working storage record contains level 77 items, the migration tool includes a data declaration statement for the level 77 record. See "Level 77 items in records" on page 48 for details and potential problems.
PSB (Specifications)	Not supported in this release.	The migration tool comments out a use declaration for the PSB. If the program does not use DL/I, you might be able to generate and run it in this release.
Firstmap (Specifications)	inputForm (program property)	No special considerations.
First UI Record (Specifications)	Not supported in this release	No special considerations.
Map Group (Specifications)	**use** *formGroup* (use declaration)	No special considerations.
Help Map Group (Specifications)	**use** *formGroup* **{ helpGroup=yes }** (use declaration)	No special considerations.
Message table prefix (Program property)	msgTablePrefix (program property)	No special considerations.
Allow implicit data items (Program property)	Not supported.	The migration tool does not create implicit definitions for you. See "Implicit data items in programs" on page 63 for details and potential problems.

Table 86. Programs — program specifications, properties, tables and additional records list (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations

Keys assignment:
• Help key (1 key)
• Bypass keys (up to 5 keys)
• F1–12 = F13–24
(Program property)
Note: The keys assignment is specified once for the program and applies to both the map group and the help map group.
	EGL keys assignment example:
	`use formGroup`
	`  { [ helpGroup = yes ]`
	`  helpKey = pfNumber,`
	`  validationBypassKeys =`
	`  ( pfNumberList ) ,`
	`  pfKeyEquate = yes | no  } ;`
	(Use declaration properties for the program's form group and help form group.)
	Note:
	• The values in the validationBypassKeys list must be separated by commas.
	• The validationBypassKeys property is not specified for the program's help form group.
		The migration tool includes the EGL equivalent of the keys assignment information on the *use declaration* statements for both the form group and the help form group. The migration tool omits the validationBypassKeys property from the use declaration for the help form group.

Tables and Additional Records:
• Records
Note:
• Redefinition information is stored in the VAGen Redefined record, not in the program.
• Records that are used as I/O objects are never included in the Tables and Additional Records list.
	EGL additional record example:
	`recordName recordName`
	`  [ { redefines =`
	`  otherRecord } ] ;`
	(data declaration)
	Note:
	• The redefines property must be specified on the program's data declaration if the record provides a different record layout for the same physical storage as another record.
	• Data declarations are required for all records used in the program, including the I/O records.
		The migration tool always uses the same record name as the type definition.
		If a VAGen record is used in the program as a redefined record, the migration tool includes the redefines property on the data declaration statement. See "Redefined records" on page 47 for details and potential problems.
		The migration tool also includes data declarations for all records that are used as I/O objects by the program.
		The migration tool includes data declarations for records that are specified as attributes of any MQ Message record that is used as an I/O object by the program.

Tables and Additional Records:
• Tables
• You can specify Keep After Use for each table.
	EGL *use* declaration example:
	`use tableName`
	`  [{deleteAfterUse = yes}];`
	`(use declaration)`
		The migration tool converts tables on the Tables and Additional Records list to *use* declarations.
		DeleteAfterUse has the opposite meaning from the VAGen Keep After Use. The migration tool reverses yes and no.

Table 87. Programs — main functions and flow statements

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations

VAGen programs specify the main functions as the top-level functions in the VAGen Structure Diagram. All other functions appear only when you expand the Structure Diagram.
 	
Each main function can have flow statements. These statements do not appear in the Structure Diagram, but can be accessed from the diagram.
	EGL programs specify only one main function. This function is always named *main*.
	
	There are no flow statements.
	
	An example of the syntax for the program's main function is as follows:
	`function main ( )`
	`{ functionLabel:`
	`functionName( ) ;`
	`[ { functionFlowStatements } ] }`
	`end // end main`
		The migration tool builds the EGL main function. The tool includes the following within the main function for each VAGen main function:
		• functionLabel so that the VAGen main function can be referenced in an EGL *exit stack functionLabel* statement. The tool always sets the functionLabel to the functionName.
		• function invocation statement to invoke the VAGen main functionName.
		• flow statements, if any, for the VAGen main function.
		
		See the following for details on the migration of flow statements:
		• See "Statements" on page 225
		• See "EZE words" on page 238
		• See "Service Routines" on page 247

Functions

The following tables compare the VAGen function part with the EGL function part and describe how the migration tool handles the conversion.

The functions section is organized into the following tables:

- Functions - general syntax, description, parameters, return value, and local storage, Table 88 on page 214
- Functions - EXECUTE I/O option, Table 89 on page 216
- Functions - I/O options for maps and UI records, Table 90 on page 216
- Functions - I/O error routine for records, Table 91 on page 217
- Functions - I/O options for serial, indexed, relative, and message queue records, Table 92 on page 217
- Functions - I/O options for default (unmodified) SQL statements, without Execution Time Statement Build, Table 93 on page 218
- Functions - I/O options for modified SQL statements, without Execution Time Statement Build, Table 94 on page 220
- Functions - I/O options for SQL statements with Execution Time Statement Build, Table 95 on page 223

Note: This release of EGL does not support web transactions, UI records, or a replacement for CONVERSE UI record. However, the EGL converse statement is expected to be the same for both a map and a UI record. The migration tool converts the CONVERSE I/O option without regard to whether the I/O object is a map or a UI record. This preserves as much of your logic as possible.

Table 88. Functions — general syntax, description, parameters, return value, and local storage

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations

Function parts can contain the following:
• Function name
• I/O option
• I/O object
• Properties:
– Error routine
– Description
• Function return value
• Function parameters
• Function local storage
• SQL statement
• Statements before the I/O option
• Statements after the I/O option
• DL/I call
	Function parts can contain the following:
	• functionName
	• functionParameterList
	• returnItemType
	• dataDeclarations
	• Statements before the I/O statement
	• I/O statement
	• Statements after the I/O Statements
	
	An example of the format of a function invocation statement created by the migration tool is as follows:
	`// Description`
	`Function functionName`
	`(functionParamterList)`
	`[ returns( returnItemType ) ]`
	`[ dataDeclarations ]`
	`[ beforeStatements ]`
	`[ I/O Statement ]`
	`[ afterStatements ]`
	`end // end functionName`
	
	Note:
	• The VAGen I/O option, I/O object, error routine, and SQL statement are used to create the EGL I/O statement.
	• DL/I call is not supported in this release.
		The migration tool uses the I/O option, I/O object, and error routine to build the EGL I/O statement. See Table 91 on page 217
		
		For details on how the migration tool handles the Description, Function return value, Function parameters, and Function local storage, see the following rows in this table.
		
		See the following for details on the migration of before and after statements:
		• For statements, see “Statements” on page 225
		• For EZE words, see “EZE words” on page 238
		• For service routines, see “Service Routines” on page 247
		
		See the following for details on I/O options and error routines:
		• See Table 89 on page 216 for the EXECUTE I/O option.
		• See Table 90 on page 216 for I/O options for maps and UI records.
		• See Table 91 on page 217 for I/O error routine for records.
		• See Table 92 on page 217 for I/O options for serial, indexed, relative, and message queue records.
		
		For details on SQL statements, see the following tables in this section:
		• See Table 93 on page 218 for I/O options for unmodified SQL statements.
		• See Table 94 on page 220 for I/O options for modified SQL statements without Execution Time Statement Build.
		• See Table 95 on page 223 for I/O options for SQL statements with Execution Time Statement Build.

Description	Not applicable	The migration tool converts the function description to a comment that precedes the Function definition.

Table 88. Functions — general syntax, description, parameters, return value, and local storage (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations

Function parameters:
• Function parameters are entered in a special window.
• Items used as function parameters can be shared or nonshared. The definition for nonshared functions is stored in the function.
	Function parameters:
	• Parameters must be separated by commas.
	• Each parameter has type information.
	• Optionally, each parameter has parameter type information.
	An example of the format of function parameters is as follows:
	`( parameterName typeInfo`
	`[ parameterType ]`
	`{ , parameterName typeInfo`
	`[ parameterType ] } )`
	A specific example of function parameters is as follows:
	`(parmSharedItem parmSharedItem`
	`field,`
	`parmNonSharedItem char(10)`
	`nullable,`
	`parmRecord parmRecord)`
		Function parameters:
		The migration tool sets the type information as follows:
		• For a record, the typeInfo is a type definition that specifies the same record name.
		• If the item type is one of the VAGen Any* types, the typeInfo is the corresponding EGL special item type.
		• If the item is a shared data item, then the typeInfo is the itemName used as a type definition.
		• If the item is a nonshared data item, then the typeInfo is migrated based on the item type, length, and decimals, and follows the rules described in Table 61 on page 179.

Function parameters: • Function parameter types: – Record – Item – Map item – SQL item	Function parameters: • Function parameter types: – Not applicable – Not applicable – field – nullable	
Function parameters: • Special item types, length is not specified: – AnyChar – AnyDBCS – AnyMix – AnyHex – AnyUnicode – AnyNumeric	Function parameters: • Special item types, length is not specified: – char – dbchar – mbchar – hex – unicode – number	

Function return value:
• Data type
• Length
• Decimals
• Description
	EGL returns value:
	• The following is an example of the *returns* statement format:
	`returns( returnItemType )`
	`// Description`
		If the function includes a return value, the migration tool migrates the data type, length, and decimals based the rules described in Table 61 on page 179.

Table 88. Functions — general syntax, description, parameters, return value, and local storage (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations

Function local storage:
• Function local storage is entered in a special window.
• Items used as function local storage can be shared or nonshared. The definition for nonshared items is stored in the function.
	Function variable declarations:
	• Function variable declarations must include variable names and their associated type information.
	• An example of the format of function variable declarations is as follows:
	`// Function Declarations`
	`variableName typeInfo ;`
	`{ variableName typeInfo ; }`
		Function local storage:
		The migration tool sets the typeInfo as follows:
		• For a record, the typeInfo is a type definition that specifies the same record name.
		• If the item is a shared data item, then the typeInfo is the itemName used as a type definition.
		• If the item is a nonshared data item, then the typeInfo is migrated based on the item type, length, and decimals, and follows the rules described in Table 61 on page 179.

Function local storage: • Function local storage types: – Record – Item	Function local storage: • Function local storage types: – Not applicable – Not applicable	

Table 89. Functions — EXECUTE I/O option

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
• I/O object: none • I/O option: EXECUTE	No equivalent statement.	The migration tool eliminates the EXECUTE I/O option.

Table 90. Functions — I/O options for maps and UI records

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations

• I/O object: mapName
• I/O option: DISPLAY
Note: DISPLAY is used for both display and printer maps.
	To display a text form, use the *display* statement. To print a print form, use the *print* statement.
	The following are examples of a display statement and a print statement:
	`display mapName;`
	`print mapName;`
	Note: In VisualAge Generator Compatibility mode, *display printForm* is treated as though it is *print printForm.*
		The migration tool converts to the display or print statement based on the map type. See "DISPLAY statement for maps" on page 66 for details and potential problems.

• I/O object: mapName • I/O option: CONVERSE	Use the *converse* statement. The following is an example of a converse statement: `converse mapName;`	No special considerations.
• I/O object: UIRecordName • I/O option: CONVERSE	Use the *converse* statement. The following is an example of a converse statement: `converse UIRecordName;`	No special considerations.

Table 91. Functions — I/O for records — general information and I/O error routine

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations

VAGen record I/O:
• I/O option
• I/O object (always a record)
• I/O error routine (optional)
	EGL record I/O:
	• An I/O statement
	• Record name
	• *try onException end* statement with error routine name (optional)
	
	If an I/O error routine is specified, the statements are enclosed within a *try...end* block. An example of record I/O with an error routine is as follows:
	`try`
	`  add recordName ;`
	`  [onException  error-routine ; ]`
	`end`
		The migration tool does the following:
		• Changes the VAGen I/O option to the corresponding EGL I/O statement.

An error routine is optional for functions that do I/O for records.
Note: The error routine is invoked if there is a soft error or if EZEFEC = 1.
	An error routine is optional for functions that do I/O for records. An I/O example without an error routine is as follows:
	`add recordName;`
	
	An I/O example with an error routine is as follows:
	`try`
	`  add recordName ;`
	`  onException  error-routine ;`
	`end`
	
	Note: The *onException* statement is invoked if there is a soft error or if handleHardIOErrors = 1.
		The migration tool does the following:
		• If the error-routine is not specified, the tool does not include the try, onException, or end statements.
		• If an error-routine is specified, the tool includes the try and end statements.
		• The migration tool converts to the onException statement based on the VAGen error routine name. When the migration tool migrates programs, it always migrates the VAGen main function names to both the main function label and the main function invocation statement. That way, when migrating a function's I/O error routine, the mainFunctionLabel is always the same as the mainFunctionName.
		
		Special considerations apply for the migration of error routines that are function names. See "I/O error routine" on page 67 for details and potential problems.

Error routine values:
 	
`EZECLOS`
`EZEFLO`
`EZERTN`
`mainFunctionName`
 	
`nonmainFunctionName`
	onException block statements:
	
	`onException exit program;`
	`onException exit stack;`
	`Omit the onException statement.`
	`onException exit stack`
	`  mainFunctionLabel;`
	`onException`
	`  nonmainFunctionName();`	

Table 92. Functions — I/O options for serial, indexed, relative, and message queue records

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
• I/O object: recordName • I/O option: ADD	Use the *add* statement. The following is an example: `add recordName;`	No special considerations.
• I/O object: recordName • I/O option: SCAN	Use the *get next* statement. The following is an example: `get next recordName;`	No special considerations.
• I/O object: recordName • I/O option: SCANBACK	Use the *get previous* statement. The following is an example: `get previous recordName;`	No special considerations.

Table 92. Functions — I/O options for serial, indexed, relative, and message queue records (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
• I/O object: recordName • I/O option: CLOSE	Use the *close* statement. The following is an example: `close recordName;`	No special considerations.
• I/O object: recordName • I/O option: INQUIRY	Use the *get* statement. The following is an example: `get recordName;`	No special considerations.
• I/O object: recordName • I/O option: UPDATE	Use the *get forUpdate* statement. The following is an example: `get recordName forUpdate;`	No special considerations.
• I/O object: recordName • I/O option: DELETE	Use the *delete* statement. The following is an example: `delete recordName;`	No special considerations.
• I/O object: recordName • I/O option: REPLACE	Use the *replace* statement. The following is an example: `replace recordName;`	No special considerations.

Table 93. Functions — I/O options for default (unmodified) SQL statements without Execution Time Statement Build)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
• I/O object: recordName • I/O option: ADD	Use the *add* statement. The following is an example: `add recordName;`	No special considerations.
• I/O object: recordName • I/O option: SCAN	Use the *get next* statement. The following is an example: `get next recordName;`	No special considerations.
• I/O object: recordName • I/O option: CLOSE	Use the *close* statement. The following is an example: `close recordName;`	No special considerations.

• I/O object: recordName
• I/O option: INQUIRY
(with and without Single row select)
	Use the *get* statement. If you are doing a single row select, also use *singleRow*. An example without single row select is as follows::
	`get recordName;`
	An example with single row select is as follows:
	`get recordName singleRow;`
		If Single row select is specified in VisualAge Generator, the migration tool includes the EGL singleRow option.

• I/O object: recordName • I/O option: UPDATE	Use the *get forUpdate* statement. The following is an example: `get recordName forUpdate resultSetID;`	The migration tool always includes the *resultSetID* when migrating an SQL UPDATE statement. The tool sets the resultSetID to the function name followed by a customer-specified suffix. You can control the suffix with the Stage 2 VAGen Migration Syntax Preferences. Special considerations apply if the migration tool cannot determine if the record is SQL or non-SQL. See "SQL I/O with multiple updates" on page 73 for details and potential problems.

Table 93. Functions — I/O options for default (unmodified) SQL statements without Execution Time Statement Build) (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
• I/O object: recordName • I/O option: DELETE	Use the *delete* statement. The following is an example: `delete recordName;`	No special considerations.

• I/O object: recordName
• I/O option: REPLACE
(with or without UPDATE/SETUPD functionName
	Use the *replace* statement. The following are some examples:
	`replace recordName;`
	`replace recordName from resultSetID;`
		If the UPDATE/SETUPD function name was included in VisualAge Generator, the migration tool includes the resultSetID and sets the resultSetID to the UPDATE/SETUPD function name followed by a customer-specified suffix. You can control the suffix with the Stage 2 VAGen Migration Syntax Preferences.

• I/O object: recordName
• I/O option: SETINQ
(with and without Declare cursor with hold)
	Use the *open* statement. If you are doing a *Declare cursor with hold*, also use the *hold* option. The following are examples of both types of statement:
	`open resultSetID for recordName;`
	`open resultSetID hold for recordName;`
		The migration tool sets the resultSetID to the function name followed by a customer-specified suffix. You can control the suffix with the Stage 2 VAGen Migration Syntax Preferences.
		If *Declare cursor with hold* is selected in VisualAge Generator, the migration tool includes the EGL *hold* option.

• I/O object: recordName
• I/O option: SETUPD
(with and without Declare cursor with hold)
	Use the *open forUpdate* statement. If you are doing a *Declare cursor with hold*, also use the *hold* option. The following are examples of both types of statement:
	`open resultSetID forUpdate for recordName;`
	`open resultSetID hold forUpdate for recordName;`
		The tool sets the resultSetID to the function name followed by a customer-specified suffix. You can control the suffix with the Stage 2 VAGen Migration Syntax Preferences.
		If *Declare cursor with hold* is selected in VisualAge Generator, the migration tool includes the EGL *hold* option.

• I/O object: recordName • I/O option: SQLEXEC with Model SQL Statement **Note:** • The SQL record name is included in this form of SQLEXEC. • The values for the model type are: – None – Update – Delete • If the model type is None, VisualAge Generator does not do any I/O. Generation still processes the I/O error routine, but there will not be an error.	Use the *execute* statement. The following is an example: `execute modelType for recordName;` **Note:** *modelType* is either update or delete.	The migration tool sets the EGL modelType based on the VAGen Model SQL Statement value. If the VAGen Model SQL Statement is None, the migration tool omits the I/O statement because the VAGen I/O statement did not do anything. The migration tool includes the *try*, *onException*, and *end* statements based on the function's I/O error routine.

Table 94. Functions — I/O options for modified SQL statements, without Execution Time Statement Build

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
General Information for modified SQL statements: • VisualAge Generator builds the table clause from the SQL row record at test and generation time. The table clauses are as follows: – **insert into** *sqlTableName* for the ADD I/O option – **from** *sqlTableName sqlTableLabel* for the INQUIRY, UPDATE, SETINQ, and SETUPD I/O options – **update** *sqlTableName* for the REPLACE I/O option • Depending on when the function was last modified, other SQL clauses might not be stored in the function definition. If the SQL clause is not stored, VisualAge Generator creates a default clause based on the record definition of the I/O object. • !itemColumnName variables are permitted. These variables specify the name of an item in the SQL row record. At test or generation time, VisualAge Generator substitutes the corresponding SQL column name. • sqlClauses are written in SQL syntax.	General Information for modified SQL statements: • If you need to modify any SQL clause, EGL requires that all clauses be explicitly specified. The table clause must be explicitly included in the SQL statement. The table clauses are as follows: – **insert into** *sqlTableName* for the add statement. – **from** *sqlTableName sqlTableLabel* for the get and open statements. – **update** *sqlTableName* for the replace statement. • EGL requires that all clauses be explicitly specified if any SQL clause is specified. The required SQL clauses vary with the type of I/O. • EGL requires that the SQL column names be explicitly included in the SQL statement. !itemColumnName variables are not supported. • sqlClauses are written in SQL syntax.	The migration tool uses the tables and table labels from the SQL row record to build the tables clause for the EGL I/O statement. Both table names and table name host variables are included in the table clause of the EGL I/O statement. If a required SQL clause is not stored in the function definition, the migration tool creates a default clause based on the record definition in the same way as in VisualAge Generator. The migration tool converts any !itemColumnName variables to their corresponding SQL column name. The migration tool converts VAGen comments (/*) to SQL comments (—) Special considerations apply if the SQL record and its alternate specification record, if any, are not available during migration. See "SQL I/O statements" on page 69 for details and potential problems.

• I/O object: recordName
• I/O option: ADD
 	
Clauses that can be modified:
• Columns
• VALUES
	Use the *add* statement. The following is an example:
	`add recordName`
	`  with #sql{`
	`   insert into`
	`   sqlTablename`
	`   (columnName1, columnName2)`
	`   values`
	`   (valueInfo1, valueInfo2)`
	`};`
		The migration tool creates the *insert into* clause based on the table name in the record definition. Special considerations apply. See "SQL I/O statements" on page 69 for details and potential problems.

Table 94. Functions — I/O options for modified SQL statements, without Execution Time Statement Build (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations

• I/O object: recordName
• I/O option: INQUIRY
(with and without Single row select)
Clauses that can be modified:
• SELECT
• INTO
• WHERE, GROUP BY, HAVING
• ORDER BY
	Use the *get* statement.
	The following is an example without single row select:
	`get recordName singleRow`
	`  with #sql{`
	`  select`
	`  Name1,`
	`  Name2,`
	`  Age`
	`  from`
	`  sqlTable1 sqlLabel1,`
	`  sqlTable2 sqlLabel2`
	`  where`
	`  Name1 = :Namex`
	`  order by`
	`  Age`
	`}`
	`into`
	`nameA, nameB, myage;`
		If Single row select is specified in VisualAge Generator, the migration tool includes the EGL singleRow option.
		The migration tool creates the *from* clause based on the table names and table labels in the record definition. Special considerations apply. See "SQL I/O statements" on page 69 for details and potential problems.

• I/O object: recordName
• I/O option: UPDATE
Clauses that can be modified:
• SELECT
• INTO
• WHERE
• FOR UPDATE OF
	Use the *get forUpdate* statement .
	The following is an example:
	`get recordName forUpdate`
	`resultsetID`
	`  with #sql{`
	`  select`
	`  Name1, Name2, Age`
	`  from`
	`  sqlTable1 sqlLabel1`
	`  where`
	`  Name1 = :Namex`
	`  for update of`
	`  Name2, Age`
	`  }`
	`  into`
	`  Name1, Name2, Age;`
		The migration tool always includes the resultSetID when migrating an UPDATE statement for an SQL record. The tool sets the resultSetID to the function name followed by a customer-specified suffix. You can control the suffix with the Stage 2 VAGen Migration Syntax Preferences.
		The migration tool creates the *from* clause based on the table name or table label in the record definition. Special considerations apply. See "SQL I/O statements" on page 69 for details and potential problems.

• I/O object: recordName
• I/O option: REPLACE
(optional UPDATE/SETUPD functionName)
Clause that can be modifed:
• SET
	Use the *replace* statement.
	The following is an example of the *replace* statement:
	`replace recordName`
	`with #sql{`
	`  update`
	`  sqlTableName`
	`  set`
	`  columnName1 = value1,`
	`  columnName2 = value2`
	`}`
	`from resultSetID;`
		If an UPDATE/SETUPD function name is included in VisualAge Generator, the migration tool includes the *from resultSetID* clause. The migration tool sets the resultSetID to the UPDATE/SETUPD function name followed by a customer-specified suffix. You can control the suffix with the Stage 2 VAGen Migration Syntax Preferences.
		The migration tool creates the *update* clause based on the table name in the record definition. Special considerations apply. See "SQL I/O statements" on page 69 and "SQL I/O and missing required SQL clauses" on page 70 for details and potential problems.

Table 94. Functions — I/O options for modified SQL statements, without Execution Time Statement Build (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations

• I/O object: recordName
• I/O object: SETINQ
(with or without Declare cursor with hold)
• I/O object: recordName
• I/O object: SETINQ
(with or without Declare cursor with hold)
Clauses that can be modified:
• SELECT
• INTO
• WHERE, GROUP BY, HAVING
• ORDER BY
	Use the *open* statement. If you are doing a *Declare cursor with hold*, also use the *hold* option.
	The following is an example of an *open* statement using the *hold* option:
	`open resultSetID hold`
	`with #sql{`
	`select`
	`Name1, Name2`
	`from`
	`sqlTable1 sqlLabel1,`
	`sqlTable2 sqlLabel2`
	`where`
	`Name1 > :Name2`
	`order by`
	`Name1`
	`}`
	`into Name1, Name2`
	`for recordName;`
		The tool sets the resultSetID to the function name followed by a customer-specified suffix. You can control the suffix with the Stage 2 VAGen Migration Syntax Preferences.
		If *Declare cursor with hold* is selected in VisualAge Generator, the migration tool includes the EGL *hold* option after the resultSetID.
		The migration tool creates the *from* clause based on the table names and table labels in the record definition. Special considerations apply. See "SQL I/O statements" on page 69 for details and potential problems.

• I/O object: record
• I/O option: SETUPD
(with or without Declare cursor with hold)
Clauses that can be modified:
• SELECT
• INTO
• WHERE
• FOR UPDATE OF
	Use the *open forUpdate* statement. The following is an example using the *hold* option:
	`open resultSetID hold forUpdate`
	`with #sql{`
	`select`
	`Column1, Column2`
	`from`
	`sqlTable1 sqlLabel1,`
	`sqlTable2 sqlLabel2`
	`where`
	`Column1 > :Item1`
	`for update of`
	`Column2`
	`}`
	`into Item1, Item2`
	`for recordName;`
		The migration tool sets the resultSetID to the function name followed by a customer-specified suffix. You can control the suffix with the Stage 2 VAGen Migration Syntax Preferences.
		If *Declare cursor with hold* is selected in VisualAge Generator, the migration tool includes the EGL *hold* option.
		The migration tool creates the *from* clause based on the table name or table label in the record definition. Special considerations apply. See "SQL I/O statements" on page 69 for details and potential problems.

• I/O object: record
• I/O option: SQLEXEC
with Model SQL Statement
Note:
• The SQL record name is optional in this form of SQLEXEC.
• The values for the model type are as follows:
– None
– Update
– Delete
	Use the *execute sql* statement. The following is an example of the statement:
	`execute modelType`
	`#sql{`
	`UPDATE mysqltable`
	`set Column1 = Column1 * 2`
	`where Column2 = :Column2`
	`}`
	`for recordName;`
	Note: The values for model type include Update and Delete.
		The migration tool does the following:
		• Converts SQLEXEC to the *execute* statement.
		• Uses the I/O object, if it is specified, as the recordName in the *for* clause.
		The migration tool includes the VAGen Model SQL Statement value, if any, as a comment on the EGL *execute* statement.
		The migration tool migrates the VAGen SQLEXEC clauses to EGL SQL clauses.

Table 95. Functions - I/O options for modified SQL statements with Exection Time Statement Build

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations

Execution time statement build can only be used with the following I/O options:
• INQUIRY
• UPDATE
• SETINQ
• SETUPD
• SQLEXEC
 	
You specify Execution time statement build to cause VisualAge Generator to prepare the SQL statement dynamically every time the I/O statement is executed.
	In EGL, you code the SQL prepare statement directly whenever you want the SQL statement to be dynamically prepared. You must also code the open, execute, or get statement that follows the prepare. For example, the EGL equivalent of a VAGen INQUIRY I/O option with Execution time statement build is as follows:
	`prepare prepID from`
	`  "sqlStatementString"`
	`  for recordName;`
	`get recordName with prepID`
	`  into itemList;`
	
	Note:
	• The sqlStatementString in the prepare statement is a concatenated string of constants and variables that is written in SQL notation. An example of a where clause that uses both column names and variable is as follows:
	`[other clauses]`
	`+ " where columnName = "`
	`+ itemName`
	`+ " AND columnName2 = "`
	`+ itemName2`
	`+ .... [other clauses]`
	• The examples shown in the rest of this table do not include splitting the variables outside the double quotes.
		The migration tool uses the SQL clauses in the function and the table names and /or table name variables in the record definition to build the sqlStatementString. The migration tool builds the sqlStatementString as follows:
		• Does all the processing as though the Execution time statement build were not specified, including the following:
		– Using the table names and / or table labels from the SQL row record to build the tables clause for the EGL I/O statement. Both table names and table name host variables are included in the tables clause of the EGL I/O statement.
		– Creating default clauses as necessary based on the record definition.
		– Converting any !itemColumnName variables to their corresponding SQL column name.
		– Converting VAGen comments (/*) to EGL comments (//) in the prepare statement.
		
		Then the migration tool does additional processing to create the sqlStatementString, including:
		• Enclosing constants, column names and SQL operators in double quotes.
		• Placing variables outside double quotes.
		• Using the + string concatenation operator to concatenate the strings and variables together.
		
		Special considerations apply if the SQL record and its alternate specification record, if any, are not available during migration. See “SQL I/O statements” on page 69 and “SQL I/O and missing required SQL clauses” on page 70 for details and potential problems.

Table 95. Functions - I/O options for modified SQL statements with Exection Time Statement Build (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations

• I/O object: record
• I/O option: INQUIRY
with and without Single row select
Clauses that can be modified:
• SELECT
• INTO
• WHERE, GROUP BY, HAVING
• ORDER BY
	Use the *prepare* statement. The following is an example:
	`prepare prepID from`
	`  " select columnName "`
	`+ ", columnName2 "`
	`+ " from table1 t1 "`
	`+ "[ where whereClause ]"`
	`+ "[order by orderByClause ]"`
	`[ for recordName ];`
	`get recordname with prepID`
	`  into itemList;`
		No special considerations.

• I/O object: record
• I/O option: UPDATE
Clauses that can be modified:
• SELECT
• INTO
• WHERE
• FOR UPDATE OF
	Use the *prepare* statement. The following is an example:
	`prepare prepID from`
	`  " select columnName "`
	`+ ", columnName2 "`
	`+ " from table1 t1 "`
	`+ "[ where whereClause ]"`
	`+ " for Update of columnList "`
	`[ for recordName ];`
	
	`get recordName forUpdate`
	`  resultSetID`
	`  with prepID`
	`  into itemList;`
		The migration tool always includes the resultSetID when migrating an UPDATE statement for an SQL record. The tool sets the resultSetID to the function name followed by a customer-specified suffix. You can control the suffix with the Stage 2 VAGen Migration Syntax Preferences.

• I/O object: record
• I/O option: SETINQ
with or without Declare cursor with hold
Clauses that can be modified:
• SELECT
• INTO
• WHERE, GROUP BY, HAVING
• ORDER BY
	Use the *prepare* statement. The following is an example:
	`prepare prepID from`
	`  " select columnName "`
	`+ ", columnName2 "`
	`+ " from table1 t1 "`
	`+ "[ where whereClause ]"`
	`+ "[order by orderByClause ]"`
	`[ for recordName ];`
	`open resultSetID [ hold ]`
	`  with prepID`
	`  into itemList`
	`  [ for recordName ];`
		The migration tool sets the resultSetID to the function name followed by a customer-specified suffix. You can control the suffix with the Stage 2 VAGen Migration Syntax Preferences.
		If *Declare cursor with hold* is selected in VisualAge Generator, the migration tool includes the EGL hold option after the resultSetID.

• I/O object: record
• I/O option: SETUPD
with or without Declare cursor with hold
Clauses that can be modified:
• SELECT
• INTO
• WHERE
• FOR UPDATE OF
	Use the *prepare* statement. The following is an example:
	`prepare prepID from`
	`  " select columnName "`
	`+ ", columnName2 "`
	`+ " from  table1 t1 "`
	`+ "[ where whereClause ] "`
	`+ " for update of columnList "`
	`[ for recordName ] ;`
	`open resultSetID [ hold ]`
	`forUpdate`
	`with prepID`
	`into itemList`
	`[ for recordName ] ;`
		The tool sets the resultSetID to the function name followed by a customer-specified suffix. You can control the suffix with the Stage 2 VAGen Migration Syntax Preferences.
		If *Declare cursor with hold* is selected in VisualAge Generator, the migration tool includes the EGL hold option after the resultSetID.

Table 95. Functions - I/O options for modified SQL statements with Exection Time Statement Build (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations

• I/O object: record
• I/O option: SQLEXEC
with Model SQL Statement
Note:
• The SQL record name is optional in this form of SQLEXEC.
• The values for the model type are as follows:
– None
– Update
– Delete
	Use the *prepare* statement. The following is an example:
	`prepare prepID from`
	`  " grant " + group_privileges`
	`+ " on " + table_name`
	`+ " to " + userid`
	`[ for recordName ] ;`
	`execute prepID`
	`[ for recordName ] ;`
	`// model = type`
		The migration tool includes the VAGen Model SQL Statement value, if any, as a comment on the EGL execute statement.
		The migration tool converts the VAGen SQLEXEC clauses to EGL SQL clauses.

Statements

The statements section is organized into the following tables:

Table 96. Statements - General rules - data item qualification and numeric literals

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
Data item qualification rules: If an item is not qualified, VisualAge Generator looks for the item in the following order: • Items in the function's local storage or parameter list. • The function's I/O object and records in the function's local storage or parameter list. If the item name is not unique in this category, the item name must be qualified. • Records, maps, and tables in the program's primary working storage record, called parameter list, table and additional records list, and all I/O objects. If the item name is not unique in this category, the item name must be qualified. • If the item name is not found within the program and the program allows implicit items, VisualAge Generator creates a data item definition based on the use of the item.	Data item qualification rules: If an item is not qualified, EGL looks for the item in the following order: • Items in the function's local storage or parameter list. • The records and forms used in the function's I/O statements and records in the function's local storage or parameter list. If the item name is not unique in this category, the item name must be qualified. • Records, forms, and tables in the program's data declarations, use declarations, and parameter list. If the item name is not unique in this category, the item name must be qualified. • EGL does not permit implicit items. Every item must be explicitly defined.	See "Level 77 items in records" on page 48and "Implicit data items in programs" on page 63 for details and potential problems.
Numeric literals: • Not enclosed in quotes. • Can use either a period (.) or a comma (,) as the decimal point, depending on the national language.	Numeric literals: • Not enclosed in quotes. • Must use the period as the decimal point. At generation time, the decimalSymbol build descriptor option determines whether the period or comma is used as the decimal point in the generated Java or COBOL code.	The migration tool converts the commas used as decimal points to a period except for initial values of form variable fields.

Table 97. Statements — Function invocation

VisualAge Generator4.5	EGL produced by the migration tool	Migration tool considerations
VAGen syntax example: `functionName( [argumentList] );`	EGL syntax example: *`functionName`*`( [`*`argumentList`*`] );`	See "EZE words" on page 238 for the EGL equivalent system library functions. See Table 91 on page 217 for function invocations from an I/O error routine.
In flow statements: `functionName();`	Flow statements are not supported. **`goto`** *`functionName`*`;`	No special considerations.

Table 98. Statements — Assignment, MOVE, and MOVEA

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
VAGen syntax example: `target = functionName ( [argumentList] ) ;`	EGL syntax example: *target = functionName ([argumentList]) ;*	See "EZE words" on page 238 for the EGL equivalent system library functions.

`target = numericExpression;`
or
`target = numericExpression (R;`
	target = numericExpression ;
	or
	target = **sysLib.round** (*numericExpression*) ;
		If the (R option is specified, the migration tool converts the option to the EGL round system function.

`target = source;`
or
`MOVE source [ TO ] target;`
 	
Note:
• The target can be a record, map, data item, or certain EZE data words.
• The source can be a record, map, literal, data item, or certain EZE data words.
• If the target is a record or map, the source must also be a record or map. A move corresponding occurs.
	target = source ;
	or
	move *source* **to** *target* **byName ;**
	or
	move *source* **to** *target ;*
	
	Note:
	• For assignment statements:
	– The target can be a record, data item, or certain system variables. If the target is a record, the source must also be a record; the source is moved to the target on a byte-by-byte basis.
	– The source can be a record, literal, data item, or certain system variables.
	– Forms cannot be used in assignment statements.
	– Move corresponding is never done for an assignment statement.
	• For move statements:
	– The target and source can be the same as in VisualAge Generator assignment or MOVE statements.
	– If *byName* is specified, EGL does a move corresponding.
	– If no modifier is specified, the move is either an item to item move or a move corresponding depending on the part type of the source.
	
	The data conversion and truncation rules are the same as in VisualAge Generator.
		The migration tool considers the following EGL rules when migrating assignment and move statements:
		• EGL prefers that the assignment statement be used for item-to-item moves.
		• The *move byName* statement is required for moves involving records or forms to preserve the VAGen *move corresponding* behavior.
		• Move without a modifier is tolerated and treated as an item-to-item move or a move corresponding depending on the part type of the source.
		
		Therefore, the migration tool does the following:
		• Converts to an assignment statement for any of the following:
		– The source or target is an EZE data word (for example: EZEAPP).
		– The source is a literal.
		– The source or target is a qualified or subscripted item.
		– The source or target is an item in the function's parameter list or local storage.
		• Converts to a move byName if the source or target is the function's I/O object or a record in the function's parameter list or local storage.
		• Converts to a move without a modifier in all other situations.
		
		See "Assignment statements" on page 74 for details and potential problems.

Table 98. Statements — Assignment, MOVE, and MOVEA (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations

`MOVEA source [TO] target;`
or
`MOVEA source [TO] target`
`FOR occurrence;`
 	
Note: The *source* can be an array or a scalar.
	move *source* **to** *target* **for all ;**
	or
	move *source* **to** *target*
	for *occurrence* **;**
		The migration tool converts the MOVEA statement to a move statement with the *for* modifier. The tool also does the following:
		• Includes the *for all* option if the FOR occurrence option was not specified in VisualAge Generator.
		• Includes the *for occurrence* option if the FOR occurrence option was specified in VisualAge Generator.
		• Sets the target subscript to 1 if the subscript was not previously specified.
		• Does not set the subscript to 1 for the source because the source can be an array or a scalar item.

Table 99. Statements — SET

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
General information: • Commas or blanks can be used to separate multiple options on a single SET statement.	General information: • Commas are required to separate multiple options on a single set statement.	No special considerations.

`SET record SCAN;`
 	
OR
 	
`SET record EMPTY;`
 	
Note: SET record EMPTY does not affect level 77 items.
	set record **position;**
	
	OR
	
	set record **empty;**
		The migration tool does not add a statement for the level 77 record.

SET sqlItem NULL; **Note:** *sqlItem* can be an item in an SQL row record or an SQLITEM parameter for a function.	**set** *sqlItem* **null ;** **Note:** *sqlItem* can be an *isNullable=yes* item in an SQL row record or a nullable parameter for a function.	No special considerations.

`SET map [ ALARM |`
`[ CLEAR | EMPTY] ] ;`
 	
Note:
• CLEAR and EMPTY are mutually exclusive.
• ALARM and either CLEAR or EMPTY can be combined with the PAGE option.
	set *form* [**alarm** |
	[**initial** | **empty**]] **;**
	• *Initial* and *empty* are mutually exclusive.
	• The replacement for the PAGE option cannot be combined with any other options.
		If ALARM, CLEAR, or EMPTY are used in combination with the PAGE option, the migration tool splits the VAGen statement into two EGL statements.

Table 99. Statements — SET (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations

`SET map PAGE ;`
 	
Note: PAGE can be combined with ALARM and with either CLEAR or EMPTY.
	`sysLib.clearScreen();`
	`// display form`
	or
	`sysLib.pageEject();`
	`// printer form`
	
	Note: The replacement for the PAGE option cannot be combined with any other options.
		The migration tool migrates SET map PAGE as follows:
		• If SET map PAGE is used in combination with any other options, the migration tool splits the VAGen statement into two EGL statements.
		• If the map is a display map, the migration tool converts the statement to *sysLib.clearScreen();*
		• If the map is a printer map, the tool converts the statement to *sysLib.pageEject();*
		• If the map is not available to determine the map type, the migration tool converts the statement to *sysLib.EZE_SETPAGE();*
		
		See "SET map PAGE statement" on page 76 for details and potential problems.

`SET mapItem`
`[ CURSOR | FULL |`
`[ NORMAL | DEFINED ] ] ;`
 	
Note:
• *mapItem* can be an item on a map or a MAPITEM parameter for a function.
• NORMAL and DEFINED are mutually exclusive.
• CURSOR and FULL can be combined with either NORMAL or DEFINED.
• VisualAge Generator tolerates setting CURSOR, FULL, NORMAL, and DEFINED for print maps, but they had no effect on the printed output.
	`set` *formField*
	`[ cursor | full |`
	`[ normal | initialAttributes ] ] ;`
	
	Note:
	• *formField* can be a variable field on a form or a field parameter for a function.
	• *normal* and *initialAttributes* are mutually exclusive.
	• *cursor* and *full* can be combined with either *normal* or *initialAttributes*.
	• EGL does not support setting *cursor, full, normal*, or *initialAttributes* for print forms.
		The migration tool migrates to the EGL equivalent of each option without regard to whether the formField is on a text or print form. See "SET mapItem attributes" on page 77 for details and potential problems.

Table 99. Statements — SET (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
SET mapItem [CURSOR | FULL | color | extendedHighlight | MODIFIED | [BRIGHT | DARK] | [PROTECT | AUTOSKIP]] ; **Note:** • *mapItem* can be an item on a map or a MAPITEM parameter for a function. • BRIGHT and DARK are mutually exclusive. • PROTECT and AUTOSKIP are mutually exclusive. • Any of the other options can be combined. • VisualAge Generator tolerates setting these attributes for print maps. However, only the extended highlighting option of USCORE has any effect on the printed output.	**set** *formField* [**cursor** | **full** | *color* | *extendedHighlight* | **modified** | [**bold** | **invisible**] | [**protect** | **skip**]] ; **Note:** • *formField* can be a variable field on a form or a field parameter for a function. • *Bold* and *invisible* are mutually exclusive. • *Protect* and *skip* are mutually exclusive. • Any of the other options can be combined. • Except for the extended highlighting option of underline, EGL does not support setting these attributes for print forms.	The migration tool migrates to the EGL equivalent of each option without regard to whether the formField is on a text or print form. See "SET mapItem attributes" on page 77 for details and potential problems. See later rows in this table for color and *extendedHighlight* information.
color: MONO | BLUE | GREEN | PINK | RED | TURQ | YELLOW | WHITE	color: defaultColor | blue | green | magenta | red | cyan | yellow | white	No special considerations.
extendedHighlight: NOHILITE | BLINK | RVIDEO | USCORE	extendedHighlight: noHighLight | blink | reverse | underline	No special considerations.

Table 100. Statements — RETRIEVE and FIND

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
RETR dataItem1 table[.searchColumn] dataItem2 [returnColumn] ; **Note:** • If the *searchColumn* is not specified, the default is the first column in the table. • If the *returnColumn* is not specified, the default is the second column in the table.	**if** (*dataItem1* **in** *table.searchColumn*) **dataItem2** = *table.returnColumn*[**sysVar.arrayIndex**]; **end** **Note:** The *searchColumn* and *returnColumn* are required.	The migration tool converts the RETR statement to an *if* statement and an assignment statement. Special considerations apply if the table is not available during migration. See "RETR statement" on page 75 for details and potential problems.

Table 100. Statements — RETRIEVE and FIND (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations

`FIND dataItem`
`  table[.searchColumn]`
`  trueStatement;`
 	
OR
 	
`FIND  dataItem`
`  table[.searchColumn]`
`  , falseStatement ;`
 	
OR
 	
`FIND  dataItem`
`  table[.searchColumn]`
`  trueStatement`
`  [,]  falseStatement ;`
 	
Note:
• If the *searchColumn* is not specified, the default is the first column in the table.
• If FIND is used in program flow, the *trueStatement* and the *falseStatement* can be the name of a main function or EZECLOS.
• If FIND is used in a function, the *trueStatement* and the *falseStatement* can be the name of any function, EZECLOS, EZEFLO, or EZERTN.
	if (*dataItem in table.searchColumn*)
	`  EGLtrueStatement ;`
	end
	
	OR
	
	if (*dataItem* **in** *table.searchColumn*)
	else
	`  EGLfalseStatement ;`
	end
	
	OR
	
	if (*dataItem* **in** *table.searchColumn*)
	`  EGLtrueStatement ;`
	else
	`  EGLfalseStatement ;`
	end
	
	Note: The *searchColumn* is required.
		The migration tool converts the FIND statement to an *if* statement and the EGL equivalent of the true and false statements. See later rows in this table for conversion of the trueStatement and falseStatement to the corresponding EGL statements.

true/falseStatement in flow: • functionName() (main only) • EZECLOS	**Corresponding EGL replacements:** • **goto** *functionName***;** • **exit program;**	No special considerations.
true/falseStatement in a function: • functionName (any function) • EZECLOS • EZEFLO • EZERTN	**Corresponding EGL replacements:** • *functionName()***;** • **exit program;** • **exit stack;** • **return;**	No special considerations.

Table 101. Statements — IF, WHILE, and TEST, including EZEAID, EZESYS, and I/O error values

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations

`IF logicalExpression ;`
`  { statement ; }`
`[ ELSE;`
`  { statement ; } ]`
`END;`
	if (*EGLLogicalExpression*)
	`  { EGLStatement ; }`
	[else
	`  { EGLStatement ; } ]`
	end
		See later rows in this table for the relationship between the VAGen logical expressions and the EGL logical expressions.

`WHILE logicalExpression ;`
`  { statement ; }`
`END;`
	while (*EGLLogicalExpression*)
	`  { EGLStatement ; }`
	end
		See later rows in this table for the relationship between the VAGen logical expressions and the EGL logical expressions.

Table 101. Statements — IF, WHILE, and TEST, including EZEAID, EZESYS, and I/O error values (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations

`TEST testCondition`
`  trueStatement ;`
 	
`TEST testCondition`
`  , falseStatement ;`
 	
`TEST testCondition`
`  trueStatement`
`  [, ] falseStatement ;`
 	
Note:
• The TEST statement is similar to an IF ... IS statement. The exception to this is TEST mapItem nnn, +nnn, and -nnn, which does not have an IF statement equivalent.
• If TEST is used in program flow, the *trueStatement* and the *falseStatement* can be the name of a main function or EZECLOS.
• If TEST is used in a function, the *trueStatement* and the *falseStatement* can be the name of any function, EZECLOS, EZEFLO, or EZERTN.
	`if ( EGLLogicalExpression )`
	`  EGLtrueStatement ;`
	`end`
	
	`if ( EGLLogicalExpression )`
	`else`
	`  EGLfalseStatement ;`
	`end`
	
	`if ( EGLLogicalExpression )`
	`  EGLtrueStatement ;`
	`else`
	`  EGLfalseStatement ;`
	`end`
		With the exception of TEST mapItem nnn, +nnn, and -nnn, the migration tool converts the TEST statement to the equivalent *if ... is* statement and the EGL equivalent of the true and false statements.
		
		See later rows in this table for the relationship between the VAGen logical expressions and the EGL logical expressions.
		
		See later rows in this table for conversion of the *trueStatement* and *falseStatement* to the corresponding EGL statements.

VisualAge Generator boolean operators for IF and WHILE: • AND • OR	Corresponding EGL boolean operators for *if* and *while*: • && • ||	No special considerations.
VisualAge Generator relational operators for IF and WHILE: • EQ and = • NE and ^= • LE and <= and =< • LT and < • GE and >= and => • GT and >	Corresponding EGL relational operators for *if* and *while*: • = • != • <= • < • >= • >	No special considerations.
VisualAge Generator state operators for IF and WHILE: • IS • NOT	Corresponding EGL state operators for *if* and *while*: • is • not	The migration tool always migrates a VAGen TEST statement to an EGL *if ... is* statement.
VisualAge Generator array operator for IF and WHILE: • IN	Corresponding EGL state operators for *if* and *while*: • in	No special considerations.

Table 101. Statements — IF, WHILE, and TEST, including EZEAID, EZESYS, and I/O error values (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
VisualAge Generator *mapItem* state conditions: • BLANK or BLANKS • CURSOR • DATA • MODIFIED • NULL or NULLS • NUMERIC	Corresponding EGL *formField* state conditions: • blanks • cursor • data • modified • blanks • numeric	The migration tool converts to the equivalent EGL state conditions. Special considerations apply to *mapItem* NULL. See "Checking SQL and map items for NULL" on page 78 for details and potential problems.
Special mapItem state condition for the TEST statement: nnn | +nnn | -nnn **Note:** This compares the length of the data the user entered to nnn. The test is =, >, or < corresponding to nnn, +nnn, or -nnn.	EGL does not provide direct support for this state condition. However, you can achieve the equivalent function by doing the following: • Use the system library function *sysLib.fieldInputLength*, which returns the length of the data entered by the user. • Use an *if* statement to compare the resulting length for = , >, or < corresponding to nnn, +nnn, or -nnn, respectively.	When migrating any program, the migration tool always includes a declaration for: `<custPrefix>EZE_ITEMLEN` The migration tool does the following for TEST nnn, +nnn, or -nnn: • Adds an extra statement just before the TEST statement to set `<custPrefix>EZE_ITEMLEN` using the system library function *sysLib.fieldInputLength(item)*. • Changes the TEST statement to an if statement and compares `<custPrefix>EZE_ITEMLEN` to = nnn, > nnn, or < nnn.
VisualAge Generator map state conditions: • MODIFIED	Corresponding EGL form state conditions: • modified	No special considerations.
VisualAge Generator EZEAID state conditions: • ENTER • BYPASS • PA*n*, where n = 1, 2, 3 • PF*n*, where n is 1 to 24 • PA • PF	Corresponding EGL *sysVar.eventKey* state conditions: • enter • bypass • pa*n*, where n = 1, 2, 3 • pf*n*, where n is 1 to 24 • pakey • pfkey	No special considerations.
VisualAge Generator *sqlItem* state conditions: • BLANK or BLANKS • NULL • NUMERIC • TRUNC	Corresponding EGL *sqlItem* state conditions: • blanks • null • numeric • trunc	The migration tool converts to the equivalent EGL state conditions. Special considerations apply to sqlItem NULL. See "Checking SQL and map items for NULL" on page 78 for details and potential problems.

Table 101. Statements — IF, WHILE, and TEST, including EZEAID, EZESYS, and I/O error values (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
VisualAge Generator record state conditions: • DED • DUP • EOF • ERR • FMT • FNA • FNF • FUL • HRD • LOK • NRF • UNQ **Note:** • DUP is supported for both SQL and non-SQL records. • For SQL records, DUP and UNQ are equivalent and are always hard errors. • For non-SQL records, DUP and UNQ are not equivalent; both are soft errors. • LOK is only supported on OS/400 and is a soft error.	Corresponding EGL record state conditions: • deadLock • duplicate or unique • endOfFile • ioError • invalidFormat • fileNotAvailable • fileNotFound • full • hardIOError • deadLock • noRecordFound • unique **Note:** • *duplicate* is only supported for non-SQL records and is a soft error. • *unique* is a hard error for both SQL and non-SQL records. • LOK is converted to *deadLock*, which is always a hard error.	The migration tool converts to the equivalent EGL state conditions. Special considerations apply to migrating DUP based on the record type. See "I/O error values UNQ and DUP" on page 79 for details and potential problems. Special considerations also apply to migrating LOK. See "I/O error value LOK" on page 81 for details and potential problems.
User Interface record state conditions: • MODIFIED	Corresponding EGL uiRecord conditions: • modified	This release of EGL does not support web transactions or UI records. However, testing the UI record state for MODIFIED is expected to be the same for both a map and a UI record. The migration tool converts a logical expression that tests for MODIFIED without regard to whether the part being tested is a map or a UI record. This preserves as much of your logic as possible.
VisualAge Generator *dataItem* state conditions: • BLANK or BLANKS • NUMERIC	Corresponding EGL *dataItem* state conditions: • blanks • numeric	No special considerations.

Table 101. Statements — IF, WHILE, and TEST, including EZEAID, EZESYS, and I/O error values (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
VisualAge Generator EZESYS state conditions: • AIX • AIXCICS • HP • IMSBMP • IMSVS • MVSBATCH • MVSCICS • NTCICS • OS2 • OS2CICS • OS2GUI • OS400 • SCO • SOLACICS • SOLARIS • TSO • VMCMS • VMBATCH • VSEBATCH • VSECICS • WINGUI • WINNT • ITF	Corresponding *sysVar.systemType* state conditions: • aix • AIXCICS • hp • imsbmp • imsvs • zosbatch • zoscics • NTCICS • OS2 • OS2CICS • OS2GUI • iseriesc • SCO • SOLACICS • solaris • TSO • VMCMS • VMBATCH • VSEBATCH • VSECICS • WINGUI • win • debug	The migration tool converts to the equivalent EGL state conditions. Special considerations apply to checking the state for EZESYS. See "EZESYS" on page 83 for details and potential problems. **Note:** Not all of the VAGen runtime environments are supported by this release. However, the migration tool always converts to an equivalent value, even if it will not be valid in EGL. There will be an error in the Problems view if the old VAGen value is not currently supported.
true/falseStatement in flow: • functionName() (main only) • EZECLOS	**Corresponding EGL replacements:** • **goto** functionName **;** • **exit program;**	No special considerations.
true/falseStatement in a function: • functionName (any function) • EZECLOS • EZEFLO • EZERTN	**Corresponding EGL replacements:** • functionName()**;** • **exit program;** • **exit stack;** • **return**;	No special considerations.

Table 102. Statements — CALL

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations

`CALL programName argument`
`  [ { [,] argument} ]`
`  [ (options ] ;`
 	
OR
 	
`CALL serviceRoutine argument`
`  [ { [,] argument} ]`
`  [ (options ] ;`
 	
Note: Commas to separate the arguments are optional.
	call *programName argument*
	`  [ { , argument } ]`
	`  [ options ]  ;`
	
	Note:
	• Commas to separate the arguments are **required**.
	• The programName cannot be a reserved word. If the program is a non-EGL program, use a linkage table entry to specify the real name.
		See later rows in this table for conversion of the options to the corresponding EGL statements or options.
		
		See "Service Routines" on page 247 for information on migrating the CALL statement for them.

REPLY option	If the REPLY option is specified in VisualAge Generator, the corresponding EGL statements are as follows:
	try
	`  call programName argument`
	`  [ { , argument } ]`
	`  [ otherOptions ]  ;`
	end
		The migration tool includes the *try...end* block if the REPLY option is specified.

otherOptions: • NOMAPS • NONCSP	Corresponding EGL otherOptions: • noRefresh • externallyDefined	No special considerations.

Table 103. Statements — DXFR

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations

`DXFR target`
`  [ recordName ]`
`  [ (NONCSP ] ;`
 	
where target is
 	
`programName`
 	
OR
 	
`EZEAPP`
 	
Note:
• Any record can be passed.
• If a working storage record is passed, any level 77 items are not included.
	transfer to program *target*
	`  [ passing  recordName ]`
	`  [ externallyDefined ] ;`
	
	where *target* is
	
	`programName`
	
	OR
	`sysVar.transferName`
	
	Note: Any record can be passed.
		No special considerations.

Table 104. Statements — XFER

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations

Variation 1 - Migrate to Transfer (no map or UI record)
`XFER target`
`[ recordName ]`
`[ (NONCSP] ;`
 	
where target is
 	
`transactionName`
 	
OR
 	
`EZEAPP`
 	
Note:
• This format of XFER does not include a map or UI record.
• Any record can be passed. If a working storage record is passed, any level 77 items are not included.
• *transactionName* is the *programName* in nontransactional runtime environments.
	EGL syntax for *transfer* statement:
	transfer to transaction *target*
	[**passing** *recordName*]
	[**externallyDefined**] ;
	
	where *target* is
	
	`transactionName`
	
	OR
	`sysVar.transferName`
	
	Note:
	• Any record can be passed.
	• *transactionName* is the program name in nontransactional runtime environments.
		If there is no comma in the statement, the migration tool converts the XFER to an EGL transfer to transaction statement.

Variation 2 - Migrate to Show (XFER with map)
`XFER target`
`[ recordName ]`
`, map`
`[ (NONCSP ] ;`
 	
where target is
 	
`transactionName`
 	
OR
 	
`EZEAPP`
 	
Note:
• Any record can be passed. If a working storage record is passed, any level 77 items are not included.
• *transactionName* is the *programName* in nontransactional target environments.
	EGL syntax for *show* statement:
	show *formName*
	returning to *target*
	[**passing** *recordName*]
	[**externallyDefined**] ;
	
	where *target* is
	
	`transactionName`
	
	OR
	
	`sysVar.transferName`
	
	Note:
	• Any record can be passed.
	• *transactionName* is the *programName* in nontransactional target environments.
		The migration tool distinguishes between Variation 2 (XFER with a map) and Variation 3 (XFER with a UI record) as follows:
		• Use Variation 2 (XFER with a map) if any of the following are true:
		– (NONCSP is specified.
		– The second argument is a map.
		• Use Variation 3 (XFER with a UI record), if any of the following are true:
		– The target is ' '.
		– The name of the second argument is longer than 8 characters.
		– The second argument is a UI record.
		• If the migration tool is unable to determine if the second argument is a map or UI record, the tool uses variation 3.
		
		Special considerations apply if the migration tool cannot distinguish between a map and a UI record. See "XFER" on page 82 for details and potential problems.

Table 104. Statements — XFER (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations

Variation 3 - Migrate to Show (XFER with UI Record)
`XFER target [ record ]`
`, UIRecord`
 	
where target is
 	
`transactionName`
 	
OR
 	
`EZEAPP`
 	
OR
 	
`' '`
 	
Note:
• Any record can be passed. If a working storage record is passed, any level 77 items are not included.
• *transactionName* is the *programName* in nontransactional target environments.
	EGL syntax for *forward* statement:
	`forward UIRecord`
	`[ returning to target ]`
	`[ passing recordName ] ;`
	
	where target is
	
	`transactionName`
	
	OR
	
	`sysVar.transferName`
	
	OR
	`UI record`
	
	Note:
	• Any record can be passed.
	• *transactionName* is the *programName* in nontransactional target environments.
		This release of EGL does not support web transactions, UI records, or a replacement for XFER with a UI record. However, the *forward* statement is expected to be the replacement for XFER with a UI record. The EGL syntax might not be correct. However, using this migration technique preserves as much of your logic as possible. The migration tool distinguishes between Variation 2 (XFER with a map) and Variation 3 (XFER with a UI record) as follows:
		• Use Variation 2 (XFER with a map) if any of the following are true:
		– (NONCSP is specified.
		– The second argument is a map.
		• Use Variation 3 (XFER with a UI record), if any of the following are true:
		– The target is ' '.
		– The name of the second argument is longer than 8 characters.
		– The second argument is a UI record.
		• If the migration tool is unable to determine if the second argument is a map or UI record, the tool uses variation 3.
		
		Special considerations apply if the migration tool cannot distinguish between a map and a UI record. See "XFER" on page 82 for details and potential problems.

EZE words

Note: VAGen EZE words in the left column of the tables are matched with their EGL equivalents in the right column.

Program flow EZE words

The left table column shows the VisualAge Generator 4.5 EZE word. The right column shows what the migration tool converts the EZE word to in EGL.

Table 105. Program flow EZE words

EZE word in VisualAge Generator 4.5	EGL

EZECLOS	**This depends on the location:**
	• If used as an I/O error routine, the migration tool converts EZECLOS to the following, within the try...end block:
	`onException exit program;`
	• Used anywhere else, including use as the true or false operand of a TEST or FIND, the migration tool converts EZECLOS to the following:
	`exit program;`
	Note: The exit program has a default return code of <ezeDataPrefix>.returnCode, which is the equivalent of EZERCODE. This default provides the same capability as VisualAge Generator.

EZEFLO
Note: EZEFLO cannot be used in flow statements.
	This depends on the location:
	• If used as an I/O error routine, the migration tool converts EZEFLO to the following, within the try...end block:
	`onException exit stack;`
	• Used anywhere else, including use as the true or false operand of a TEST or FIND, the migration tool converts EZEFLO to the following:
	`exit stack;`

EZERTN or EZERTN(return value)
Note:
• EZERTN cannot be used in flow statements.
• EZERTN(return value) cannot be used as an I/O error routine.
	EZERTN — This depends on the location:
	• If used as an I/O error routine, the migration tool includes the try...end block but omits the onException statement.
	• Used anywhere else, the migration tool converts EZERTN to the following:
	`return;`
	`OR`
	`return(returnValue);`
	Note: If the returnValue is EZESYS, see EZESYS for additional considerations.

SQL EZE words

The left table column shows the VisualAge Generator 4.5 EZE word. The right column shows what the migration tool converts the EZE word to in EGL.

Table 106. SQL EZE words

EZE word in VisualAge Generator 4.5	EGL definition
EZECONCT	sysLib.connectionService The arguments are the same as in VAGen. However, for debug and Java generation, not all of the values for the unit of work argument are supported. JDBC only supports single-phase commit.
EZESQCOD	sysVar.sqlcode
EZESQISL **Note:** • For VisualAge Generator 4.5, EZESQISL is supported for use with ODBC. • Otherwise, EZESQISL is not supported or is ignored in VisualAge Generator for all environments, but it has been kept for compatablity.	sysVar.sqlIsolationLevel
EZESQLCA	sysVar.sqlca **Note:** sysVar.sqlca is only partially supported in EGL. For debug and Java generation, EGL does not set the fields within sysVar.sqlca that contain the values for sysVar.sqlerrmc and sysVar.sqlwarn[7].
EZESQRD3	sysVar.sqlerrd[3] **Note:** The migration tool changes this to an array reference.
EZESQRRM	sysVar.sqlerrmc **Note:** sqlerrmc is not supported for debug or Java generation.
EZESQWN1	sysVar.sqlwarn[2] **Note:** The migration tool changes this to an array reference.
EZESQWN6	sysVar.sqlwarn[7] **Note:** The migration tool changes this to an array reference. sqlwarn[7] is not supported for debug or Java generation.
N/A	sysVar.sqlState **Note:** This is new for EGL and has no equivalent in VisualAge Generator 4.5. The migration tool does not convert anything to this.

Date and time EZE words

The left table column shows the VisualAge Generator 4.5 EZE word. The right column shows what the migration tool converts the EZE word to in EGL.

Table 107. Date and time EZE words

EZE word in VisualAge Generator 4.5	EGL definition
EZEDAY	sysVar.currentShortJulianDate
EZEDAYL	sysVar.currentJulianDate

Table 107. Date and time EZE words (continued)

EZE word in VisualAge Generator 4.5	EGL definition
EZEDAYLC	sysVar.currentFormattedJulianDate
EZEDTE	sysVar.currentShortDate
EZEDTEL	sysVar.currentDate
EZEDTELC	sysVar.currentFormattedDate
EZETIM	sysVar.currentFormattedTime

Other data EZE words

The left table column shows the VisualAge Generator 4.5 EZE word. The right column shows what the migration tool converts the EZE word to in EGL.

Table 108. Other data EZE words

EZE word in VisualAge Generator 4.5	EGL definition
EZEAID	sysVar.eventKey
EZEAPP	sysVar.transferName
EZECNVCM	sysVar.commitOnConverse
EZECONVT	sysVar.callConversionTable
record.EZEDEST	record.resourceAssociation **Note:** The qualification is still the record name.
EZEDESTP	sysVar.printerAssociation
EZEFEC	sysVar.handleHardIOErrors
EZELOC	sysVar.remoteSystemID
EZELTERM	sysVar.terminalID

EZEMNO	• If EZEMNO is used as the *target* of a MOVE or assignment, the following occurs:
	– If EZEMNO is set from a numeric literal other than 9999, EZEMNO becomes:
	`validationFailed(numericLiteral);`
	– If EZEMNO is set from numeric literal 9999, EZEMNO becomes:
	`validationFailed();`
	– If EZEMNO is set from an item, EZEMNO becomes:
	`if (itemName = 9999)`
	`  validationFailed();`
	`else`
	`  validationFailed(itemName);`
	`end`
	• If EZEMNO is used anywhere else, it is replaced with:
	`sysVar.validationMsgNum`

Table 108. Other data EZE words (continued)

EZE word in VisualAge Generator 4.5	EGL definition
EZEMSG **Note:** EZEMSG as a data item exists only if it is placed on a map. If it is placed on multiple maps, EZEMSG must be qualified.	<custPrefix>EZEMSG **Note:** • There is no dot between <custPrefix> and EZEMSG • Where EZEMSG is used in functions, the migration tool keeps the same qualifications for <custPrefix>EZEMSG that were used by EZEMSG in those functions. For example, xxxx.EZEMSG becomes xxxx.<custPrefix>EZEMSG • Where EZEMSG is used in maps, the migration tool does the following: – Changes the field name to <custPrefix>EZEMSG – Sets the map msgField property to <custPrefix>EZEMSG
EZEOVER	sysVar.handleOverflow
EZEOVERS	sysVar.overflowIndicator
EZERCODE **Note:** VisualAge Generator permitted, but did not recommend, negative values and values greater than 512 for EZERCODE.	sysVar.returnCode **Note:** EGL does not permit negative values or values greater than 512 for sysVar.handleSysLibErrors.
EZEREPLY	sysVar.handleSysLibErrors
EZERT2 **Note:** In VisualAge Generator 4.5, EZERT2 is used only as the condition code for MQ Series access.	sysVar.mqConditionCode
EZERT8 **Note:** EZERT8 is set for the following: • CALL statements if the (REPLY option is specified. • EZE system function invocations if EZEREPLY is set to 1. • I/O statements for serial, indexed, relative, and message queue records.	sysVar.errorCode **Note:** sysVar.errorCode is set for the following: • All CALL statements. • All sysLib system function invocations. • I/O statements for serial, indexed, relative, and message queue records. The value of sysVar.errorCode changes more frequently in EGL than it did in VisualAge Generator.
EZESEGM	sysVar.segmentedMode
EZESEGTR	sysVar.transactionID

Table 108. Other data EZE words (continued)

EZE word in VisualAge Generator 4.5	EGL definition
EZESYS	To use the EGL values in an *if* or *while* statement, use: **sysVar.systemType** To get the old VAGen values for use in any other statement, use: *myItem* = **<ezeSysLib>.getVAGSysType();** and then use *myItem* in the statement. If you need to use the old VAGen value in a migrated VAGen program, use: **<***custPrefix***>EZESYS** where <custPrefix> is the Renaming Prefix you specified during Stage 2 of migration. The migration tool always includes a data declaration for <custPrefix>EZESYS and a statement to initialize it to the old VAGen value. See "EZESYS" on page 83 for details and potential problems.
EZETST **Note:** Set for IF...IN, and MOVEA. EZETST is 2–byte binary.	sysVar.arrayIndex **Note:** arrayIndex is int (4–byte binary).
EZEUSR	sysVar.sessionID
EZEUSRID	sysVar.userID

General function EZE words

The left table column shows the VisualAge Generator 4.5 EZE word. The right column shows what the migration tool converts the EZE word to in EGL. Except where noted, the argument lists are the same in VisualAge Generator as they are in EGL, so they are omitted from the table.

Table 109. General function EZE words

EZE word in VisualAge Generator 4.5	EGL definition
result = EZEBYTES(itemOrRecord) **Note:** VisualAge Generator documents that only items and records can be used as arguments for EZEBYTES. However, VisualAge Generator tolerates a map as the argument for EZEBYTES.	result = sysLib.bytes(itemOrRecord) **Note:** EGL does not support a map as the argument for sysLib.bytes. The migration tool converts the argument without regard to whether it is an item, record, or map.
EZECOMIT()	sysLib.commit()
EZECONV(target,direction, conversionTable)	sysLib.convert
EZEC10(xxx, yyy, zzz)	sysLib.verifyChkDigitMod10

Table 109. General function EZE words (continued)

EZE word in VisualAge Generator 4.5	EGL definition
EZEC11(xxx, yyy, zzz)	sysLib.verifyChkDigitMod11
EZEG10(xxx, yyy, zzz)	sysLib.calculateChkDigitMod10
EZEG11(xxx, yyy, zzz)	sysLib.calculateChkDigitMod11
EZEPURGE(queueName)	sysLib.purge
EZEROLLB()	sysLib.rollback()
EZEWAIT(variableName) **Note:** *variableName* provides the time in hundredths of a second.	sysLib.wait(*variableName*); **Note:** • *variableName* provides the time in seconds. • The migration tool converts the time to seconds. See "EZEWAIT" on page 85 for details and potential problems.

String EZE words

The left table column shows the VisualAge Generator 4.5 EZE word. The right column shows what the migration tool converts the EZE word to in EGL. The argument lists are the same in VisualAge Generator as they are in EGL, so they are omitted from the table.

Table 110. String EZE words

EZE word in VisualAge Generator 4.5	EGL definition
EZESBLKT	strLib.setBlankTerminator
EZESCCWS	strLib.concatenateWithSeparator
EZESCMPR	strLib.compareStr
EZESCNCT	strLib.concatenate
EZESCOPY	strLib.copyStr
EZESFIND	strLib.findStr
EZESNULT	strLib.setNullTerminator
EZESSET	strLib.setSubStr
EZESTLEN	strLib.strLength
EZESTOKN	strLib.getNextToken

Math EZE words

The left table column shows the VisualAge Generator 4.5 EZE word. The right column shows what the migration tool converts the EZE word to in EGL. Except where noted, the argument lists are the same in VisualAge Generator as they are in EGL, so they are omitted from the table.

Table 111. Math EZE words — General math functions

EZE word in VisualAge Generator 4.5	EGL definition
EZEABS	mathLib.abs
EZECEIL	mathLib.ceiling
EZEEXP	mathLib.exp
EZEFLOOR	mathLib.floor

Table 111. Math EZE words — General math functions (continued)

EZE word in VisualAge Generator 4.5	EGL definition
EZEFREXP	mathLib.frexp
EZELDEXP	mathLib.ldexp
EZELOG	mathLib.log
EZELOG10	mathLib.log10
EZEMAX	mathLib.maximum
EZEMIN	mathLib.minimum
EZEMODF	mathLib.modf
EZENCMPR	mathLib.compareNum
EZEPOW	mathLib. pow
EZEPRSCN	mathLib.precision
EZEROUND	mathLib.round **Note:** mathLib.round is also used to replace VAGen statements with the *(R* option. The assignment statement migration for those statements has the following syntax: `result = mathLib.round(numericExpression);`
EZESQRT	mathLib.sqrt

Table 112. Math EZE words — Trigonometric math functions

EZE word in VisualAge Generator 4.5	EGL definition
EZEACOS	mathLib.acos
EZEASIN	mathLib.asin
EZEATAN	mathLib.atan
EZEATAN2	mathLib.atan2
EZECOS	mathLib.cos
EZECOSH	mathLib.cosh
EZESIN	mathLib.sin
EZESINH	mathLib.sinh
EZETAN	mathLib.tan
EZETANH	mathLib.tanh

Table 113. Math EZE words — Floating point math functions

EZE word in VisualAge Generator 4.5	EGL definition
EZEFLADD	mathLib.floatingSum
EZEFLDIV	mathLib.floatingQuotient
EZEFLMOD	mathLib.floatingMod
EZEFLMUL	mathLib.floatingProduct
EZEFLSET	mathLib.floatingAssign
EZEFLSUB	mathLib.floatingDifference

User interface EZE words

This release of EGL does not support web transactions or UI records. However, there are replacements for the VisualAge Generator EZEUIxxx special function words for use in the new EGL page handlers. The migration tool converts the EZEUIxxx special function words to their EGL equivalent. This preserves as much of your logic as possible. You might be able to use the function if you develop any new EGL page handlers.

Table 114. User interface EZE words

EZE word in VisualAge Generator 4.5	EGL definition
EZEUIERR	sysLib.setError
EZEUILOC	sysLib.setLocale

EZE Java words

Table 115. EZE Java words

EZE word in VisualAge Generator 4.5	EGL definition
EZEJAVA*******	sysLib.java******* **Note:** There are a variety of Java words. The migration tool deletes the EZE and concatenates the prefix in its place.

Object scripting EZE words

Table 116. Object scripting EZE words

EZE word in VisualAge Generator 4.5	EGL definition
EZESCRPT(targetScriptName)	The migration tool issues an error message. The function part cannot be migrated correctly. The original EZESCRPT statement is included as a comment in the EGL source code.

DL/I EZE words

Note: This release of EGL does not support DL/I. However, if any of the EZEDL words are used in your functions, the migration tool migrates the EZEDL word as shown in the following table. The migration tool uses what is currently expected to be the EZEDL status word replacement in a future release of EGL. This preserves as much of your logic as possible. The migration tool also issues an error message. You cannot test, generate, or run any program that uses the function in this release of EGL.

Table 117. DL/I EZE words

EZE word in VisualAge Generator 4.5	Expected definition in future release of EGL
EZEDLCER	sysVar.dliCicsErrorCode
EZEDLCON	sysVar.dliCicsConditionCode
EZEDLDBD	sysVar.dliDbdName
EZEDLERR	sysVar.handleHardDliErrors
EZEDLKEY	sysVar.dliKey
EZEDLKYL	sysVar.dliKeyLength

Table 117. DL/I EZE words (continued)

EZE word in VisualAge Generator 4.5	Expected definition in future release of EGL
EZEDLLEV	sysVar.dliLevel
EZEDLPCB **Note:** This is an array with a default subscript of 1.	sysVar.dliPcb **Note:** If no subscript was specified, the tool changes this to sysVar.dliPcb[1].
EZEDLPRO	sysVar.dliPcbOptions
EZEDLPSB	sysVar.dliPsbName
EZEDLRST	sysVar.dliCicsProgramRestarted
EZEDLSEG	sysVar.dliSegmentName
EZEDLSSG	sysVar.dliSegmentCount
EZEDLSTC	sysVar.dliStatusCode
EZEDLTRM **Note:** EZEDLTERM is equivalent to EZECNVCM. Both are converted to commitOnConverse at migration.	sysVar.commitOnConverse **Note:** This prefix is for general data EZE words, not for DL/I specifically.

Service Routines

The service routines section is organized into the following tables:

- Service Routines - general syntax, Table 118 on page 247
- Service Routines - VisualAge Generator and EGL equivalent routines, Table 119 on page 247

Table 118. Service Routines - general syntax

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
`CALL serviceRoutine` `  [ argumentList ] ;`	`syslib.`*`EGLSystemFunction`* `  [ (`*`argumentList`*`) ] ;` **Note:** EGL system functions use the same argument list as in VisualAge Generator.	No special considerations.
`CALL serviceRoutine` `  [ argumentList ] (REPLY ;`	**`try`** `  sysLib.`*`EGLSystemFunction`* `  [ (`*`argumentList`*`) ] ;` **`end;`** **Note:** EGL system functions use the same argument list as in VisualAge Generator.	If the (REPLY option is included in VisualAge Generator, the migration tool includes a *try... end* block.

Table 119. Service Routines - VisualAge Generator and EGL equivalent routines

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
CALL AUDIT	sysLib.audit	No special considerations.
CALL COMMIT	sysLib.commit	No special considerations.
CALL CREATX	sysLib.startTransaction	No special considerations.
CALL CSPTDLI	sysLib.callDli **Note:** CSPTDLI is not supported in this release.	The migration tool converts the call to *sysLib.callDli* and issues an error message. You cannot run the program.

Table 119. Service Routines - VisualAge Generator and EGL equivalent routines (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
CALL EZCHART	**call EZCHART** [*arguments*] **externallyDefined ;** **Note:** There is no replacement for EZCHART in EGL.	The VAGen migration tool converts EZCHART to a call to an externally defined program. If the REPLY option is specified in VisualAge Generator, the migration tool nests the call statement within a *try ... end* block.
CALL RESET	sysLib.rollback	No special considerations.

PSBs

PSBs are not supported in this release of EGL.

Control parts

In VisualAge Generator, control parts are entered using a free-form text editor. The control parts are not validated until they are actually used during generation. Whether something is in upper or lower case is not significant. In EGL, control parts are stored in .eglbld files in XML notation, with a special editor for each type of control part. In EGL, upper and lower case *are* significant. The tables in this section compare the information you enter in the VisualAge Generator free-form text editor with the XML tag or attribute value that is used in EGL. The tables only show the tag or attribute values, not the actual XML syntax.

Note:

- Where possible, the migration tool migrates generation options, linkage table options, and resource association options for runtime environments that are not supported at this time, but which are currently planned for support. This preserves as much of your information as possible in as useful a way as possible. For example, most IMS generation options are migrated even though IMS is not a currently supported runtime environment. These options are not displayed when you use the normal EGL Build Part Editor. However, you can see the options if you open the file with the Text Editor.
- The migration tool includes as comments those generation options, linkage table options, and resource association options that have no corresponding EGL replacement but which might be useful to you in determining related information that is required for EGL. For example, the /MFSDEV generation option that is used for the IMS environment cannot be migrated at this time. The migration tool includes the option as a comment to preserve the information for future use. These comments are not displayed when you use the normal EGL Build Part Editor. However, you can see the comments if you open the file with the Text Editor.
- The migration tool eliminates generation options that have no corresponding EGL replacement if the information is not useful in determining current or future EGL options. For example, there is no replacement for */lineinfo*, which was an option to assist IBM support in debugging the VAGen generator. This option is not useful for the EGL generator, so the migration tool does not include it as a comment.

- The migration tool does not rename control parts. The tool issues an error message of the part name conflicts with an EGL reserved word.

The control parts section is organized into the following tables:

Table 120. General control part information

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
VAGen control part names: • Can include the period (.) in the name. • For bind and linkedit parts, any portion of the name after the first period is treated as a suffix. The suffix can be specified in the /bind and /linkedit generation options.	EGL build parts: • The period (.) is not valid in a build part name.	The migration tool changes the period (.) to an underscore (_).
Upper and lower case are not significant in VAGen control part tags and values.	Upper and lower case are significant in EGL control part tags and values.	The migration tool converts the control part tags and values to the correct case required for EGL.

Generation options part

Table 121. Generation options

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
VAGen generation options part: • Contains one or more generation options. • Can be chained using the /options generation option. • Can reference any other control part that is included in the workspace at generation time. The referenced control parts are **not** considered to be associates of the generation options part.	EGL build descriptor part: • Contains one or more build descriptor options. • Can be chained using the *nextBuildDescriptor* build descriptor option • Can only reference other build parts where one of the following is true: – The build parts are included in the same *.eglbld* file. – The build parts are in files that are imported by the *.eglbld* file.	If your VAGen control parts are all in the same VisualAge Java package or VisualAge Smalltalk application, the control parts will all be placed in the same *.eglbld* file. In this situation, no import statements are required. If your VAGen control parts are in different VisualAge Java packages or VisualAge Smalltalk applications, the migration tool does not create the import statements. You will need to add the import statements. There will be an error in the Problems view if EGL is unable to resolve references to other control parts.

Table 121. Generation options (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
VAGen generation option values are only enclosed in quotes if they contain special characters for a directory or file name.	EGL build descriptor option values must be enclosed in quotes. However, if you use the EGL Build Parts Editor, the editor automatically inserts the quotes for you into the XML source. You do not see the quotes in the editor.	The migration tool includes the quotes automatically when it builds the XML source for the *.eglbld* file.
Many VAGen generation options can be specified as /xxxx or /noxxxx to reflect the positive or negative of the generation option. The following is an example: • */prep* indicates that you want the preparation step to be automatically started immediately after generation. • */noprep* indicates that you do not want the preparation step to be started automatically because you plan to run it at a later time.	Many EGL build descriptor options can be specified as xxxx="YES" or xxxx="NO" to reflect the positive or negative of the build descriptor option. The following is an example: • *prep="YES"* indicates that you want the preparation step to be automatically started immediately after generation. • *prep="NO"* indicates that you do not want the preparation step to be started automatically because you plan to run it at a later time.	The migration tool processes the options as follows: • The migration tool converts /xxxx to the corresponding xxxx="YES" option unless otherwise indicated. • The migration tool converts /noxxxx to the corresponding xxxx="NO" option unless otherwise indicated.
/ansisql	Not supported.	The migration tool includes this option as a comment.
/bidicontable=xxxx	bidiConversionTable="xxxx"	No special considerations.
/bind=xxxx In VisualAge Generator, xxxx is the suffix of the bind part. The bind part for a program is named pgmname.xxxx, where xxxx is the suffix specified by the /bind option. The reasons you might have specified a /bind=suffix are as follows: 1. A special bind is needed for the program because you bind the program into multiple DB2 plans, and 2. VisualAge Generator did not enable you to easily create a bind part with exactly the same name as the program.	bind="xxxx" The meaning of the bind option is not the same as in VisualAge Generator. In EGL, xxxx is the full name of the bind part. The bind option only needs to be specified if the bind part name differs from that of the program. In most cases, the program and bind part will have the same name, so there is no need to include the bind option. The bind option is only necessary if you generate the same program for multiple runtime environments and require special bind commands for each environment. Another use for the bind option is to specify the name of a part that contains a template for your bind command. A project administrator or DBA can define a bind part that includes substitutable SYMPARMS for member-specific parameters. You can use the EGL bind option to point to this template part. This technique works well if you bind a package for each program.	Because the value for bind has different meanings in VisualAge Generator and EGL, the migration tool cannot migrate this option. The migration tool includes /bind as a comment.

Table 121. Generation options (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
/checktype=*xxxx* *xxxx* is one of the following: • none • low • all	checkType="*xxxx*" *xxxx* is one of the following: • NONE • LOW • ALL	No special considerations.
/cicsdbcs	Not supported.	The migration tool does not include this option as a comment because all supported CICS translators now include support for DBCS.
/cicsentries=*xxxx* *xxxx* is one of the following: • none • rdo • macro	cicsEntries="*xxxx*" xxxx is one of the following: • NONE • RDO • MACRO	No special considerations.
/cobollevel=le | vs	Not supported.	The migration tool includes this option as a comment.
commentlevel=*n* or /commentlevel=*commentText* *n* or *commentText* are one of the following: • 0 or minimum • 1 or info • 2 or logic • 3 or data • 4 or statements **Note:** • Either the numeric value or its equivalent *commentText* can be specified. • 0 = genoption comments only • 1 = alias names, standard generation information • 2 = program and table prolog, and function descriptions • 3 = record prologs and data item descriptions • 4 = source statements and comments • For C++, the only valid values are 0 = none and 1 = comments	commentLevel="*n*" *n* is one of the following: • 0 • 1 • 1 • 1 • 1 **Note:** • 0 = no comments • 1 = comments are included	The migration tool migrates /commentlevel=0 or minimum to 0 and all other values to 1.
/configmapname="*xxxx*" *xxxx* is the name of a VisualAge Smalltalk configuration map.	Not supported.	The migration tool includes this option as a comment because it might be useful in determining groups of related EGL projects that should be checked into your source code repository as a unit.

Table 121. Generation options (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
`/configmapversion="xxxx"` *xxxx* is the version name of the VisualAge Smalltalk configuration map specified by /configmapname.	Not supported.	The migration tool includes this option as a comment because it might be useful in determining groups of related EGL projects that should be checked into your source code repository as a unit.

`/contable=xxxx`
 	
xxxx is the name of a conversion table.
	`clientCodeSet="yyyy"`
	`serverCodeSet="zzzz"`
	
	yyyy and *zzzz* are the names of the client and server conversion tables, respectively.
		The migration tool sets both the *clientCodeSet* and the *serverCodeSet* options from the VAGen /contable generation option. See Table 122 on page 262 for the correspondence between the VAGen and EGL values. If the value for /contable=xxxx is not in Table 122 on page 262, the migration tool sets both clientCodeSet and serverCode to *xxxx*.

/createdds	genDDSFile=″YES″ | ″NO″ **Note:** This is for ISERIESC.	No special considerations.
/currency=xxx (1 to 3 characters)	currencySymbol=″xxx″	No special considerations.
/data = 24 | 31	data=″24″ | ″31″	No special considerations.
`/dbms=xxxx` *xxxx* is one of the following: • db2 • oracle • odbc	`dbms="xxxx"` *xxxx* is one of the following: • DB2 • ORACLE • DB2 **Note:** EGL Java generation supports JDBC instead of ODBC.	The migration tool changes *odbc* to *DB2* and issues a warning message.
/dbpassword=xxxx	sqlPassword=″xxxx″	The migration tool merges the VAGen /dbpassword and /sqlpassword options into the EGL *sqlPassword* option. If a generation options part includes both /dbpassword and /sqlpassword, the migration tool includes the sqlPassword twice. There should be an error in the Problems view.
/dbuser=xxxx	sqlID=″xxxx″	The migration tool merges the VAGen /dbuser and /sqlID options into the EGL *sqlID* option. If a generation options part includes both /dbuser and /sqlID, the migration tool includes the sqlID twice. There should be an error in the Problems view.
/debugtrace	debugTrace=″YES″ | ″NO″	No special considerations.
/destaccount=xxxx	Not supported.	The migration tool includes this option as a comment.
/destdir=xxxx	destDirectory=″xxxx″	No special considerations.
/desthost=xxxx	destHost=″xxxx″	No special considerations.

Table 121. Generation options (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
/destlib=xxxx	destLibrary="xxxx" **Note:** This is for ISERIESC.	No special considerations.
/destpassword=xxxx	destPassword="xxxx"	No special considerations.
/destuid=xxxx	destUserID="xxxx"	No special considerations.
/dxfrcancel	cancelAfterTransfer="YES" | "NO"	No special considerations.
/dxfrxctl	useXctlForTransfer="YES" | "NO"	No special considerations.
/ejbgroup=xxxx	Not supported.	The migration tool includes this option as a comment.
/endcommarea	endCommarea="YES" | "NO"	No special considerations.
/errdest=xxxx	errorDestination="xxxx" **Note:** This is for IMS.	No special considerations.
/fastpath	imsFastPath="YES" | "NO" **Note:** This is for IMS.	No special considerations.
/fold	Not supported.	The migration tool includes this option as a comment.
/ftptranslationcmddbcs=xxxx	Not supported. EGL only supports TCP/IP for transferring files to the host.	The migration tool includes this option as a comment.
/ftptranslationcmdsbcs=xxxx	Not supported. EGL only supports TCP/IP for transferring files to the host.	The migration tool includes this option as a comment.
/genauthortimevalues	Not supported.	The migration tool includes this option as a comment.
/genhelpmaps	genHelpFormGroup="YES" | "NO"	No special considerations.
/genmaps	genFormGroup="YES" | "NO"	No special considerations.
/genout=xxxx	genDirectory="xxxx"	No special considerations.

`/genproperties`
`/nogenproperties`
	`genProperties="GLOBAL"`
	`genProperties="NO"`
	
	EGL also provides genProperties="PROGRAM".
		The migration tool converts /genproperties to the EGL *genProperties*="GLOBAL" because this is the closest value in terms of what is generated.

/genresourcebundle	Not supported.	The migration tool includes this option as a comment.
/genret	genReturnImmediate="YES" | "NO"	No special considerations.
/gentables	genDataTables="YES" | "NO"	No special considerations.
/genuirecords	genUIRecords="YES" | "NO" **Note:** This is for UI records only.	No special considerations.
/groupname=xxxx	Not supported.	The migration tool includes this option as a comment.
`/inedit=all` `/inedit=inonly`	`validateOnlyIfModified="NO"` `validateOnlyIfModified="YES"`	No special considerations.

Table 121. Generation options (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
/initaddws In VisualAge Generator, the primary working storage record is always initialized. The /initaddws generation option provides initialization of other working storage records specified on the Tables and Additional Records list.	initNonIOData="YES" | "NO" In EGL, the record specified as the inputRecord for main programs is always initialized. The *initNonIOData* build descriptor option provides initialization of other basic records specified in the data declarations for the program. In addition, EGL enables you to specify the initialized property for any record declaration in the program. The initialized property provides a finer control than the initNonIOData build descriptor option.	The migration tool migrates the primary working storage record for main programs to the EGL *inputRecord* property and also includes the record declaration without the initialized property. The migration tool migrates the primary working storage record for called programs to a record declaration with the initialized property. If the migration tool created an additional record for level 77 items, the migration tool includes a data declaration for the record and also includes the initialized property. This provides the same initialization for primary working storage records as in VisualAge Generator. All other basic records are initialized based on the *initNonIOData* build descriptor option.
/initrecd	initIORecords=″YES″ | ″NO″	No special considerations.
/javadestdir=xxxx	Not supported.	The migration tool includes this option as a comment.
/javadesthost=xxxx	Not supported.	The migration tool includes this option as a comment.
/javadestpassword=xxxx	Not supported.	The migration tool includes this option as a comment.
/javadestuid=xxxx	Not supported.	The migration tool includes this option as a comment.
/javasystem=xxxx	Not supported.	The migration tool includes this option as a comment.
/jobcard=xxxx	Not supported. The MVS build server handles the jobcard.	The migration tool includes this option as a comment.
/jobname=xxxx	Not supported. However, you can use $USERID as the job name in the build script. EGL generation substitutes $USERID with the value from the destUserID build descriptor option concatenated with a number to provide a unique job name. **Note:** The VAGen /destuid generation option migrates to the EGL destUserID.	The migration tool includes this option as a comment.
/jspreldir=″xxxx″	Not supported. The JSP is no longer generated from EGL source.	The migration tool includes this option as a comment.
/leftjust	leftAlign=″YES″ | ″NO″	No special considerations.
/lineinfo	Not supported.	The migration tool does not include this option as a comment because the option was only meanighful for IBM support to debug the VAGen generator. It had no effect on the generated COBOL.

Table 121. Generation options (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
/lines=nn	Not supported.	The migration tool includes this option as a comment.
/linkage=*xxxx* *xxxx* is the name of a VAGen linkage table part.	linkage="*xxxx*" *xxxx* is the name of an EGL linkage options part.	No special considerations.
/linkedit=*xxxx* In VisualAge Generator, *xxxx* is the suffix of the linkedit part. The linkedit part for a program is named pgmname.xxxx, where xxxx is the suffix specified by the /linkedit option. The reasons you might have specified a /linkedit=suffix are as follows: 1. A special linkedit is needed for the program such as for static linkedit to PL/I, and 2. VisualAge Generator did not enable you to easily create a linkedit part with exactly the same name as the program.	linkEdit="*xxxx*" The meaning of *linkEdit* is not the same as in VisualAge Generator. In EGL, *xxxx* is the full name of the linkedit part. The linkEdit option only needs to be specified if the linkedit part name differs from that of the program. In most cases, the program and linkedit part will have the same name, so there is no need to include the linkEdit option. The linkEdit option is only necessary if you generate the same program for multiple runtime environments.	Because the value for linkedit has different meanings in VisualAge Generator and EGL, the migration tool cannot migrate this option. The migration tool includes /linkedit as a comment.
/listing /listingonerror /nolisting **Note:** This is a three-way switch.	Not supported.	The migration tool includes this option as a comment.
/locvalid	Not supported.	The migration tool includes this option as a comment.
/log=xx OR /nolog	imsLogID="xx" OR include /nolog as a comment **Note:** This is for IMS.	The migration tool processes this option as follows: • /log=xx is converted to imsLogID=″xx″ • /nolog is converted to a comment.
/math=*xxxxx* *xxxxx* is one of the following: • cobol • cspae	math="*xxxxx*" *xxxxx* is one of the following: • COBOL • CSPAE	No special considerations.
/mfsdev=*list-of-devices* *list-of-devices* provides device information for generating VAGen maps into MFS source code for the IMS environment.	Not currently supported. **Note:** This is for IMS.	The migration tool includes this option as a multi-line comment.

Table 121. Generation options (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
/mfseattr /nomfseattr /mfseattrncd In VisualAge Generator, these 3 options provide a 3-way switch to give information that is needed to generate extended attribute support for maps in MFS format.	mfsExtendedAttr="YES" mfsExtendedAttr="NO" mfsExtendedAttr="NCD" **Note:** This is for IMS.	No special considerations.
/mfsignore	mfsIgnore=″YES″ | ″NO″ **Note:** This is for IMS	No special considerations.
/mfstest	mfsUseTestLibrary=″YES″ | ″NO″ **Note:** This is for IMS	No special considerations.
/msgtableprefix=*xxxx* In VisualAge Generator, the message table prefix is specified on the program. When you generate the UI record by itself you must specify the message table prefix during generation.	Not supported.	The migration tool includes this option as a comment.
/msp=*xxxx* *xxxx* is one of the following: • all • gsam • mfs • seq	formServicePgmType="*xxxx*" *xxxx* is one of the following: • ALL • GSAM • MFS • SEQ **Note:** This is for IMS.	No special considerations.
/nullfill	fillWithNulls=″YES″ | ″NO″	No special considerations.
/numovfl	checkNumericOverflow=″YES″ | ″NO″	No special considerations.
/options=*xxxx* *xxxx* is the name of another VAGen generation options part.	nextBuildDescriptor="*xxxx*" *xxxx* is the name of another EGL build descriptor part.	No special considerations.
/packagename=xxxx	Not supported.	The migration tool includes this option as a comment.
/possign=*x* *x* is one of the following: • f • c	positiveSignIndicator="*x*" *x* is one of the following: • F • C **Note:** This is for ISERIESC.	No special considerations.
/prep	prep =″YES″ | ″NO″	No special considerations.
/prepfile	buildPlan=″YES″ | ″NO″	No special considerations.

Table 121. Generation options (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
/printdest=*xxxx* *xxxx* is one of the following: • ezeprint • termid	printDestination="*xxxx*" *xxxx* is one of the following: • PROGRAMCONTROLLED • TERMINALID	No special considerations.
/project="*xxxx*"[,"*version*"] *xxxx* is the name of a VisualAge for Java project, and *version* is the version name for the specified project.	Not supported.	The migration tool includes this option as a comment because it might be useful in determining groups of related EGLprojects that should be checked into your source code repository as a unit.
/projectid=xxxx	projectID=″xxxx″	No special considerations.
/recovery	restoreCurrentMsgOnError=″YES″ | ″NO″ **Note:** This is for IMS.	No special considerations.
/resource=*xxxx* *xxxx* is the name of a VAGen resource associations part.	resourceAssociations="*xxxx*" *xxxx* the name of an EGL resource associations part.	No special considerations.
/resourcebundlelocale=xxxx	Not supported. In EGL this information is specified on the PageHandler part.	The migration tool includes this option as a comment.
/resvword=xxxx	reservedWord=″xxxx″	No special considerations.
/rt=xxxx	returnTransaction=″xxxx″	No special considerations.
/runfile	genRunFile=″YES″ | ″NO″	No special considerations.
/sendtranslationcmddbcs=xxxx	Not supported. EGL only supports TCP/IP for transferring files to the host.	The migration tool includes this option as a comment.
/session=xxxx	Not supported. **Note:** EGL only supports TCP/IP for transferring files to the host.	The migration tool includes this option as a comment.
/setfull	setFormItemFull=″YES″ | ″NO″	No special considerations.
/sp	checkToTransaction=″YES″ | ″NO″	No special considerations.
/spa=xxxx	spaSize=″nnnn″ **Note:** This is for IMS.	No special considerations.
/spzero	spacesZero=″YES″ | ″NO″	No special considerations.
/sqldb=xxxx	sqlDB=″xxxx″	No special considerations.
/sqlid=xxxx	sqlID=″xxxx″	The migration tool merges the VAGen /dbuser and /sqlID options into the EGL *sqlID* option. If a generation options part includes both /dbuser and /sqlID, the migration tool includes the sqlID twice. There should be an error in the Problems view.

Table 121. Generation options (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
/sqlpassword=xxxx	sqlPassword="xxxx"	The migration tool merges the VAGen /dbpassword and /sqlpassword options into the EGL *sqlPassword* option. If a generation options part includes both /dbpassword and /sqlpassword, the migration tool includes the sqlPassword twice. There should be an error in the Problems view.
/sqlvalid	validateSQLStatements="YES" | "NO"	No special considerations.
`/symparm=`*`pppppppp`*`,'`*`vvvv`*`'` • *pppppppp* is the name of the symbolic parameter. *pppppppp* is 1 - 8 characters. • *vvvv* is the value. Two consecutive single-quotes within the value is one single-quote.	EGL supports many of the same predefined symbolic parameters as VisualAge Generator. You can also use any user-defined symbolic parameters as long as they do not conflict with any of the new EGL symbolic parameters.	The migration tool processes symbolic parameters as follows: • The migration tool converts any VAGen-defined symbolic parameters to the corresponding EGL symbolic parameter. • If there is no corresponding EGL symbolic parameter, VisualAge Generator migrates to the syntax of an EGL symbolic parameter without changing the parameter name or value. The migration tool also issues an error message. • The migration tool converts any user-defined symbolic parameters to the syntax of an EGL symbolic parameter without changing the parameter name or value.
/SYMPARM=EZALTXTR,'xxxx'	transferErrorTransaction="xxxx"	No special considerations.
/SYMPARM=EZONEAS2,'xxxx'	oneFormItemCopybook="YES"	No special considerations.
/syncdxfr	synchOnPgmTransfer="YES" | "NO" **Note:** This is for DL/I for the CICS environment.	No special considerations.
/syncxfer	synchOnTrxTransfer="YES" | "NO"	No special considerations.
/syscodes	sysCodes="YES" | "NO"	No special considerations.

Table 121. Generation options (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
`/system=xxxx` *xxxx* is one of the following: • MVSBATCH • MVSCICS • AIX • JAVAWRAPPER • WINNT • LINUX • OS400 The following environments can also be specified in VAGen, but not in EGL: IMSBMP, IMSVS, JAVA, HP, JAVAGUI, WINGUI, OS2GUI, OS2, OS2CICS, AIXCICS, NTCICS, SOLARIS, SOLACICS, TSO, VMCMS, VMBATCH, VSEBATCH. VSECICS	`system="xxxx"` *xxxx* is one of the following: • ZOSBATCH • ZOSCICS • AIX • WIN • WIN • LINUX • ISERIESC	The migration tool processes this option as follows: • If /systemxxxx has a corresponding value in EGL, the migration tool migrates to the corresponding EGL value. • If /system=xxxx does not have a corresponding value in EGL, the migration tool includes /system=xxxx as a comment. • For /system=JAVAWRAPPER, the migration tool also includes the EGL build descriptor option *enableJavaWrapperGen="ONLY"*. This specifies that the you want to generate only the Java wrapper for a program. • For /system=MVSBATCH or /system=MVSCICS, the migration tool issues a warning message that you need to set the *destPort* build descriptor option.
`/targnls=xxx` *xxx* is a 3-character national language code.	`targetNLS="xxx"` *xxx* is the 3-character national language code. All the values except ENP (uppercase English) are identical in VisualAge Generator and EGL. ENP does not have a counterpart in EGL.	The migration tool uses the VAGen value as the targetNLS value. If the value is ENP, there will be an error in the Problems view. You can edit the .eglbld file and change the value. You might want to use ENU (mixed case English) as a replacement for ENP.
`/templates=xxxx` In VisualAge Generator, templates are used to generate the preparation and runtime JCL, as well as to generate CICS transaction and program entries.	`templateDir="xxxx"` In EGL, build scripts replace build templates. The only templates that are used are to produce runtime JCL for the ZOSBATCH and ISERIESC target environment.	No special considerations.
`/trace=xxxx,yyyy` *xxxx* is one of the following: • none • sqlerr • sqlio *yyyy* is optional. If *yyyy* is present, it is set to *stmt*. Any combination of *none, sqlerr,* or *sqlio*, with or without *,stmt* , is valid.	/trace splits into multiple build descriptor options: • If sqlerr is included -- sqlErrorTrace="YES" • If sqlio is included --- sqlIOTrace="YES" • if stmt is included --- statementTrace="YES"	No special considerations.
`/transfertype=xxxx` *xxxx* is one of the following: • tcpip • sna	Not supported. EGL only supports TCP/IP for transferring files to the host.	The migration tool includes this option as a comment.

Table 121. Generation options (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
`/transid=primaryID,restartID` In VisualAge Generator, */transid=,restart* is valid, with the primary transaction defaulting to the first 4 characters of the program name.	/transid splits into multiple build descriptor options: • If *primary* is included -- startTransactionID=*"primaryID"* • If *,restart* is included --- restartTransactionID=*"restartID"*	No special considerations.
/twaoff=nnnn	twaOffset="nnnn"	No special considerations.
`/unload` In VisualAge Generator /unload directed batch generation to unload all VisualAge Java projects or VisualAge Smalltalk configuration maps that contained VAGen parts before loading the projects or configuration maps being requested for the current generation process.	Not supported.	The migration tool does not include a comment for this option.
/validmix	validateMixedItems="YES" | "NO"	No special considerations.
/vmloadlib=xxxx	Not supported.	The migration tool includes this option as a comment.
/vselib=xxxx	Not supported.	The migration tool includes this option as a comment.
`/workdb=xxxx` *xxxx* is one of the following: • aux • main • dli • sql	`workdb="xxxx"` *xxxx* is one of the following: • AUX • MAIN • DLI • SQL **Note:** The values for DLI and SQL are for IMS and are not currently supported in EGL.	The migration tool processes this option as follows: • If /workdb=aux or /workdb=main, the migration tool converts /workdb to the corresponding EGL value. • If /workdb=dli or /workdb=sql, the migration tool includes this option as a comment.
Not used.	vagCompatibility="YES"	The migration tool always adds this option to every build descriptor part.
In VisualAge Generator, *decimalSymbol* was a runtime property used when assigning or comparing CHA and NUM values in Java.	`decimalSymbol="x"` *x* is one of the following: • a period (.) • a comma (,) In EGL, you can specify this information at generation time or at runtime.	The migration tool does not set the *decimalSymbol*. You can add this property to your build descriptor part if you plan to generate Java source. If you do this, the decimalSymbol will be generated into any EGL properties files. Alternatively, you can add the property directly to the generated EGL properties file.

Table 121. Generation options (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
Not used.	`destPort="xxxx"` In EGL, *destPort* specifies the port to use when transferring generation outputs to a host system to prepare them for execution. The destPort build descriptor option is required for the ZOSCICS and ZOSBATCH target environments.	The migration tool does not set *destPort* and there is no default value. However, the sample JCL for starting an MVS build server uses the default port of 5555.
Not supported. In VisualAge Generator, even if your program checks the value of EZESYS, all the VAGen source code must be valid for every target environment for which you might generate the program.	`eliminateSystemDependentCode= "YES" | "NO"` In EGL, if your program checks the value of *systemType*, you can choose to omit source code that can never be run for your current target generation environment. This can make the resulting COBOL or Java source code smaller.	The migration tool does not set *eliminateSystemDependentCode*. The default value is ″YES″.
Not used.	sessionBeanID=″xxxx″	The migration tool does not set *sessionBeanID*. If you are generating Java or Java wrappers, see the EGL online helps to determine if you need to set the sessionBeanID build descriptor option.

In VisualAge Generator, you include the SQL JDBC driver class, JNDI name, and connection URL information in a properties file that is used at runtime.	`sqlJDBCDriverClass="xxxx"`
	`sqlJNDIName="xxxx"`
	`sqlValidationConnectionURL="xx"`
	
	In EGL, you can specify this information at generation time or at runtime.
		The migration tool does not set the *sqlJDBCDriverClass, sqlJNDIName,* and *sqlValidationConnectionURL*. If you want to specify these values at generation time, you can do the following:
		• Specify workspace preferences. This technique only works if you are generating in the Eclipse environment.
		• Specify the build descriptor options in your build descriptor parts. This technique works when you generate in the Eclipse environment as well as when you generate in batch.
		
		In either case, you must also include the *genProperties=″YES″* build descriptor option so that the properties file will be generated.
		
		If you want to specify the value at runtime, you can modify the runtime properties in either the J2EE deployment descriptor or a properties file.

Not supported.	cicsj2cTimeout=″nnnn″	The migration tool does not set the *cicsj2cTimeout* value. If you are generating Java, see the EGL online helps to determine if you need to set the *cicsj2cTimeout* build descriptor option.

Table 122. Generation options - conversion table values

Language	**Conversion Table VAGen** */contable* **value**	**EBCDIC Character Set EGL** *serverCodeSet*	**ASCII Character Set EGL** *clientCodeSet*
Arabic	ELACNARA	IBM-420	IBM-1256
Chinese, simplified	ELACNCHS	IBM-935	IBM-1381
Chinese, simplified	ELACNGBK	IBM-1388	IBM-1386
Chinese, traditional	ELACNCHT	IBM-937	IBM-950
Danish	ELACNDKN	IBM-277	IBM-1252
Eastern European	ELACN870	IBM-870	IBM-1250
English (UK)	ELACN285	IBM-285	IBM-1252
English (US)	ELACNENU	IBM-037	IBM-1252
Finnish	ELACNFIN	IBM-298	IBM-1252
French	ELACNFRA	IBM-297	IBM-1252
German	ELACNDEU	IBM-273	IBM-1252
Greek	ELACNGRE	IBM-875	IBM-1253
Hebrew	ELACNHEB	IBM-424	IBM-1255
Italian	ELACNITA	IBM-280	IBM-1252
Japanese, Katakana	ELACNJPN	IBM-930	IBM-943
Japanese, Latin	ELACNJPL	IBM-939	IBM-943
Korean	ELACNKOR	IBM-933	IBM-949
Norwegian	ELACNDKN	IBM-277	IBM-1252
Portuguese	ELACNPTB	IBM-037	IBM-1252
Russian	ELACNCYR	IBM-1025	IBM-1251
Spanish	ELACNESP	IBM-284	IBM-1252
Swedish	ELACNSWE	IBM-278	IBM-1252
Swiss German	ELACNDES	IBM-500	IBM-1252
Turkish	ELACNTUR	IBM-1026	IBM-1254
User–defined (not in the above list)	XXXXXXXX	XXXXXXXX	XXXXXXXX

Linkage table parts

The linkage table parts are Calllink, Filelink, Crtxlink, and Dxfrlink.

callLink

Table 123. Linkage table options for :calllink

VisualAge Generator 4.5	**EGL produced by the migration tool**	**Migration tool considerations**
:calllink	callLink	No special considerations.

Table 123. Linkage table options for :calllink (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
`linktype=xxxx` *xxxx* is one of the following: • dynamic • static • cicslink • remote • csocall • sessionejb	Type of call, where the EGL equivalent options are the following: • localCall • localCall • localCall • remoteCall • remoteCall • ejbCall	If the VAGen linktype is omitted, the migration tool uses *localCall*. The migration tool also uses linktype in additional places to set other properties for the EGL CallLink information.
`applname=programName` *programName* is the name of the program being called. Wildcards are permitted.	pgmName="programName"	No special considerations.
externalname=applname	alias="applname"	If your VAGen program had to be renamed because the name was an EGL reserved word, you can use the *alias* property either on the program definition or in the linkage table to provide the original VAGen name for the program as the name of the generated program. Either technique can help you avoid having to modify non-VAGen programs that call the VAGen program.
package=packageName	package="packageName"	If you generate Java and the calling and called programs are in different packages, you can include the package name in the linkage entry for the called program. Alternatively, change the CALL statement to explicitly qualify the program with the package name or include an import statement for the package in the file that contains the CALL statement.

`library=libraryName`
 	
OR
 	
`dllname=libraryName`
 	
In VisualAge Generator, library and dllname are treated as synonyms.
	library="libraryName"	The migration tool merges the VAGen library or dllname into the EGL library property.

`linktype=xxxx` *xxxx* is one of the following: • dynamic • static • cicslink	`linkType="xxxx"` *xxxx* is one of the following: • DYNAMIC • STATIC • CICSLINK	No special considerations.

Table 123. Linkage table options for :calllink (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
`parmform=xxxx` *xxxx* is one of the following: • oslink • commptr • commdata • cicsoslink	`parmForm="xxxx"` *xxxx* is one of the following: • OSLINK • COMMPTR • COMMDATA • CICSOSLINK	No special considerations.
`contable=xxxx` *xxxx* is one of the following: • a conversionTableName • * • EZECONVT • BINARY • NONE	`conversionTable="xxxx"` *xxxx* is one of the following: • converstionTableName • * • PROGRAMCONTROLLED • not supported • not supported	The migration tool uses the same *conversionTableName* when creating the EGL CallLink information. The migration tool migrates the VAGen contable=BINARY to BINARY, which is an unsupported value in EGL. The migration tool also issues an error message. There will be an error in the Problems view. You must correct the error by editing the *.eglbld* file and selecting the supported value that you want to use. The migration tool omits the *conversionTable* property if the VAGen contable=NONE.
`location=xxxx` *xxxx* is one of the following: • systemName • EZELOC	`location="xxxx"` *xxxx* is one of the following: • systemName • PROGRAMCONTROLLED	No special considerations.

Table 123. Linkage table options for :calllink (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
`remotecomtype=xxxx` *xxxx* is one of the following: • appcims • ca400 • cicsclient • dce • dcesecure • direct • exci • ipc • java400 • lu2 • tcpip	`remoteComType="xxxx"` *xxxx* is one of the following: • not supported • not supported • CICSECI • not supported • not supported • DIRECT • not supported • DISTINCT • JAVA400 • not supported • TCPIP	The migration tool converts cicsclient to CICSECI because that is the closest corresponding EGL value. If the VAGen :calllink entry did not already specify the ctgport and ctglocation, the migration tool issues an error message to remind you to specify these values. The migration tool migrates the values listed as not supported ″as is″ and issues a message. You must determine what communications protocol you want to use now and then update the EGL CallLink entry with the correct information. There will be an error in the Problems view until you correct the CallLink information. If you decide to use CICSSSL, you must add the *ctgPort, ctgLocation, ctgKeyStore*, and *ctgKeyStorePassword* properties to the EGL CallLink information. If you decide to use CICSJ2C, you must add the pgmName, conversionTable, remotePgmType, luwControl, remoteBind, location, and parmForm properties to the EGL CallLink information.
`remoteapptype=xxxx` *xxxx* is one of the following: • vg • nonvg • vgjava • itf	`remotePgmType="xxxx"` *xxxx* is one of the following: • EGL • EXTERNALLYDEFINED • not applicable • not supported	If the VisualAge Generator remoteapptype=vgjava, the migration tool migrates the :calllink entry, but omits the *remotePgmType* property. If remoteapptype=itf, the migration tool turns the entire :calllink entry into a comment.
serverid=serverName	serverID=″serverName″	No special considerations.
`luwcontrol=xxxx` *xxxx* is one of the following: • client • server	`luwControl="xxxx"` *xxxx* is one of the following: • CLIENT • SERVER	No special considerations.
`remotebind=xxxx` *xxxx* is one of the following: • generation • runtime	`remoteBind="xxxx"` *xxxx* is one of the following: • GENERATION • RUNTIME	No special considerations.
providerURL=URLName	providerURL=″URLName″	No special considerations.
ctglocation=’tcpipInfo’	ctgLocation=″tcpipInfo″	No special considerations.
ctgport=portID	ctgPort=″portID″	No special considerations.

Table 123. Linkage table options for :calllink (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
`bitmode=nn` *nn* is one of the following: • 16 • 32	Not supported.	The migration tool includes this option as a comment.
`binform=xxxx` *xxxx* is one of the following: • intel • host	Not supported.	The migration tool includes this option as a comment.
Not supported. In VisualAge Generator, you specify the NOMAPS option on a CALL statement to achieve better performance if the called program does not send any maps to the screen.	refreshScreen="YES" | "NO"	The migration tool does not set this property. If you previously specified NOMAPS for a VAGen call statement, you can continue to use the *noRefresh* option on the EGL CALL statement if you use the *vagCompatibility="YES"* build descriptor option. Alternatively, you can obtain the same support by specifying *refreshScreen="NO"* on the CallLink entry for the called program.
Not used. None of the communication protocols supported by VisualAge Generator required this information.	`ctgKeyStore` `ctgKeyStorePassword`	The migration tool does not set this property. *ctgKeyStore* and *ctgKeyStorePassword* are required if you decide to use *remoteComType="CICSSSL"*.
Not used. In VisualAge Generator, you use the /system=JAVAWRAPPER generation option whenever you want to generate a Java wrapper for a called batch program.	javaWrapper="YES" | "NO"	The migration tool does not set this property. You must specify *javaWrapper="YES"* if you want a Java wrapper to be generated whenever you generate the called program.

fileLink

Table 124. Linkage table options for :filelink

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
:filelink	fileLink	No special considerations.
`linktype=xxxx` *xxxx* is one of the following: • local • remote In VisualAge Generator, the default is local.	Type of file, where the EGL equivalent options are as follows: • localFile • remoteFile	.If the VAGen linktype is not specified, the migration tool converts to *localFile*.
`filename=fileName` *fileName* is the name of a file in a VAGen record definition. Wildcards are permitted.	fileName="*fileName*"	No special considerations.

Table 124. Linkage table options for :filelink (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
`contable=xxxx` *xxxx* is one of the following: • a conversionTableName • * • EZECONVT • BINARY	`conversionTable="xxxx"` *xxxx* is one of the following: • conversionTableName • * • PROGRAMCONTROLLED • not supported	The migration tool uses the same *conversionTableName* when creating the EGL FileLink information. The migration tool migrates the VAGen contable=BINARY to BINARY, which is an unsupported value in EGL. The migration tool also issues an error message. There will be an error in the Problems view. You must correct the error by editing the *.eglbld* file and selecting the supported value that you want to use.
`location=xxxx` *xxxx* is one of the following: • CICS • EZELOC	`locationSpec="xxxx"` *xxxx* is one of the following: • CICS • PROGRAMCONTROLLED	No special considerations.

Crtxlink

Table 125. Linkage table options for :crtxlink

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
:crtxlink	asynchLink	No special considerations.
`linktype=xxxx` *xxxx* is one of the following: • local • remote **Note:** In VisualAge Generator the default is local.	Type of file, where the EGL equivalent options are the following: • localAsynch • remoteAsynch	If the VAGen linktype is not specified, the migration tool converts to *localAsynch*.
`recdname=recordName` *recordName* is the name of a VAGen record definition. Wildcards are permitted.	recordName=*"recordName"*	No special considerations.
`contable=xxxx` *xxxx* is one of the following: • a conversionTableName • * • EZECONVT • BINARY	`conversionTable="xxxx"` *xxxx* is one of the following: • conversionTableName • * • PROGRAMCONTROLLED • not supported	The migration tool uses the same *conversionTableName* when creating the EGL AsynchLink information. The migration tool converts the VAGen contable=BINARY to BINARY, which is an unsupported value in EGL. The migration tool also issues an error message. There will be an error in the Problems view. You must correct the error by editing the *.eglbld* file and selecting the supported value that you want to use.

Table 125. Linkage table options for :crtxlink (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
`location=xxxx` *xxxx* is one of the following: • CICS • EZELOC	`locationSpec="xxxx"` *xxxx* is one of the following: • CICS • PROGRAMCONTROLLED	No special considerations.
package=packageName	package=″packageName″	No special considerations.

Dxfrlink

Table 126. Linkage table options for :dxfrlink

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
:dxfrlink	transferToProgram	No special considerations.
`fromappl=programName` *programName* is the name of the program that is transferring with a DXFR to another program. Wildcards are not permitted.	fromPgm=″*programName*″	No special considerations.
`toappl=programName2` *programName2* is the name of the program to which the transfer is occurring.	toPgm=″programName2″	No special considerations.
`linktype=xxxx` *xxxx* is one of the following: • dynamic • static • noncsp	`linkType="xxxx"` *xxxx* is one of the following: • DYNAMIC • STATIC • EXTERNALLYDEFINED	If you previously specified NONCSP for a VAGen DXFR statement, you can continue to use the *externallyDefined* option on the EGL transfer to program statement if you include *vagCompatibility=″YES″* in your build descriptor options. Alternatively, you can obtain the same support by specifying *linkType=″EXTERNALLYDEFINED″* on the *transferToProgram* entry for the program to which you are transferring.
Not supported.	alias=″applname″	If your VAGen program had to be renamed because the name was an EGL reserved word, you can use the *alias* property either on the program definition or in the linkage table to provide the original VAGen name for the program as the name of the generated program. Either technique can help you avoid having to modify non-VAGen programs that call the VAGen program.

Resource association part

Table 127. Resource association

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
In VisualAge Generator, a resource association part specifies how a file is to be implemented for a specific target environment. The file is the File Name specified in a VAGen record definition. The resource association part can also specify how print output is to be implemented for a specific target environment. When generating a program, the fileName for each indexed, serial, relative or print output is matched to the resource association part. The first entry that matches based on the fileName and generation target environment is the entry that is used for that file.	The EGL resource association part specifies how a file is to be implemented for a specific target environment. The file is the *fileName* property that is specified in an EGL record definition. The resource association part can also specify how print output is to be implemented for a specific target environment. When generating a program, the fileName for each indexed, serial, relative or print output is matched to the resource association part. The first entry that matches based on the fileName and generation target environment is the entry that is used for that file.	No special considerations.
For VisualAge Generator, resource association files are also used at runtime.	For EGL, resource association information is stored in EGL parts.	The migration tool includes support for converting additional options that were only valid in VAGen resource association files.

Table 127. Resource association (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
`/system=targetSystem` *targetSystem* is one of the following: • AIX * • AIXCICS * • HP-UX * • IMSBMP • IMSVS • LINUX ** • MVSBATCH • MVSCICS • NTCICS * • OS2 * • OS2CICS • OS400 • SCO * • SOLACICS * • SOLARIS * • TSO • VMCMS • VMBATCH • VSEBATCH • VSECICS • WINNT ** **Note:** • * — Indicates environments used for C++ generation. • ** — Indicates environments used for Java generation. • /system is optional. • VisualAge Generator supports an * as a wildcard in the target system. (For example, MVS* or *CICS).	This is the EGL target environment. The corresponding environment values are as follows: • aix • not supported • not supported • not supported • not supported • linux • zosbatch • zoscics • values will not be valid • not supported • not supported • iseriesc • not supported • not supported • not supported • not supported • not supported • not supported • not supported • not supported • win **Note:** Wildcards are not supported.	The migration tool processes the /system option as follows: • For a target system that is listed as not supported, the migration tool includes the information for the VAGen resource association entry as a comment in the EGL resource association part. This helps preserve as much of your information as possible. • If the /system option is omitted from the VAGen resource association entry, the migration tool uses *any* as the EGL resource association target environment. • If the /system option uses a wildcard, the migration tool migrates the option exactly as it is, including the wildcard (for example, mvs* or *cics). The migration tool also issues an error message.

Table 127. Resource association (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
`filetype=fileType` *fileType* is one of the following: • BTRIEVE • GSAM • IBMCOBOL • MFCOBOL • MMSGQ • MQ • OS2COBOL • SEQ • SEQRS • SMSGQ • SPOOL • TEMPAUX • TEMPMAIN • TRANSIENT • VSAM • VSAMRS	The EGL file type, where the corresponding values are in the following list: • not supported • not supported • ibmcobol • not supported • not supported • mq • not supported • seq or seqws • seqrs • not supported • spool • tempaux • tempmain • transient • vsam • vsamrs	The migraton tool processes the /filetype option as follows: • If the /system option specifies a host target environment, the migration tool converts the VAGen SEQ file type to the EGL *seq* file type. • If the /system option is a workstaton environment, the migration tool converts the VAGen SEQ file type to the EGL *seqws* file type. • If the /system is MVSBATCH and the filetype is GSAM, the migration tool migrates the resource association entry as a comment. • For all other unsupported file type values, if the resource association is for a /system that is supported, the migration tool creates an EGL resource association entry using the VAGen file type and issues an error message. There will also be an error in the Problems view. You must fix this error before you can use the EGL resource association part.
sysname=systemName	systemName=″systemName″	The migration tool converts any symbolic parameters that are used within the /system option to the corresponding EGL replacement symbolic parameter.
`/replace` `/noreplace`	`replace="YES"` `replace="NO"`	No special considerations.

`/dup`
`/nodup`
	`duplicates="YES"`
	`duplicates="NO"`
	
	Note: This is for ISERIESC.
		No special considerations.

`/commit`
`/nocommit`
 	
These options are only used for the OS/400 target environment.
	`commit="YES"`
	`commit="NO"`
	
	Note: This is for ISERIESC.
		No special considerations.

`/noff` There is no /FF option in VisualAge Generator. This option is only supported in VAGen resource association files.	`FormFeedOnClose="NO"` `FormFeedOnClose="YES"`	The migration tool converts /noff to *FormFeedOnClose=″NO″*.
`/text` There is no /NOTEXT option in VisualAge Generator. This option is only supported in VAGen resource association files.	`text="YES"` `text="NO"`	The migration tool converts /text to *text=″YES″*.

Table 127. Resource association (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
`/basename=xxxx` In VisualAge Generator, this option is only used for the OS/2® target environment. This option is only supported in VAGen resource association files.	Not supported.	The migration tool comments out any entry for the OS/2 target environment.
`/contable=xxxx` *xxxx* is one of the following: • a conversionTableName • EZECONVT This option is only supported in VAGen resource association files.	`conversionTable="xxxx"` *xxxx* is one of the following: • a conversionTableName • PROGRAMCONTROLLED	The migration tool uses the same *conversionTableName* when creating the EGL resource association entry.
`/keys=xxxx` In VisualAge Generator, this option is only used with /filetype=BTRIEVE. This option is only supported in VAGen resource association files.	`KEYS="xxxx"`	Because BTRIEVE is used in supported target environments, the migration tool migrates the /keys option to an EGL *keys* option.

`/blksize`
`/sysnum`
 	
These options are only used for VSE target environments.
	Not supported.	The migration tool comments out the resource association entry for target environments that are not supported.

/pcbno=n This is only valid for IMSVS or IMSBMP target environments or for MVS Batch if the file type is GSAM.	Not supported.	The migration tool comments out the resource association entry for target environments that are not supported or for MVS Batch if the file type is GSAM.

Link edit part

Table 128. Link edit part

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
In VisualAge Generator, the link edit part is typically named *programName.suffix*, where the first part of the name is the same as the VAGen program name and the suffix is LKG. The VAGen /linkedit generation option specifies the value for the suffix.	By default, in EGL the link edit part name must be the same name as the program name. If this is the case you do not need to specify the link edit build descriptor option. If you have multiple link edit parts for a program, then you must use different part names for the program's link edit build descriptor option parts. In this case, you must specify the complete link edit part name in the linkedit build descriptor option.	If the suffix is .LKG, the migration tool removes the suffix when creating the new EGL link edit part. If the suffix is anything other than .LKG, the migration changes *.suffix* to *_suffix* because periods (.) are not valid characters in EGL part names.

Table 128. Link edit part (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
A VAGen link edit part contains the link edit statements needed for linkediting a program during the preparation process in a host environment.	In EGL, a link edit part contains the link edit statements needed for linkediting a program during the build process in a host environment	The migration tool does the following: • Converts any symbolic parameters that are used within the link edit part to the corresponding EGL replacement symbolic parameter. • Uses the same indentation as in the VAGen part.

Bind control part

Table 129. Bind control part

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
In VisualAge Generator, the bind control part is typically named *programName.suffix*, where the first part of the name is the same as the VAGen program name and the suffix is BND. The VAGen /bind generation option specifies the value for the suffix.	By default, in EGL the bind control part name must be the same name as the program name. If this is the case you do not need to specify the bind control build descriptor option. If you have multiple bind control parts for a program, then you must use different part names for the program's bind control parts. In this case, you must specify the complete bind control part name in the bind build descriptor option.	If the suffix is .BND, the migration tool removes the suffix when creating the new EGL bind control part. If the suffix is anything other than .BND, the migration changes *.suffix* to *_suffix* because periods (.) are not valid characters in EGL part names.
A VAGen bind control part contains the DB2 bind commands needed for binding the DB2 DataBase Resource Module (DBRM) for a program during the preparation process in an MVS host environment.	In EGL, a bind control part contains the bind commands needed for binding the DBRM for a program during the build process in a ZOS host environment.	The migration tool does the following: • Adds additional commands at the beginning of the bind control part. These commands are needed by the build server. • Converts any symbolic parameters that are used within the bind control part to the corresponding EGL replacement symbolic parameter. • Uses the same indentation as in the VAGen part.

Symbolic parameters

The following tables show the relation between VAGen symbolic parameters and EGL symbolic parameters.

Table 130. Part-related symbolic parameters

Part-related symbolic parameters	Corresponding EGL symbolic parameter
EZECOBOLTYPE	Not supported
EZEDATA	DATA
EZEDBCS	Not supported
EZEDESTLIB	Not supported
EZEDESTNAME	Not supported

Table 130. Part-related symbolic parameters (continued)

Part-related symbolic parameters	Corresponding EGL symbolic parameter
EZEDLI	Not applicable — DL/I only
EZEENTRY	Not supported
EZEENV	SYSTEM
EZEGDATE	EZEGDATE
EZEGENOUT	Not supported
EZEGMBR	EZEGMBR
EZEGTIME	EZEGTIME
EZEMBR	In a linkedit or bind part, EZEALIAS. Otherwise, EZEMBR
EZEMBRPATH	Not supported
EZEMSG	Not supported
EZEPID	EZEPID
EZEPREPDESTACCOUNT	Not supported
EZEPREPDESTHOST	Not supported
EZEPREPDESTDIR	Not supported
EZEPREPDESTPASSWORD	Not supported
EZEPREPDESTUID	Not supported
EZEPREPFTPCMDSBCS	Not supported
EZEPREPFTPCMDBCS	Not supported
EZEPREPSENDCMDDBCS	Not supported
EZEPREPSESSION	Not supported
EZEPREPSP	Not supported
EZEPREPSQLDB	Not supported
EZEPREPWORKDB	Not supported
EZEPSB	Not applicable — DL/I and IMS only
EZEPTYPE	Not supported
EZESQL	EZESQL
EZETBLNAME	Not supported
EZETPROC	Not supported
EZETRAN	Not supported
EZETRANSFERTYPE	Not supported
EZETWASIZE	Not supported
EZEUSERID	Not supported
EZEVMLOADLIB	Not applicable — VM only
EZEVSELIB	Not applicable — VSE only
EZEXAPP	Not supported.

Table 131. File-related symbolic parameters

File-related symbolic parameters	Corresponding EGL symbolic parameter
EZEBLK	EZEBLK
EZEDBD	Not applicable — DL/I only

Table 131. File-related symbolic parameters (continued)

File-related symbolic parameters	Corresponding EGL symbolic parameter
EZEDD	EZEDD
EZEDLBL	Not applicable — VSE only
EZEDSN	EZEDSN
EZELRECL	EZELRECL
EZERECFM	EZERECFM

Table 132. User-defined symbolic parameters

User-defined symbolic parameters	Corresponding EGL symbolic parameter
COB2LIB	COBCICS
COBLIST	Not supported
DBDLIB	Not applicable — DL/I only
DSNLOAD	DSNLOAD
DSYS	DSYS
ELA	ELA
EZALTXTR	Special migration to normal build option—see Generation Options section, transferErrorTransaction="xxx"
EZONEAS2	Special migration to normal build option -- See Generation Options section, oneFormItemCopybook="YES"
EZUAUTH	EZUAUTH
EZUINST	EZUINST
PSBLIB	Not applicable — DL/I only
PROCLIB	Not applicable — VSE only
PWRCLASS	Not applicable — VSE only
RESLIB	Not applicable — DL/I and IMS only
SQLDBNAM	Not applicable — VM and VSE only
SQLPKGNM	Not applicable — VM and VSE only
SQLPROPT	Not applicable — VM and VSE only
SQLSTMDE	Not applicable — VM and VSE only
SQLSTOPT	Not applicable — VM and VSE only
SQLUSRPW	Not applicable — VM and VSE only
VMFMODE	Not applicable — VM only
VMDISKADDR	Not applicable — VM only
VUSERLIB	Not applicable — VSE only

Appendix C. Messages from the migration tools

This section contains the messages that are issued by the migration tools. You can find the messages based on their prefix in the following sections:

- HPT.EGL.00xxx - Stage 1 Common Messages
- HPT.EGL.01xxx - Stage 1 on VisualAge for Java
- HPT.EGL.02xxx - Stage 1 on VisualAge Smalltalk
- IWN.MIG - Stages 2 and 3 on the Rational Developer product

The character in the last position of each message number is a suffix that indicates the severity of the message:

- *i* — Informational message to indicate status or that the migration tool eliminated information during migration due to the differences between the VisualAge Generator and EGL languages. No user action is required.
- *w* — Warning message to indicate a possible problem. For example, the migration tool made a best guess for the EGL syntax. User action is only required if validation or generation detects an error.
- *e* — Error message. The migration tool was unable to make a reasonable guess for the EGL syntax. User action is required to provide missing or incomplete information.
- *t* - Trace message to indicate more detailed status than is provided by the informational messages. The trace message include details about when commit points are taken. The trace messages are self-explanatory and are not included in this migration guide.

Messages from the VisualAge Generator to EGL migration tool—Stage 1

The Stage 1 migration tools are shipped as samples. The messages are not translated within the sample tool itself. However, the messages **as shipped with the samples** are translated here in the Migration Guide.

Stage 1 common messages

The following messages are common to the VAGen migration tool on both VisualAge for Java and VisualAge Smalltalk

HPT.CM.215.e File *filename* cannot be opened. The return code is *returnCode* (*returnCodeText*).

Explanation: The specified file cannot be opened. The *returnCode* and *returnCodeText* indicate the reason why. *returnCode* 2 indicates the file cannot be found.

User Response: Provide a valid file for the migration tool.

HPT.EGL.0001.w Table name *tableName* is a reserved word. It must be renamed.

Explanation: The migration tool does not rename tables for you.

User Response: You must change the name of the table and all references to it, including references in any program's table and additional records list, logic statements, data item edit routines, and map edit routines. Be sure to change any non-VAGen references to the table name, including CICS program definitions. Alternatively, you can wait until you have migrated, rename the table in the Rational Developer product, and use the EGL *alias* property to specify the original table name.

HPT.EGL.0002.w Map group name *mapGroupName* is a reserved word. It must be renamed.

Explanation: The migration tool does not rename map groups for you.

User Response: You must change the name of the map group, the names of all maps in the map group, and all references to the map group, including references as any program's map group or help map group. Be sure to change any non-VAGen references to the map group name, including CICS program definitions. Alternatively, you can wait until you have migrated, rename the form group in the Rational Developer product, and use the EGL *alias* property to specify the original map group name.

HPT.EGL.0003.w Program name *programName* is a reserved word. It must be renamed.

Explanation: The migration tool does not rename programs for you.

User Response: You must change the name of the program and all references to it, including references on CALL, DXFR, and XFER statements. Also change the names of any bind control or linkedit parts that correspond to this program. Be sure to change any non-VAGen references to the program name, including CICS program definitions. Alternatively, you can wait until you have migrated, rename the program in the Rational Developer product, and use the EGL *alias* property to specify the original program name.

HPT.EGL.0004.w Control part name *partName* is a reserved word. It must be renamed.

Explanation: The specified control part name uses dot notation, where the name before the dot is a reserved word. The migration tool assumes that the name before the dot is a program name and that this control part is closely tied to a program. Because the migration tool does not rename programs, it also does not rename control parts that are in dot notation.

User Response: You must change the name of the program and all references to it, including references on CALL, DXFR, and XFER statements. Also change the names of any bind control or linkedit parts that correspond to this program. Be sure to change any non-VAGen references to the program name, including CICS program definitions. Alternatively, you can wait until you have migrated, rename the program in the Rational Developer product, and use the EGL *alias* property to specify the original program name. Refer to Appendix Appendix A, "Reserved words," on page 173 that lists the EGL reserved words.

HPT.EGL.0005.w UI Record *partName* is a reserved word or starts with the # symbol. It must be renamed.

Explanation: The specified UI record conflicts with the EGL naming conventions. This release of EGL does not support web transactions or UI records. The Stage 1 migration tool includes the UI record in the migration set. The Stage 2 and 3 migration tools filter out the UI records, without migrating them to EGL source.

User Response: None. There is no need to rename the UI record in VisualAge Generator. The expectation is that the Stage 2 and 3 migration tools will handle renaming for UI records just as they do for other records.

HPT.EGL.0006.i Migration of *preferenceFile* will produce *outputList*.

Explanation: Migration of *preferenceFile* will produce *outputList*. Possible outputs are migration plans, report, and database updates.

User Response: None.

HPT.EGL.0007.w No migration files were created based on the current filters.

Explanation: No migration files were created based on the current filters.

User Response: Change the filter preferences.

HPT.EGL.0008.e *PreferenceValue* is an invalid value for preference option *preferenceOption*.

Explanation: The value is invalid for the preference option.

User Response: Changes the preference option value in the preferences file.

HPT.EGL.0009.e Migration set *migrationSetName* requires the preferences for the spanning maps suffixes be specified.

Explanation: The specified migration set contains one or more map groups that span multiple projects or multiple packages. The migration tool requires you to specify the spanning maps suffix preferences so that it can create the project or package necessary for the map group.

User Response: Edit the Stage 1 migration preferences file. On the Mapping page, in the Spanning Maps section, specify values for the Project suffix and Package suffix fields. See "Mapping page" on page 93 for Java or "Mapping page" on page 111 for Smalltalk for more details.

HPT.EGL.0010.w No migration action was requested.

Explanation: You have not selected any output options for the Stage 1 migration tool.

User Response: Select one or more options. The options enable you to create a migration plan file, create a report, or update the database.

HPT.EGL.0011.i Starting the database clean up of migration set *migrationSetName***.**

Explanation: The migration database already contained information for the specified migration set. The migration tool deleted the migration set information in preparation for running Stage 1 with a new set of preferences.

User Response: None.

HPT.EGL.0012.i Completed the database clean up of migration set *migrationSetName***.**

Explanation: The migration database already contained information for the specified migration set. The migration tool deleted the migration set information in preparation for running Stage 1 with a new set of preferences.

User Response: None.

HPT.EGL.0013.e Each renaming rule must have a unique order value.

Explanation: Two or more renaming rules have the same order number.

User Response: Edit the Stage 1 migration preferences file and change the renaming rules so that each rule has a unique order number.

HPT.EGL.0014.i Migration set *migrationSetName-migrationSetVersion* **produced** *n* **error messages,** *n* **warning messages, and** *n* **informational messages.**

Explanation: *n* is the number of messages issued by the Stage 1 migration tool for the specified migration set. The count for the informational messages includes message HPT.EGL.0014.i.

User Response: None.

HPT.EGL.0015.e Derived EGL project name *eglProjectName* **contains invalid character(s):** *characterList***.**

Explanation: Using the renaming rules that you specified, the Stage 1 migration tool has created a proposed EGL project name that does not meet the EGL project naming conventions. The characters that are invalid shown in the *characterList*.

User Response: Edit the Stage 1 migration preferences file and modify the project renaming rules so that they result in valid EGL project names. When you modify the renaming rules, be sure to consider the effect of any renaming rules that specify a Mapping Context of *both*.

HPT.EGL.0016.e Derived EGL package name *eglPackageName* **contains invalid character(s):** *characterList***.**

Explanation: Using the renaming rules that you specified, the migration tool has created a proposed EGL package name that does not meet the EGL package naming conventions. The characters that are invalid shown in the *characterList*.

User Response: Edit the Stage 1 migration preferences file and modify your package renaming rules so that they result in valid EGL package names. When you modify the renaming rules, be sure to consider the effect of any renaming rules that specify a Mapping Context of *both*.

HPT.EGL.0017.e Derived EGL project name *eglProjectName* **cannot end with a period (.).**

Explanation: Using the renaming rules that you specified, the migration tool has created a proposed EGL project name that ends in a period. This name does not meet the EGL project naming conventions.

User Response: Edit the Stage 1 migration preferences file and modify your project renaming rules so that they result in valid EGL project names. When you modify the renaming rules, be sure to consider the effect of any renaming rules that specify a Mapping Context of *both*. Also consider the effect of any renaming rules that specify a String Context of *any*, *back* or *token*.

HPT.EGL.0018.e Derived EGL package name *eglPackageName* **cannot begin with a digit or end with a period (.).**

Explanation: Using the renaming rules that you specified, the migration tool has created a proposed EGL package name that ends in a period or begins with a digit. This name does not meet the EGL package naming conventions.

User Response: Edit the Stage 1 migration preferences file and modify your package renaming rules so that they result in valid EGL package names. When you modify the renaming rules, be sure to consider the effect of any renaming rules that specify a Mapping Context of *both*.

HPT.EGL.0019.i The Migration Feature *featureName versionName* **is loaded.**

Explanation: This informational message provides the migration feature name and version that is currently loaded into your Java workspace or Smalltalk image.

User Response: None.

HPT.EGL.0020.i The Migration Feature *featureName versionName* **is not loaded.** *listOfNames* **are not loaded.**

Explanation: This informational message provides the migration feature name and version that should be added to your Java workspace or loaded into your Smalltalk image. However, one or more Java packages or Smalltalk applications are not at the version expected for the migration feature. The *listOfNames* provides the list of Java packages or Smalltalk applications that are currently loaded but which are not at the expected version.

User Response: If you have not modified the Stage 1 migration tool, try adding the migration feature again for Java or loading the migration feature again for Smalltalk. If you have modified the Stage 1 migration tool, then this message serves as a reminder of the Java packages or Smalltalk applications that you have modified.

Stage 1 on VisualAge for Java

The following messages occur only in the VisualAge for Java version of the VisualAge Generator to EGL migration tool.

HPT.EGL.0101.e Current package name *vagenPackageName* **- results in EGL package name** *eglPackageName*, **which starts with # or uses reserved word(s)** *reservedWordList*. **It must be renamed.**

Explanation: EGL reserved words cannot be used as any word in the dot notation for EGL package names.

User Response: Use the Stage 1 renaming rules to create an EGL package name that does not violate the EGL naming restrictions. Refer to Appendix Appendix A, "Reserved words," on page 173 that lists the EGL reserved words. Be sure that the resulting EGL package name does not start with the # symbol.

HPT.EGL.0102.e Migration Set *migrationSetName* **-** *migrationSetVersion* **references version** *projectVersion1* **and** *projectVersion2* **of project** *projectName*. **The migration set was not created.**

Explanation: The migration tool expanded the high-level PLP project for the specified migration set version. The expanded high-level PLP project contains multiple versions of the same project name. Migration cannot continue.

User Response: If you are using PLP projects, modify the high-level PLP project and any lower-level PLP projects that it references so that only one version of each project is included in the PLP chain. If you created the migration plan file by hand, modify the migration plan file so that only one version of each project is specified for the migration set.

HPT.EGL.0103.e An error occurred while loading the database driver. driver: *driverName*. **Please check to ensure that the db2java.zip file is in the classpath.**

Explanation: The database driver that is specified in the Stage 1 preferences file could not be found.

User Response: Modify the Stage 1 preferences file to point to the correct driver name and location.

HPT.EGL.0104.e An error occurred while connecting to the database. database: *databaseName*.

Explanation: The Stage 1 migration tool was not able to connect to the migration database.

User Response: Make sure that the specified database has been created. Also review your user ID and password settings in the Stage 1 preferences file to ensure that they are correct.

HPT.EGL.0105.e Error occurred when closing the database connection.

Explanation: The Stage 1 migration tool was not able to close the connection to the migration database.

User Response: The Stage 1 migration tool does any commits before it tries to close the database connection. You should be able to shut down VisualAge Generator to force the connection to close.

HPT.EGL.0106.e Error accessing repository in method *methodName*.

Explanation: The specified method for the Stage 1 migration tool was not able to access the repository.

User Response: Verify that your repository is accessible and that there are no network problems if you are using a remote repository. Then try migrating again.

HPT.EGL.0107.e Error occurred while writing out XML file *fileName***.**

Explanation: The Stage1 migration tool was not able to write the specified file name.

User Response: Verify that there is sufficient space available for the file. Then try migrating again.

HPT.EGL.0108.w *partType* **part was excluded from migration due to invalid whitespace in the name** *partName***. The part is in package** *packageName***,** *versionName***.**

Explanation: In VisualAge Generator, it is possible to create a part that contains blanks at the end of the part name. Frequently when this occurs, there are 2 parts with the same name, except that one part has blanks at the end of its name. These are not duplicate parts because the names differ slightly. However, the part name in the External Source Format file is identical, even though the rest of the source code for the parts might differ. The Stage 1 migration tool omits any part name that contains blanks from the migration set. This is because the part name that ends with blanks cannot be referenced by any other VAGen part. In the case where there is no similarly named part, this technique ensures that a part that cannot be referenced by other parts is not migrated. In the case where there is a similarly named part, this technique ensures that the Stage 2 migration tool converts the correct part definition to EGL.

User Response: None. However, you might want to review the part in VisualAge Generator to determine if there were similarly named parts. Use the VAGen Parts Browser, and search all parts in the migration set for *partName**.

HPT.EGL.0109.e An unexpected exception occurred: *javaExceptionStackTrace*

Explanation: An unexpected error occurred during the Stage 2 or Stage 3 migration tool.

User Response: Review the *javaExceptionStackTrace*. Depending on the error it might be something you can ignore or correct. For example:

- You can ignore a message that indicates a character that could not be converted was replaced by a substitute character. This message occurs if an invalid character occurs in the External Source Format. The character is replaced by a blank in the migration database. The Stage 1 migration tool continues processing. You can use your migration database for Stage 2 and 3.
- You can correct the problem if the message indicates that an SQL column is too short to contain the EGL file name. In this case, the Stage 1 migration tool stops processing because the information in the migration database is invalid. You can correct the problem by modifying the SQL table definition to increase the length of the column and then running Stage 1 again.

If you are not able to resolve the problem, contact IBM support for assistance.

Stage 1 on VisualAge Smalltalk

The following messages occur only in the VisualAge Smalltalk version of the VisualAge Generator to EGL migration tool.

HPT.EGL.0201.e Current application name *vagenApplicationName* **- results in EGL package name** *eglPackageName***, which starts with # or uses reserved word(s)** *reservedWordList***. It must be renamed.**

Explanation: EGL reserved words cannot be used as any word in the dot notation for EGL package names.

User Response: Use the Stage 1 renaming rules to create an EGL package name that does not violate the EGL naming restrictions. Refer to Appendix Appendix A, “Reserved words,” on page 173 that lists the EGL reserved words. Be sure that the resulting EGL package name does not start with the # symbol.

HPT.EGL.0202.e Migration set *migrationSetName* **references Configuration map** *configurationMapName***, which is not defined in the repository.**

Explanation: The Stage 1 migration tool expanded the high-level configuration map for the specified migration set. However, when the tool expanded the high-level configuration map and the chain of required maps and applications, one or more required maps was not available in the library.

User Response: Determine whether the required map should be included in the migration set. If so, check to see whether the requested version of the configuration map has been purged from the library. If so, salvage the requested configuration map and then migrate again.

HPT.EGL.0203.e *ProgramContext* **encountered a database error** *errorMessage***.**

Explanation: A database error occurred. Possible problems are invalid schema name, user authority restrictions, or missing tables.

User Response: Correct the migration preferences file. If the problem persists, contact IBM support for assistance.

HPT.EGL.0204.e An error occurred while connecting to the database. *ErrorMessage***.**

Explanation: A database error occurred on connect. Possible problems are invalid userid or password name or invalid database name.

User Response: Correct the migration preferences file. If the problem persists, contact IBM support for assistance.

HPT.EGL.0205.i Migration produced *n* **migration sets from the** *v* **versions of configuration map** *configMapName***. The preference file specified the version depth as** *d***.**

Explanation: The Stage 1 migration preferences file specified that you wanted to migrate the number of versions specified by *d*. The Stage 1 migration tool should have produced *d* migration sets -- one for each version of the specified configuration map. However, the migration tool only created the number of migration sets specified by *n*. The number specified by *v* is the number of versions of the specified configuration map that the migration tool found in the library. If *v* is less than *d*, this means that there were not as many versions of the configuration map as you anticipated. In this case, *n* and *v* should be equal, indicating that all the configuration map versions resulted in migration sets. If *v* is greater than *d*, this means that there are more versions of the configuration map in the library. In this case, *n* and *d* should be equal, indicating that your version depth preference was met.

User Response: None.

HPT.EGL.0206.e Migration set *migrationSetName* **encountered a load error.**

Explanation: The migration set could not be loaded into your image. This could occur because there are duplicate part names in the migration set.

User Response: Review the **System Transcript** to determine the cause of the error. Correct the problem and then run Stage 1 migration again.

HPT.EGL.0207.w *partType* **part was excluded from migration due to invalid whitespace in the name** *partName***. The part is in application** *applicationName***,** *versionName***.**

Explanation: In VisualAge Generator, it is possible to create a part that contains blanks at the end of the part name. Frequently when this occurs, there are 2 parts with the same name, except that one part has blanks at the end of its name. These are not duplicate parts because the names differ slightly. However, the part name in the External Source Format file is identical, even though the rest of the source code for the parts might differ. The Stage 1 migration tool omits any part name that contains blanks from the migration set. This is because the part name that ends with blanks cannot be referenced by any other VAGen part. In the case where there is no similarly named part, this technique ensures that a part that cannot be referenced by other parts is not migrated. In the case where there is a similarly named part, this technique ensures that the Stage 2 migration tool converts the correct part definition to EGL.

User Response: None. However, you might want to review the part in VisualAge Generator to determine if there were similarly named parts. Use the VAGen Parts Browser, and search all parts in the migration set for *partName**.

HPT.EGL.0208.e Database column *schemaName.tableName.columnName* **has truncated data.**

Explanation: One or more of you renaming rules resulted in an EGL project, package, or version name that is longer than fits in the corresponding SQL columns.

User Response: Modify your renaming rules so that they result in shorter EGL project, package, or version names. Alternatively, you can modify the DB2 table to increase the length of the SQL column. However, before you increase the length of the SQL column, consider whether you will want to scroll these long names after migration and whether a longer name might exceed the EGL limits. After you have modified your renaming rules or the SQL column, run Stage 1 migration again.

Messages from the VisualAge Generator to EGL migration tool— Stage 2

The message inserts are always the VAGen part name, before any required renaming for EGL reserved words.

IWN.MIG.0001.e Exception parsing External Source Format file *fileName* **- invalid External Source Format header.**

Explanation: The migration tool only processes External Source Format that is exported from VisualAge Generator 4.5. The first line of the specified External Source Format file does not have the proper header for a VisualAge Generator 4.5 External Source Format file.

User Response: Import the External Source Format file into VisualAge Generator 4.5. This converts your current parts to VisualAge Generator 4.5 format. Then export the parts using VisualAge Generator 4.5 and run migration again.

IWN.MIG.0002.e Exception parsing External Source Format file *fileName*, *partType*, *partName* **-** *exceptionText*

Explanation: A problem occurred parsing the External Source Format syntax from VisualAge Generator. Possible causes of this problem are:

- Mismatched quote marks, including the following: currency field for a data item
- Mismatched comment delimiters in a control part.
- National language characters that are not valid for your locale. For example, attempting to migrate VAGen source code that uses double-byte characters such as Chinese on a workstation that is not set for a double-byte locale.

User Response: Correct the part in VisualAge Generator and export the External Source Format again. Then run the Stage 2 migration tool to process the file. If you are unable to correct the part in VisualAge Generator, contact IBM support for assistance. Be prepared to provide the External Source Format source for the file.

IWN.MIG.0003.e Exception converting to EGL for file *fileName*, *partType*, *partName* **-** *exceptionText*

Explanation: A problem occurred during the creation of the EGL source. The *exceptionText* identifies the specific problem that occurred.

User Response: Contact IBM support for assistance. Be prepared to provide the External Source Format source for the file.

IWN.MIG.0047.i Migration set *Name* **— migration started.**

Explanation: This is an informational message to indicate status from the migration tool.

User Response: None.

IWN.MIG.0048.i Migration set *Name* **- migration completed.**

Explanation: This is an informational message to indicate status from the migration tool.

User Response: None.

IWN.MIG.0049.i *partType partName* **for EGL** *projectName*, *packageName*, *fileName* **- Migration started**

Explanation: This is an informational message to indicate status from the migration tool. The *partType* is one of the following: Program, Map Group, or Table. The associates for the specified *partName* will be migrated at the same time. The associates might be in the same file as the *partName* or in different projects, packages, or files based on information in the migration database. When migration of a program starts, each associated map group is migrated, followed by each associated table. Finally, any remaining associates (records, shared items, and functions) are migrated.

User Response: None.

IWN.MIG.0050.i Program *programName* **- Migration of other associates started**

Explanation: This is an informational message to indicate status from the migration tool. When migration of a program starts, each associated map group is migrated, followed by each associated table. Finally, any remaining associates (records, shared items, and functions) are migrated. Message IWN.MIG.0050.i is issued when the migration of the remaining associates for the program starts.

User Response: None.

IWN.MIG.0051.e Exception parsing migration set *planName*, *partType*, *partName* **- invalid External Source Format header.**

Explanation: The migration tool only processes External Source Format that is exported from VisualAge Generator 4.5. The first line of the External Source Format for the specified part does not have the proper

header for a VisualAge Generator 4.5 External Source Format file. This might occur if you modified the sample Stage 1 migration tool or if you wrote your own Stage 1 migration tool to load the migration database.

User Response: Import the External Source Format file into VisualAge Generator 4.5. This converts your current parts to VisualAge Generator 4.5 format. Then use the Stage 1 migration tool to export the migration set.

IWN.MIG.0052.e Exception parsing migration set *planName*, *partType*, *partName* **-** *exceptionText*.

Explanation: A problem occurred parsing the External Source Format syntax from VisualAge Generator. Possible causes of this problem are:

- Mismatched quote marks, including the following:
  - currency field for a data item
- Mismatched comment delimiters in a control part.
- National language characters that are not valid for your locale. For example, attempting to migrate VAGen source code that uses double-byte characters such as Chinese on a workstation that is not set for a double-byte locale.

User Response: Correct the part in VisualAge Generator and run Stage 1 migration again to correct the database. Then run the Stage 2 migration tool again to process the updated parts. If you are unable to correct the part in VisualAge Generator, contact IBM support for assistance. Be prepared to provide a small repository (.dat file) containing the parts that have problems.

IWN.MIG.0053.e Exception converting to EGL for migration set *planName*, *partType*, *partName* **-** *exceptionText*.

Explanation: A problem occurred during the creation of the EGL source. The *exceptionText* identifies the specific problem that occurred.

User Response: Contact IBM support for assistance. Be prepared to provide the External Source Format source for the part.

IWN.MIG.0054.e Invalid External Source Format for migration set *migrationSetName*, *partType*, *partName*.

Explanation: The External Source Format stored for the specified part is not valid. The migration tool continues processing other parts in the specified migration set. For the purposes of migrating with associated parts, the migration tool considers the specified part to be unavailable. The migration tool stores intentionally invalid EGL in the migration database for the specified part. The EGL that is stored is EZE_UNKNOWN_PARTTYPE *partName*; This ensures that there will be an error in the Problems view.

User Response: Review the specified part in VisualAge Generator. Try exporting External Source Format for the part and migrating the part in single file mode.

IWN.MIG.0055.e Migration halted - error limit exceeded.

Explanation: The error threshold has been exceeded for parts with invalid External Source Format. The migration tool stops processing.

User Response: Review all occurrences of message IWN.MIG.0054.e. If you created your own tool to load the migration database, there might be a problem with the way the tool is loading External Source Format code into the migration database. See Appendix G, "Migration Database," on page 309 for some queries that might be useful in determining what is causing the problem.

IWN.MIG.0060.e An error occurred while loading the database driver. driver: *driverName*

Explanation: The specified database driver cannot be located.

User Response: Correct the database driver name. Also confirm that your database driver location is correct.

IWN.MIG.0061.e An error occurred while connecting to the database. database: *databaseName.errorText*

Explanation: The migration tool cannot connect to the specified database using the specified schema name. The *errorText* field provides additional details of why the connection failed.

User Response: Correct the database name. If you are using the COM.ibm.db2.jdbc.app.DB2Driver to connect to a remote database, be sure that you have cataloged the database locally.

IWN.MIG.0062.e Error occurred when closing the database connection.

Explanation: Migration completed successfully, but the migration tool was not able to close the database connection.

User Response: Shut down the Rational Developer product before attempting to backup your database.

IWN.MIG.0080.i VAGen Migration Preferences file pref_store.ini not found; defaults assumed.

Explanation: There is no VAGen migration preferences file. The migration tool uses the default values for the preferences (for example, the renaming suffix and help map suffix). This might be because you specified a new workspace during migration so that preferences do not exist. The preferences file is located in

```
workspace-directory\.metadata\plugins
   \com.ibm.etools.egl.vagenmigration
   \pref_store.ini
```

User Response: See "VAGen Migration Syntax Preferences" on page 128 for information about the default values for the migration preferences.

IWN.MIG.0081.i File *fileName* - migration completed.

Explanation: The migration tool has completed processing for the specified file.

User Response: Review the log messages to see the results of the migration.

IWN.MIG.0082.e File *fileName* - required parameters are not specified.

Explanation: One or more required parameters have not been specified. The *-importFile* parameter is always required. If the *-importFile* parameter specifies an External Source Format file, then the *-eglFile* and *-package* parameters are also required.

User Response: Review the batch command file to determine which parameters were not specified. Add the parameters and then run the batch command file again.

IWN.MIG.0083.e File *fileName* - parameter *parmName* has not been assigned a value.

Explanation: *parmName* is one of the following: -importFile, -eglFile, -package.

User Response: Correct the batch command file and then run it again.

IWN.MIG.0084.e File *fileName* - parameter *parmName*, value *value* is not valid.

Explanation: *parmName* is one of the following: -importFile, -eglFile, -package. The parameter names are case sensitive.

User Response: Correct the batch command file and then run it again.

IWN.MIG.0085.e File *fileName* - invalid parameters are passed in the parameter list.

Explanation: There is a problem with the batch command file. One or more of the parameters is entered incorrectly. The only valid parameters are: -importFile, -eglFile, -package, and -overwrite.

User Response: Correct the batch command file and then run it again.

IWN.MIG.0090.e Program *programName* - is a web transaction program, which is not supported for migration.

Explanation: The specified program is a web transaction program. The migration tool does not migrate the program part.

User Response: None. You must wait until a future release of EGL to be able to migrate this program.

IWN.MIG.0091.e Record *recordName* - is a UI record, which is not supported for migration.

Explanation: The specified record is a User Interface (UI) record. The migration tool does not migrate the record part.

User Response: None. You must wait until a future release of EGL to be able to migrate this record.

IWN.MIG.0092.e Function *functionName* - uses XFER with a UI record, which is not supported for migration.

Explanation: The specified function contains an XFER statement that uses a UI record. This release of EGL does not support web transactions or UI records. The migration tool migrates the function, but the EGL syntax that is produced might not be the correct replacement for the XFER with a UI record statement. However, this technique preserves as much of your logic as possible.

User Response: Review the EGL function. You will not be able to migrate any web transaction programs or UI records that use this function. Therefore, you will not be able to use this function until a future release of EGL. You might want to move the function to a different project to avoid having errors on the EGL Problems view.

IWN.MIG.0093.w Function *functionName* - uses one or more EZEUIxxx special function words.

Explanation: The specified function contains statements that use the special function words EZEUIERR or EZEUILOC. Web transactions and UI records are not supported by this release of EGL. However, there are EGL replacements for the EZEUIxxx special function words. The migration tool migrates the function and converts the EZEUIxxx word to the EGL

equivalent system library function. This preserves as much of your logic as possible.

User Response: Review the EGL function. You will not be able to migrate any web transaction programs or UI records that use this function. However, you might be able to use the function if you create any new EGL page handlers.

IWN.MIG.0094.e PSB *psbName* **is not supported for migration.**

Explanation: PSB parts are not currently supported by the migration tool. The migration tool does not migrate the PSB part.

User Response: Review the PSB and any programs that use this PSB. If the PSB only contains TP PCBs you might be able to generate the program for another environment if all of the following are true:

- The program does not perform any DL/I I/O.
- The program does not pass EZEDLPSB or EZEDLPCB as an argument on a call.
- The program does not expect to receive EZELDPSB or EZEDLPCB as a called parameter.
- The program does not use any EZEDLxxx status words (for example, EZEDLKEY or EZEDLSTC).
- The program does not use a resource association part to associate a file or print output with an IMS message queue, GSAM or MFS.
- The program does not expect to call or be called by a program that only runs in the IMS or IMS BMP environment.

IWN.MIG.0095.e Function *functionName* **- EZESCRPT is not supported for migration.**

Explanation: The specified function contains statements that use the EZESCRPT special function word. EZESCRPT is not currently supported by the VAGen migration tool. The migration tool migrates the function, but comments out the statement that uses EZESCRPT.

User Response: Review the EGL function. You will not be able to generate or run programs that use this function in this release.

IWN.MIG.0096.e Function *functionName* **- uses one or more EZEDLxxx status words, which are not supported for migration.**

Explanation: The specified function contains statements that use one or more of the EZEDLxxx special function words (for example, EZEDLKEY and EZEDLSTC). DL/I is not currently supported by the VAGen migration tool. The migration tool migrates the function, but the EGL syntax that is produced might not be the correct replacement for this EZEDLxxx special function word.

User Response: Review the EGL function. You will not be able to generate or run programs that use this function in this release. You might want to move the function to a different project to avoid having errors on the EGL Problems view.

IWN.MIG.0097.e Program *programName* **- called parameter** *parmName* **is not supported for migration.**

Explanation: A VAGen program specifies either EZEDLPSB or EZEDLPCB as a called parameter. DL/I and the IMS runtime environment are not currently supported by the migration tool. The migration tool migrates the program, but comments out the use of this called parameter.

User Response: Review the program. You will not be able to test or generate it at this time.

IWN.MIG.0098.e Record *recordName* **is a DL/I record, which is not supported for migration.**

Explanation: The specified record name is a DL/I record. DL/I is not currently supported by the migration tool. The migration tool does not migrate the record.

User Response: Review any programs that use this record. You will not be able to test or generate the programs at this time.

IWN.MIG.0099.e Function *functionName* **- DL/I I/O is not supported for migration.**

Explanation: The specified function performs I/O on a DL/I record. DL/I is not currently supported by the VAGen migration tool. The migration tool migrates the function, but the information for the I/O statement is not correct.

User Response: Review the EGL function. You will not be able to generate or run programs that use this function in this release. You might want to move the function to a different project to avoid having errors on the EGL Problems view.

IWN.MIG.0101.e Data item *dataItemName* **- Unable to determine edit routine type for** *editRoutineName***; function assumed.**

Explanation: VisualAge Generator supports EZEC10, EZEC11, a function or a table as the map edit routine for a data item. EGL supports both a *validator* function and a *validatorTable* property for a data item. The migration tool converts the map edit routine as follows:

- EZEC10 and EZEC11 migrate to the *validator* property.
- If the part specified by *editRoutineName* is available during migration and is a function, the *editRoutineName* migrates to the *validator* property. The migration tool also migrates the edit routine to

the *validator* property if the *editRoutineName* is longer than 7 characters because table names are limited to 7 characters in VisualAge Generator.

- If the part specified by *editRoutineName* is available and is a table, the *editRoutineName* migrates to the *validatorTable* property. The migration tool also migrates the edit routine to the *validatorTable* property if an edit message is specified for the item because VisualAge Generator only uses the edit message in conjunction with EZEC10, EZEC11, or a table.
- If the part specified by the *editRoutineName* is not available during migration and the *editRoutineName* is 7 or fewer characters and an edit message is not specified, the migration tool assumes that *editRoutineName* is a function and migrates to the *validator* property. Message *IWN.MIG*.0101.e is only issued in this situation.

User Response: If the specified edit routine is not a function, modify the data item definition and change the *validator* property to the *validatorTable* property. For additional considerations, see the information on edit routines in "Map item edit routine for shared data items" on page 45.

IWN.MIG.0102.w Part *partName* **uses shared data item** *dataItemName* **- Unable to migrate to a primitive definition; using a type definition**

Explanation: You selected the preference that migrates VAGen shared data items to EGL primitive definitions whenever a shared data item is used in a record, table, function parameter list, or function local storage. The item specified by *dataItemName* is used in the part specified by *partName*. However, the data item definition is not available during migration. The migration tool uses the data item name as a type definition so that the migrated code will be valid.

User Response: No action is required if you want to use the type definition. If you want to use a primitive definition, modify the specified part to use the correct item characteristics. Alternatively, include the shared data item in your migration set (or the External Source Format file if you are migrating in single file mode) and migrate again.

IWN.MIG.0201.i Record *recordName* **contains level 77 items; creating additional record named** *level77RecordName***.**

Explanation: VisualAge Generator supports level 77 items in working storage records. EGL does not support level 77 items. EGL does permit the definition of independent data items. The migration tool splits working storage records that contain level 77 items into 2 separate basicRecords -- one containing the non-level 77 items and one containing the level 77 items. If the working storage record contains only level 77 items, then the migration tool only creates the level 77 basicRecord. If a program specifies a primary working storage record that contains level 77 items, the migration tool includes declarations for both the original basicRecord and the level 77 basicRecord in the program definition.

User Response: None. For additional considerations, including the effect if *recordName* is not available during the migration of programs and statements, see the information on level 77 items in records in "Level 77 items in records" on page 48.

IWN.MIG.0202.i Record *recordName* **redefines** *redefinedRecordName***.**

Explanation: *recordName* is a VAGen Redefined record that specifies *redefinedRecordName* as the record being redefined. *recordName* provides a different item layout for the same physical storage that is used by the *redefinedRecordName*. EGL does not retain redefinition information in the records. That information is kept only in the programs. The migration tool includes a comment in *recordName* to provide the original VAGen *redefinedRecordName* information. When migrating programs, if *recordName* is available and results in an overlay definition in VisualAge Generator, the migration tool includes the *redefines* property for the *recordName* declaration.

User Response: None. For additional considerations, including the effect if *recordName* is not available during migration of a program, see the information on redefined records in "Redefined records" on page 47.

IWN.MIG.0203.e Record *recordName* **— Does not contain any items.**

Explanation: VisualAge Generator permits you to save a record part that does not contain any items. However, the record cannot be used in any programs because it is invalid. The migration tool migrates the record.

User Response: Determine whether you still need to have the record. If so, edit the record and add one or more data items. If not, delete the record.

IWN.MIG.0204.e Record *recordName* **- alternate specification record** *altspecRecord* **is not available; SQL table names cannot be determined.**

Explanation: The record *recordName* specifies an alternate specification record named *altspecRecord*, which provides the item structure for *recordName*. In VisualAge Generator, for SQL records, the alternate specification record also provides the SQL table names. In EGL, the alternate specification record only provides the structure by using the *embed* statement. The table names must be specified in each SQL record definition. When migrating *recordName*, the record specified as the alternate specification record is not available during

migration. The migration tool cannot determine the correct table names and sets the *tableNames* property to ###TABLES_NOT_FOUND###. The definition for *recordName* is invalid.

User Response: Edit *recordName* and copy in the *tableNames* and/or *tableNameVariables* properties from the alternate specification record. The *tableNames* property provides the actual SQL table names. The *tableNameVariables* property provides table name host variables. Both the tableNames and the tableNameVariables properties can be used if the recordName references a mixture of actual SQL table names and SQL table name host variables. For additional considerations, see the information on alternate specification records in "Alternate specification records" on page 49.

IWN.MIG.0205.e Record *recordName* **- alternate specification record** *altspecRecord* **is not available; SQL key items cannot be determined.**

Explanation: The record *recordName* specifies an alternate specification record named *altspecRecord*, which provides the item structure for *recordName*. When VisualAge Generator determines the default selection condition for an SQL record, VisualAge Generator merges any items that specify key=yes in the alternate specification record with the key item, if any, specified in *recordName*. The keys are merged based on the order in which the items are listed in the record structure. In EGL, the alternate specification record only provides the structure by using the *embed* statement. All key items must be specified in each record definition. When migrating *recordName*, the record specified as the alternate specification record is not available during migration. The migration tool cannot determine the correct key items and sets the *keyItems* property to ###KEYS_NOT_FOUND###, followed by the key item, if any, from *recordName*. The definition for *recordName* is invalid.

User Response: Edit *recordName* and change the *keyItems* property to replace ###KEYS_NOT_FOUND### with the list of item names that specified key=yes in the VAGen alternate specification record (*altspecRecord*). Be sure to merge the key items from the alternate specification record with the key item specified in the VAGen definition for *recordName* so that the *keyItems* property lists the items in the same order they appear in the record structure. If an item is specified as key=yes in the alternate specification record and as the key item in *recordName*, only include the item once in the merged list of *keyItems* in *recordName*. For additional considerations, see the section on "Alternate specification records" on "Alternate specification records" on page 49.

IWN.MIG.0206.i SQL Record *recordName* **— Contains a key item** *keyItem* **without specifying an alternate specification record.**

Explanation: VisualAge Generator permits you to save an SQL record that specifies a key item even if you do not specify an alternate specification record. However, in this situation, VisualAge Generator ignores the key item during test and generation. The key item only has meaning when there is also an alternate specification record.

User Response: None. The key item was ignored in VisualAge Generator. The migration tool eliminates it during migration.

IWN.MIG.0207.i Record *recordName* **- Specifies alternate specification record** *altspecRecord* **with only level 77 items; embed statement omitted.**

Explanation: The record specified by *altspecRecord* is a working storage record that only contains level 77 items. When *recordName* specifies a working storage record as the alternate specification, VisualAge Generator uses only the structure (the non-level 77 items) from *altspecRecord*. The migration tool omits the embed statement because there are no items in the structure of *altspecRecord*.

User Response: None. However, you might want to delete *recordName* because it is an empty record. Be sure to delete all references to *recordName* in your programs.

IWN.MIG.0208.e Record *recordName* **- Alternate specification record** *altspecRecord* **is not available; cannot determine SQL column name for !itemColumnName variables.**

Explanation: The record *recordName* specifies an alternate specification record named *altspecRecord*, which provides the item structure for *recordName*. When VisualAge Generator determines the default selection condition for an SQL record, VisualAge Generator converts any !itemColumnName variables to the corresponding SQL column name. In EGL, !itemColumnName variables are not supported. The SQL columns must be explicitly named in the default selection condition for each SQL record definition. When migrating *recordName*, the record specified as the alternate specification record is not available during migration. The migration tool cannot determine the correct SQL column name that corresponds to one or more !itemColumnName variables in the default selection condition. The migration tool uses !itemColumnName in the *EGL* default selection condition. The definition for *recordName* is invalid.

User Response: Edit *recordName* and change the *defaultSelectCondition* property to replace the !itemColumnName variables with the corresponding

SQL column names from the VAGen alternate specification record (*altspecRecord*). For additional considerations, see the information on !itemColumnName variables in "Alternate specification records" on page 49.

IWN.MIG.0209.e Record *recordName* **- Alternate specification record** *altspecRecord* **has no items; embed statement omitted.**

Explanation: The record *recordName* specifies an alternate specification record named *altspecRecord*, which provides the item structure for *recordName*. However, the alternate specification record does not have any data items. The migration tool omits the embed statement from the definition for *recordName*.

User Response: None. However, you should review *recordName* and *altspecRecord* to determine whether you need to include data items or whether the two records can be deleted. Be sure to delete all references to these records in your programs.

IWN.MIG.0210.e Record *recordName* **- Unable to determine column names for !itemColumnName variables.**

Explanation: The default select condition for the specified record uses one or more VAGen !itemColumnName variables. A VAGen !itemColumnName variable specifies the name of an item in the SQL record definition which corresponds to the actual SQL column name. VisualAge Generator determines the actual SQL column names for any !itemColumnName variables from the SQL record at test and generation time. EGL does not support !itemColumnName variables. Instead, EGL requires that the actual SQL column names be used in any modified SQL statement. Message IWN.MIG.0210.e is issued when the record specified by recordName is invalid in VisualAge Generator. In this case, the record uses one or more !itemColumnName variables that are not defined within the record or its alternate specification record. The migration tool is unable to substitute the actual SQL column name.

User Response: Edit the record and change the !itemColumnName variables to the correct SQL column names.

IWN.MIG.0301.e Table name *tableName* **is a reserved word. It must be renamed.**

Explanation: The migration tool does not rename tables for you.

User Response: You must change the name of the table and all references to it. This includes references in the following places:

- Program use declaration statements
- Logic statements in programs and functions
- Data item validatorTable properties
- Form field validatorTable properties

If you want to keep the original table name as the name for the generated table, set the *alias* property to the original table name. If you do not specify the *alias* property, be sure to change any non-EGL references to the table name, including CICS program definitions.

IWN.MIG.0401.e Map group (form group) name *mapGroupName* **is a reserved word. It must be renamed.**

Explanation: The migration tool does not rename map groups (form groups) for you.

User Response: You must change the name of the form group and all references to it, including references in program use declaration statements. If you want to keep the original map group name as the name for the generated form group, set the *alias* property to the original map group (form group) name. If you do not specify the *alias* property, be sure to change any non-EGL references to the form group name, including CICS program definitions.

IWN.MIG.0402.e Map group *mapGroupName* **-- Multiple devices have the same depth and width, but different floating areas; devices are:** *devicesList*

Explanation: VisualAge Generator permits, but does not recommend, different floating area sizes for device types that have the same device size. EGL only permits one floating area for each device size. The migration tool migrates the floating area size for each device type. The form group definition is invalid. This message is repeated for each group of same-size device types that specified different floating area information in VisualAge Generator.

User Response: Edit the form group and delete all except one floating area specification for each group of same-size devices.

IWN.MIG.0403.e Form Group *formGroupName* **- Requires editing to nest forms within the form group.**

Explanation: When you migrate in single file mode, the migration tool does not nest forms within the form group. Instead, the migration tool inserts an EGL *use* statement to indicate the name of the forms that belong to the form group. The migration tool includes comments at the beginning and end of each form to indicate its form group.

User Response: Edit the file containing the form group part and move the form parts so that they are nested within the form group part. The *use* statements in the form group part indicate where the forms should be moved. After you have nested the form within the

form group, remove the *use* declaration statement.

IWN.MIG.0404.w Map Group *mapGroupName* **- Uses device** *deviceName* **and size** *depth,width* **which is no longer supported. It must be changed.**

Explanation: VisualAge Generator permits some device types which EGL COBOL generation no longer supports for floating areas and text forms. The migration tool includes the original depth and width in the *screenSize* property within the *ScreenFloatingArea* property. However, this *screenSize* is no longer supported in EGL COBOL generation. If you generate for COBOL, there will be an error message from generation.

User Response: If you plan to generate for COBOL, edit the *formGroup* in EGL and either remove the *ScreenFloatingArea* property for this depth and width or change the depth and width to a size that is supported. You might also need to modify the text forms within the *formGroup* to reposition the variables and constants to better fit the new depth and width.

IWN.MIG.0501.e Help map group *mapGroupName* **contains map** *mapName* **with variable fields —** *mapName* **conflicts with the same map name in the program's main map group.**

Explanation: VisualAge Generator permits the same map name to be used in a program's main map group and its help map group. EGL does not permit any duplicate form names in the program's two form groups. This restriction applies even if the forms with duplicate names are not used by the program. The migration tool renames maps in a program's help map group if they conflict with maps in the program's main map group and only contain constant fields. The migration tool does not rename a map in the program's help map group if it contains variable fields, even if the name conflicts with a map name in the program's main map group.

User Response: Edit the help form group and change the name of the form. Also be sure to change the form definition and all references to this form in all programs that use the form group. For additional considerations, see the information on map names in "Map names and help map names" on page 54.

IWN.MIG.0502.e Map group *mapGroupName*, **map** *mapName* **and variable field** *mapItemName* **- Unable to determine edit routine type for** *editRoutineName*; **function assumed.**

Explanation: VisualAge Generator supports EZEC10, EZEC11, a function or a table as the map edit routine for a map variable. EGL supports both a *validator* function and a *validatorTable* property for a form field. The migration tool converts the map edit routine as follows:

- EZEC10 and EZEC11 migrate to the *validator* property.
- If the part specified by *editRoutineName* is available during migration and is a function, the *editRoutineName* migrates to the *validator* property. The migration tool also migrates the edit routine to the *validator* property if the *editRoutineName* is longer than 7 characters because table names are limited to 7 characters in VisualAge Generator.
- If the part specified by *editRoutineName* is available and is a table, the *editRoutineName* migrates to the *validatorTable* property. The migration tool also migrates the edit routine to the *validatorTable* property if an edit message is specified for the item because VisualAge Generator only uses the edit message in conjunction with EZEC10, EZEC11, or a table.
- If the part specified by the *editRoutineName* is not available during migration and the *editRoutineName* is 7 or fewer characters and an edit message is not specified, the migration tool assumes that *editRoutineName* is a function and migrates to the *validator* property. Message *msgPrefix*.0502.e is only issued in this situation.

User Response: If the specified edit routine is not a function, modify the form field and change the *validator* property to the *validatorTable* property. For additional considerations, see the information on edit routines in "Variable map fields and edit routines" on page 57.

IWN.MIG.0503.w Map group *mapGroupName*, **map** *mapName* **- Unnamed variable field converted to constant field at position(***row,column***).**

Explanation: VisualAge Generator permits, but does not recommend, unnamed variable fields on maps. The program cannot access these unnamed variable fields. At test and generation, unnamed variable fields are converted to constants. The migration tool converted this unnamed variable field to a constant because one or more properties are non-default values.

User Response: Review the form definition and ensure that a constant field is the correct migration for this field. For additional considerations, see the information on unnamed variable fields in "Unnamed variable fields" on page 60.

IWN.MIG.0504.w Map group *mapGroupName*, **map** *mapName* **- Unnamed variable field removed from position(***row,column***).**

Explanation: VisualAge Generator permits, but does not recommend, unnamed variable fields on maps. The program cannot access these unnamed variable fields. At test and generation, unnamed variable fields are converted to constants. The migration tool removed

this unnamed variable field because all of its properties specify the default values for a constant field. EGL does not require that constants with default properties be explicitly defined for the form.

User Response: Review the form definition and ensure removing this field is the correct migration. For additional considerations, see the information on unnamed variable fields in "Unnamed variable fields" on page 60.

IWN.MIG.0505.w Map Group *mapGroupName*, **map** *mapName* **- Uses device** *deviceName* **and size** *depth,width* **which is no longer supported. It must be changed.**

Explanation: VisualAge Generator permits some device types which EGL COBOL generation no longer supports for text forms. The migration tool includes the original depth and width in the *screenSizes* property for the migrated text form. However, this screen size is no longer supported in EGL COBOL generation. If you generate for COBOL, there will be an error message from generation.

User Response: If you plan to generate for COBOL, edit the text form in EGL and remove the depth and width from the *screenSizes* property or change the depth and width to a size that is supported. You might also need to modify the text form to reposition the variables and constants to better fit the new depth and width.

IWN.MIG.0506.e Map Group *mapGroupName*, **map** *mapName* **- Unprotected constant at row, column; changed to protect=skip.**

Explanation: VisualAge Generator permits unprotected constants on both display and printer maps. EGL requires that constants be specified as either protect=skip or protect=no. The migration tool sets protect=skip for the field.

User Response: No action is required if protect=skip is acceptable. With protect=skip, the end user can continue typing at the end of any immediately preceding variable field and the additional characters will be placed in the next unprotected variable field. Protect=no prevents the end user for continuing to type at the end of any immediately preceding variable field. The end user must tab to the next variable field to continue typing.

IWN.MIG.0507.w Map Group *mapGroupName*, **map** *mapName* **- constant at row=0, column=0 changed to row=1, column=1.**

Explanation: VisualAge Generator tolerates, but does not fully support, a constant at position row=0, column=0 on a map. Fields at row=0, column=0 cannot specify any attribute information. EGL does not support any field at row=0, column=0. The field at row=0, column=0 is a constant and the first byte is initialized to blank. The migration tool changes the position to row=1, column=1 and deletes the first byte of the constant value. The migration tool does not include any field presentation properties such as color or highlighting for the field because this information was not recorded in the External Source Format file.

User Response: Test any programs that use this form to determine if there is a change in the appearance of the display. If there is, edit the form and set the field presentation properties to obtain the desired appearance.

IWN.MIG.0508.e Map Group *mapGroupName*, **map** *mapName* **- constant at row=0, column=0 cannot be changed.**

Explanation: VisualAge Generator tolerates, but does not fully support, a constant at position row=0, column=0 on a map. Fields at row=0, column=0 cannot specify any attribute information. EGL does not support any field at row=0, column=0. The field at row=0, column=0 is a constant and the first byte is **not** initialized to blank. The migration tool does not change the position for the field because this could cause the constant to be moved or eliminated from the form and change the appearance. The migration tool does not include any field presentation properties such as color or highlighting for the field because this information was not recorded in the External Source Format file. There will be an error in the Problems view.

User Response: Edit the form and change the constant field to position the field at row=1, column = 1. If necessary, modify the constant field to eliminate one byte to compenstate for the attribute byte that now occupies row=1, column=1. Be sure to test any programs that use this form to determine if there is a change in the appearance of the display. If there is, edit the form and set the field presentation properties to obtain the desired appearance.

IWN.MIG.0509.e Map Group *mapGroupName*, **map** *mapName* **- variable at row=0, column=0 cannot be changed.**

Explanation: This map might be from an older version of Cross System Product or VisualAge Generator. VisualAge Generator 4.5 does not support variables at row=0, column=0. Fields at row=0, column=0 cannot specify any attribute information. EGL does not support any field at row=0, column=0. The field at row=0, column=0 is a variable field. The migration tool does not change the position for the field because this would either cause the field to be moved or result in the loss of the first byte of data. The migration tool does not include any presentation properties such as color or highlighting for the field because this information was not recorded in the External Source Format file. There will be an error in the Problems view.

User Response: Edit the form and change the field to position the field at row=1, column=1. If necessary, modify other fields around the variable field to avoid the loss of any data due to the attribute byte that now occupies row=1, column=1. Be sure to test any programs that use this form to determine if there is a change in the appearance of the display. If there is, edit the form and set the field presentation properties to obtain the desired appearance.

IWN.MIG.0510.e Map Group *mapGroupName*, **map** *mapName* **-** *mapName* **conflicts with program name.**

Explanation: The program uses a map group or help map group that contains a map that is named the same as the program. VisualAge Generator permits the map name to be the same as the program name. EGL does not permit the form name to be the same as the program name. The migration tool renames a map in the program's help map group if the map name is the same as the program name and the map does not have any variable fields. However, the migration tool does not rename a map in the following situations:

- The map is a map with variable fields in the program's help map group.
- The map is any map in the program's main map group.

User Response: Edit the form group and change the name of the form. Also be sure the change the form definition and all references to this form in all programs that use the form group. For additional considerations, see the information on map names in "Map names and help map names" on page 54.

IWN.MIG.0601.w Function *functionName*, **I/O object** *recordName* **- Unable to determine record type for UPDATE option; non-SQL record assumed.**

Explanation: For SQL, if there are multiple UPDATE or SETUPD statements in a program, VisualAge Generator requires that the REPLACE function specifies the name of the corresponding UPDATE or SETUPD statement. EGL uses the resultSetID for SQL statements to specify the relationship between a replace statement and its corresponding get or open statement. The record specified by *recordName* is not available during migration. The migration tool assumed that the UPDATE function is for a non-SQL record and did not include the resultSetID.

User Response: If validation or generation flags an error because there are multiple get or open statements for the same record in the program, edit the function and add a resultSetID to the get for update statement. The resultSetID must be unique within the program. The recommended resultSetID is the the function name followed by the Result Set suffix preference you used during migration. For additional considerations, see the section on ″SQL I/O with multiple updates″ on "SQL I/O with multiple updates" on page 73.

IWN.MIG.0602.w Function *functionName* **- Unable to determine map type for I/O object** *mapName*; **display map assumed.**

Explanation: VisualAge Generator uses the DISPLAY I/O option for both display and printer maps. EGL uses the display statement only for text forms and the print statement for print forms. In VisualAge Generator Compatibility mode, the display statement can also be used for print forms. The map specified as *mapName* is not available during migration. The migration tool assumes that the map is a display map and migrates to the EGL display statement.

User Response: No action is required as long as you continue to use VisualAge Generator Compatibility mode or if the map is a display map. If the map is a print map and you want to discontinue use of VisualAge Generator Compatibility mode, you must change the function to use the print statement. For additional considerations, see "DISPLAY statement for maps" on page 66.

IWN.MIG.0603.e Function *functionName*, **SQL I/O object** *recordName* **- Unable to determine SQL table name(s).**

Explanation: VisualAge Generator determines the SQL table names from the SQL record at test and generation time. EGL requires that the table names be included in any modified SQL statement. The record specified by *recordName* is not available during migration. The migration tool uses EZE_UNKNOWN_SQLTABLE for the table name to insure that validation and generation will flag an error. The migration tool also sets the table label for the statement to T1.

User Response: Edit the function and specify the correct table name(s) and table label(s) based on the record definition. The table names are in either or both of the *tableNames* and *tableNameVariables* properties in the EGL record definition. For additional considerations, see the information on SQL I/O options in "SQL I/O and !itemColumnName" on page 72.

IWN.MIG.0604.e Function *functionName*, **SQL I/O object** *recordName* **- Unable to determine column names for !itemColumnName variable(s).**

Explanation: The modified SQL statement used one or more VAGen !itemColumnName variables. A VAGen !itemColumnName variable specifies the name of an item in the SQL record definition which corresponds to the actual SQL column name. VisualAge Generator determines the actual SQL column names for any !itemColumnName variables from the SQL record at test and generation time. EGL does not support !itemColumnName variables. Instead, EGL requires that

the actual SQL column names be used in any modified SQL statement. The record specified by *recordName* is not available during migration. The migration tool uses the !itemColumnNames in the modified SQL statement to provide as much information as possible.

User Response: Edit the function and specify the SQL column names based on the record definition. For each !itemColumnName, locate the corresponding item in the SQL record definition. The column name for that item is the column name you need to use in the EGL I/O statement. For additional considerations, see the information on !itemColumnNames in "SQL I/O and !itemColumnName" on page 72.

IWN.MIG.0605.w Function *functionName*, **SQL I/O object** *recordName* **- SQLEXEC with model=none and no SQL clauses.**

Explanation: The SQLEXEC statement specifies a model type of none, but does not contain any SQL clauses. VisualAge Generator in effect generates a no op for this statement. The migration tool generates an EGL no op statement (just a semi-colon) and includes a VAGen Info comment to indicate that the model type was none. If the VAGen function specifies an error routine, the migration tool includes the *try*, *onException*, and *end* statements appropriate for that error routine.

User Response: Review the function to determine whether the I/O statement should be eliminated or expanded.

IWN.MIG.0606.w Function *functionName* **- Unable to determine I/O object type for** *ioObjectName* **for CONVERSE; map assumed.**

Explanation: VisualAge Generator uses the CONVERSE I/O option for both display maps and UI records. EGL uses the *converse* statement, but with different syntax, for both text forms and pageHandlers. The part specified as the *ioObjectName* is not available during migration. The migration tool assumes that the I/O object is a display map and migrates to the EGL *converse* statement using the syntax required for a text form.

User Response: If EGL validation or generation flags an error, then change the *converse* statement to the EGL syntax required for a pageHandler.

IWN.MIG.0607.e Function *functionName*, **SQL I/O object** *recordName* **- Unable to determine SQL I/O clause** *clauseName*.

Explanation: In VisualAge Generator, at some points in time, only the SQL clause that was modified was saved with the function. In this situation, VisualAge Generator creates the remaining clauses from the record definition that is specified as the I/O object for the function. The specified *recordName* is not available during migration. The migration tool is unable to create the SQL clause. The *clauseNames* that might be listed in this message include: SELECT, INTO, WHERE, ORDERBY, INSERTCOLNAME, VALUES, FORUPDATEOF, and SET. The migration tool builds a skelton clause and includes EZE_UNKNOWN_SQL_CLAUSENAME.

User Response: Locate the record specified in the message. Edit the function to include the missing SQL clauses. To determine what the missing SQL clauses need to be, use VAGen SQL Statement Editor to view the SQL clauses. See "SQL I/O and missing required SQL clauses" on page 70 for more details and potential problems.

IWN.MIG.0608.e Function *functionName*, **SQL I/O object** *recordName* **- Unable to determine SQL I/O clause** *clauseName* **for alternate specification** *altspecRecordName*.

Explanation: In VisualAge Generator, at some points in time, only the SQL clause that was modified was saved with the function. In this situation, VisualAge Generator creates the remaining clauses from the record definition that is specified as the I/O object for the function. The specified *recordName* is available during migration. However, recordName specifies an alternate specification record *altspecRecordName* which is not available during migration. The migration tool is unable to create the SQL clause. The *clauseNames* that might be listed in this message include: SELECT, INTO, WHERE, ORDERBY, INSERTCOLNAME, VALUES, FORUPDATEOF, and SET. The migration tool builds a skelton clause and includes EZE_UNKNOWN_SQL_*CLAUSENAME*.

User Response: Locate the alternate specification record specified in the message. Edit the function to include the missing SQL clauses. To determine what the missing SQL clauses need to be, use VAGen SQL Statement Editor to view the SQL clauses. See "SQL I/O and missing required SQL clauses" on page 70 for more details and potential problems.

IWN.MIG.0701.e Function *functionName* **- Unable to determine map type for** *mapName* **used in SET map PAGE statement; used sysLib.EZE_SETPAGE();**

Explanation: VisualAge Generator uses SET map PAGE to indicate that the screen is to be cleared for a display map or that a page eject is to occur for a printer map. EGL uses the *sysLib.clearScreen()* statement only for text forms and the *sysLib.pageEject* statement for print forms. The map specified as *mapName* is not available during migration. The migration tool does not make an assumption about the map type. Instead, the migration tool uses the sysLib.EZE_SETPAGE() statement to insure that validation and generation will flag an error. The migration tool includes the original map name as a comment.

User Response: Review the function and determine whether *clearScreen()* or *pageEject()* is the correct choice. For additional considerations, see the information on the SET map PAGE statement in "SET map PAGE statement" on page 76.

IWN.MIG.0702.e Function *functionName* **- Unable to determine return column name for RETR statement due to missing table** *tableName***.**

Explanation: If the return column is not specified on a RETR statement, VisualAge Generator automatically determines the return column name based on the second column of the specified table. The EGL replacement for RETR is an *if* statement, followed by an assignment statement. The return column name must be explicitly specified in the assignment statement. The table specified by *tableName* is not available during migration. The migration tool uses EZE_UNKNOWN_RETURN_COLUMN to insure that validation and generation will flag an error. If this problem occurs in program flow statements, the program name appears in the message instead of a function name.

User Response: Edit the function and specify the correct return column based on the table definition. The second column in the table is the default return column that is used in VisualAge Generator. For additional considerations, see the information on the RETR statement in "RETR statement" on page 75.

IWN.MIG.0703.e Function *functionName* **- Unable to determine search column name for RETR statement due to missing table** *tableName***.**

Explanation: If the search column is not specified on a RETR statement, VisualAge Generator automatically determines the search column name based on the first column of the specified table. The EGL replacement for RETR is an *if* statement, followed by an assignment statement. The search column name must be explicitly specified in the *if* statement. The table specified by *tableName* is not available during migration. The migration tool uses EZE_UNKNOWN_SEARCH_COLUMN to insure that validation and generation will flag an error. If this problem occurs in program flow statements, the program name appears in the message instead of a function name.

User Response: Edit the function and specify the correct search column based on the table definition. The first column in the table is the default search column that is used in VisualAge Generator. For additional considerations, see the information on the RETR statement in the section "RETR statement" on page 75.

IWN.MIG.0704.e Function *functionName* **- Unable to determine search column name for FIND statement due to missing table** *tableName***.**

Explanation: If the search column is not specified on a FIND statement, VisualAge Generator automatically determines the search column name based on the first column of the specified table. The EGL replacement for FIND is an *if* statement, followed by a function invocation statement. The search column name must be explicitly specified in the *if* statement. The table specified by *tableName* is not available during migration. The migration tool uses EZE_UNKNOWN_SEARCH_COLUMN to insure that validation and generation will flag an error. If this problem occurs in program flow statements, the program name appears in the message instead of a function name.

User Response: Edit the function and specify the correct search column based on the table definition. The first column in the table is the default search column that is used in VisualAge Generator. For additional considerations, see the information on the FIND statement in "FIND statement" on page 75.

IWN.MIG.0705.e Function *functionName* **— Uses CALL CSPTDLI statement.**

Explanation: The specified function contains a CALL CSPTDLI statement which is not supported in this release of EGL.

User Response: You must wait until a future release of EGL to test or generate any program that uses this function.

IWN.MIG.0706.e Function *functionName* **- Unable to determine record type for** *recordName* **used in IF, WHILE, or TEST DUP statement; used EZE_DUPLICATE.**

Explanation: VisualAge Generator supports checking both DUP and UNQ for both non-SQL and SQL records. For SQL records, DUP and UNQ are identical. EGL supports both duplicate and unique for non-SQL records. EGL only supports unique for SQL records. The record specified by *recordName* is not available during migration. The migration tool migrates DUP to EZE_DUPLICATE to insure that validation and generation will flag an error.

Note: The migration tool migrates the TEST statement to an *if* statement.

User Response: Edit the function and change EZE_DUPLICATE to one of the following:

- *unique* for an SQL record
- *duplicate* for a non-SQL record

For additional considerations, see the information on checking for DUP in "I/O error values UNQ and DUP" on page 79.

IWN.MIG.0707.e Function *functionName* **- Unable to determine if item** *itemName* **is in a record or map when used in IF, WHILE, or TEST NULL statement; used EZE_NULL.**

Explanation: VisualAge Generator supports checking for NULL for both a map item and an SQL item. Checking a map item for NULL is equivalent to checking it for blanks. Checking an SQL item for NULL checks the null indicator variable to determine if the column is null in the database. The equivalent EGL statement is to check a map item for blanks and an SQL item for null. The item specified in *itemName* is not available during migration. The migration tool migrates NULL to EZE_NULL to insure that validation and generation will flag an error.

Note: The migration tool migrates the TEST statement to an *if* statement.

User Response: Edit the function and change EZE_NULL to one of the following:

- *blanks* for a map item
- *null* for an SQL item

For additional considerations, see the information on checking for NULL in "Checking SQL and map items for NULL" on page 78.

IWN.MIG.0708.w Function *functionName* **- Uses EZESYS in statement other than IF, WHILE, or TEST; old VAGen values will be used.**

Explanation: VisualAge Generator supports the use of EZESYS in statements other than IF, WHILE, and TEST. The migration tool migrates EZESYS based on the statement type. In IF, WHILE, and TEST statements, the migration tool converts EZESYS to sysVar.systemType and also converts the values to the new EGL values. For statements other than IF, WHILE, or TEST, the migration tool converts to *custPrefixEZESYS*, where *custPrefix* is the Renaming Prefix preference you set for migration. When migrating programs, the migration tool always includes a declaration for *custPrefixEZESYS* and a statement to initialize *custPrefixEZESYS* to the original VAGen values. The original VAGen values will be used in this statement.

User Response: Review the function and determine whether you want to use the original VAGen values or the new EGL values. If you want to use the VAGen values, no change is required. If you want to use the new EGL values, change *custPrefixEZESYS* to sysVar.systemType.

IWN.MIG.0709.e Function *functionName* **- Unable to determine type for part** *partName* **for an XFER; UI record assumed.**

Explanation: VisualAge Generator uses the XFER statement to transfer to another program or transaction while first presenting either a map or a UI record to the end user. This release of EGL does not support web transactions or UI records. The migration tool is unable to determine that a map is being used. The migration tool assumes the statement is XFER with a UI record because that is more frequently used than XFER with a map. The migration tool migrates the function, using a forward statement. This is an intelligent guess as to EGL syntax that will replace XFER with a UI record in a future release. This preserves as much of your logic as possible.

User Response: Review the function. If the function is intended for use with maps, change the forward statement to a show statement. See "XFER" on page 82 or the online EGL helps for details of the show statement syntax. If the function is intended for use with web transactions, you might want to move it to a different project to avoid having errors on the EGL Problems view.

IWN.MIG.0801.e Program name *programName* **is a reserved word. It must be renamed.**

Explanation: The migration tool does not rename programs for you.

User Response: You must change the name of the program and all references to it, including references on call, transfer, and show statements. Also change the names of any bind control or linkedit parts that correspond to this program. If you want to keep the original program name as the name for the generated program, you can specify the *alias* property. If you do not specify the *alias* property, be sure to change any non-EGL references to the program name, including CICS program definitions.

IWN.MIG.0802.w Program *programName* **— Allows implicit items. Migration does not create definitions for implicit items.**

Explanation: In VisualAge Generator, a program can specify that it allows implicit data items. If a program that allows implicit data items actually uses an item without defining it, VisualAge Generator automatically creates the definition for you at test and generation time. EGL does not allow implicit items. The migration tool does not create implicit definitions for you.

User Response: Validate the program in VisualAge Generator to determine if any implicit items are being used. If so, VisualAge Generator provides the definitions for the implicit items in the validation messages. In the Rational Developer product, edit the program definition and add the corresponding EGL

data item declarations. You do not need to create a record to contain the items. You can add the item declarations directly to the program definition.

IWN.MIG.0803.w Program *programName* **— Uses PSB** *psbName***.**

Explanation: The program specifies a PSB name in its program definition. EGL does not support the IMS and IMS BMP environments in this release. EGL also does not support DL/I in this release. The migration tool creates a use declaration statement for the PSB but comments it out. The program might or might not be a valid program, depending on its intended runtime environment and its use of DL/I.

User Response: Review the program to determine its intended runtime environment and whether it uses DL/I, including any use of CSPTDLI. If the VAGen program is generated for both the IMS and CICS environments and does not use DL/I, the PSB is used only to interface to the IMS environment. You can generate the EGL program for the CICS environment in this release. You must wait until a future release of EGL to generate for the IMS environment. If the program uses DL/I, you must wait until a future release of EGL to generate the program for either the CICS or IMS environments.

IWN.MIG.0804.w Program *programName* **- Unable to determine part type for I/O object** *partName* **used with CLOSE I/O option; record assumed.**

Explanation: In VisualAge Generator, the I/O objects are automatically included at test or generation time. The CLOSE I/O option can be used for both records and print maps. In EGL, records used in I/O statements must be explicitly declared in the program. Forms are not explicitly declared, but there must be a *use* declaration for the form group. The CLOSE I/O option is used in the specified program and the specified *partName* is used as the I/O object for the CLOSE. However, the specified *partName* is not available during migration. The migration tool assumes that the part is a record and includes the data declaration.

User Response: If the migration tool guesses incorrectly, there will be an error in the Problems view. Edit the program and remove the data declaration for the print form.

IWN.MIG.0805.w Program *programName* **- execution mode not specified; nonsegmented assumed.**

Explanation: In VisualAge Generator, at some points in time, the execution mode was not saved with the program part. Execution mode only applies to main transaction programs. The specified *programName* is a main transaction program, but does not include the execution mode in the external source format. The migration tool assumes that the execution mode is nonsegmented and includes the segmented=no property in the EGL source.

User Response: No action is required if the program should run in nonsegmented mode. If the program should run in segmented mode, edit the program and change the segmented property to segmented=yes.

IWN.MIG.1001.e Generation options part *partName* **is a reserved word. It must be renamed.**

Explanation: The migration tool does not rename programs for you. Because a program might have a special generation options part named as programName.opt, the migration tool also does not rename generation options parts.

User Response: When you change the program name, be sure to change the name of the corresponding generation options part.

IWN.MIG.1002.w Generation options part *partName* **- /dbms=odbc is migrated to dbms="DB2".**

Explanation: The specified generation options part includes the VAGen generation option /dbms=odbc. EGL only supports DB2 or Oracle. The migration tool converts /dbms=odbc to dbms="DB2" in the EGL build descriptor part. EGL provides DB2 support by using a JDBC driver. If you have a JDBC driver for your database, you might be able to use the build descriptor option dbms="DB2" as the database type.

User Response: Be sure to migrate, generate, and test a variety of VAGen programs that used ODBC support to ensure that all the functions you require work correctly with your JDBC driver.

IWN.MIG.1003.e Generation options part *partName* **- /system=***systemType* **is not supported.**

Explanation: The specified generation options part includes the VAGen /system generation option and specifies a runtime environment that is not supported by EGL. The migration tool converts the /system generation option to a comment in the EGL build descriptor part.

User Response: Determine whether this build descriptor part is used by other build descriptor parts. If not, you can delete the build descriptor part. Alternatively, you might want to keep the build descriptor part for reference if EGL supports this runtime environment at sometime in the future.

IWN.MIG.1004.w Generation options part *partName* - /system=*systemType* requires that destPort be set.

Explanation: The specified generation options part includes the /system generation option and specifies either MVSCICS or MVSBATCH as the runtime environment. The EGL build process requires you to specify a destination port using the destPort build descriptor option. If you do not specify the destPort build descriptor option, the default port is 5555.

User Response: Modify the build descriptor part that corresponds to the generation options part and include the destPort build descriptor option. If the default port of 5555 is acceptable, you do not need to modify the build descriptor part.

IWN.MIG.1005.e Generation options part *partName* - /system=*systemType* is not currently supported.

Explanation: The specified generation options part includes the VAGen /system generation option and specifies a runtime environment that is not currently supported by EGL. The migration tool converts the /system generation option to the EGL build descriptor option for future use. However, because the runtime environment is not currently supported by EGL, the value will not appear in the Build Descriptor Parts Editor. You can see the value by using a text editor.

User Response: None. You should keep this build descriptor part for possible use in future releases of EGL.

IWN.MIG.1099.e Control part *partName* - symparm *symparmName* is not supported.

Explanation: The specified control part uses or sets a symparm which is not supported in EGL. The migration tool migrates the symparm "as is" using the original VAGen symparm name. However, this symparm is not set during generation.

User Response: Modify the control part to set a default value for the symparm. Alternatively, modify the control part so that it no longer uses the specified symparm.

IWN.MIG.1101.e Linkage table part *partName* is a reserved word. It must be renamed.

Explanation: The migration tool does not rename control parts for you.

User Response: Modify the linkage options part name so that it is not a reserved word. When you change the linkage options part name, be sure to change all the build descriptor parts that reference the linkage options part.

IWN.MIG.1102.e Linkage table *partName* - /contable=BINARY is not supported. It must be changed.

Explanation: VisualAge Generator supports /contable=BINARY in the linkage table part. EGL does not support this value. The migration tool includes the conversionTable="BINARY" value in the EGL linkage table part. This value is invalid, but will not be detected until generation.

User Response: You must change the conversionTable value to a value that is supported by EGL. Refer to the information about linkage parts in the online helps for details about the EGL conversionTable attribute and the options that are available.

IWN.MIG.1103.e Linkage table part *partName* - /remotecomtype=CICSCLIENT is not supported. Defaulted to CICSECI.

Explanation: VisualAge Generator supports /remotecomtype=CICSCLIENT in the linkage table part. EGL does not support this value. The migration tool includes the remoteComType="CICSECI" in the EGL resource assocations part. This value is valid, but might not be what you plan to use. If you want to use CICSECI, you need to set the ctgPort and ctgLocation.

User Response: If you plan to use CICSECI, modify the linkage table part and set the values of ctgPort and ctgLocation for the entry that specifies CICSECI as the remoteComType. If you do not plan to use CICSECI, refer to the information about linkage parts in the online helps for details about the EGL remoteComType attribute and the options that are available in EGL.

IWN.MIG.1104.e Linkage table part *partName* - /remotecomtype=*communicationType* is not supported. It must be changed.

Explanation: VisualAge Generator supports /remotecomtype=*communicationType* in the linkage table part. EGL does not support this communications protocol. The migration tool includes the remoteComType="*communicationType*" in the EGL linkage table part. This value is not valid and must be changed.

User Response: Determine the communication protocol that you plan to use. Then edit the part and change the remoteComType to a value that is supported by EGL. Refer to the information about linkage parts in the online helps for details about the EGL remoteComType attribute and the options that are available in EGL.

IWN.MIG.1201.e Resource association part *partName* is a reserved word. It must be renamed.

Explanation: The migration tool does not rename control parts for you.

User Response: Modify the resource association part name so that it is not a reserved word. When you change the resource associations part name, be sure to change all the build descriptor parts that reference the resource association part.

IWN.MIG.1202.e Resource association part *partName* -/filetype=*fileType* is not supported. It must be changed.

Explanation: VisualAge Generator supports /filetype=BTRIEVE and /filetype=MFCOBOL for some workstation environments. EGL does not support these file types. The migration tool includes the filetype information in the EGL resource association part. The value is invalid and will cause an error in the Problems view.

User Response: You must change the filetype value to a value that is supported by EGL. Refer to the information about resource association parts in the online helps for details about the EGL filetype attribute and the options that are available.

IWN.MIG.1203.e Resource association part *partName* -/system is *targetSystem*, which is not supported; migrated based on /filetype *fileType* information.

Explanation: The resource association part contains an entry that uses the specified *targetSystem*. This target system is not supported in EGL. The migration tool migrates the resource association entry based on the *fileType*. For example, if the targetSystem is mvs* and the fileType is transient, the migration tool creates an EGL resource association entry and sets the EGL system to mvs*. This will be invalid and result in an error in the Problems view. You can correct the entry by specifying a valid EGL system (zoscics for this example). If the targetSystem is ims* and the fileType is smsgq, the migration tool migrates the resource association entry to a comment to preserve as much of your information as possible for future reference.

User Response: If there is an error in the Problems view, correct the entry in the resource associations part by specifying a valid target system.

IWN.MIG.1301.e Linkedit part *partName* is a reserved word. It must be renamed.

Explanation: The migration tool does not rename programs for you. Because the program name must match its corresponding linkedit part, the migration tool also does not rename the linkedit part.

User Response: When you change the program name, be sure to change the name of the corresponding linkedit part.

IWN.MIG.1401.e Bind control part *partName* is a reserved word. It must be renamed.

Explanation: The migration tool does not rename programs for you. Because the program name must match its corresponding bind control part, the migration tool also does not rename the bind control part.

User Response: When you change the program name, be sure to change the name of the corresponding bind control part.

Messages from the EGL into the Rational Developer product migration tool—Stage 3

The only messages produced by Stage 3 are trace messages.

Appendix D. Messages in the Problems view

In an ambiguous situation, the migration tool is not always able to determine the correct EGL syntax to build during migration. This typically occurs when an associated part is not available during migration. In these cases, the migration tool sometimes creates intentionally invalid EGL syntax so that an error will appear in the Problems view. The table below lists the specific text string that will cause an error in EGL validation. The specific EGL error message might vary, but the text string listed in the left column will appear near the EGL statement that is flagged as an error. Whenever the migration tool includes these text strings, the tool also issues a message to the migration log.

Table 133. VAGen migration text that causes EGL syntax errors

VAGen migration text in EGL syntax	Problem and Solution
###KEYS_NOT_FOUND###	**Problem:** The current SQL record embeds another record's structure. During migration the record named on the embed statement was not available. Any key item specified for the current SQL record in VAGen is included in the *keyItems* property, but the keys from the embedded record are missing. **Solution:** Find the record named on the embed statement. Replace the ###KEYS_NOT_FOUND### text with the keys listed in the embedded SQL record. Be sure to merge the embedded record keys with the current record's key item in the order that the items appear in the record structure of the embedded record. If the current record's key item is also specified as key=yes in the embedded record, only include the item once in the EGL *keyItems* property.
###TABLES_NOT_FOUND###	**Problem:** The current SQL record embeds another record's structure. During migration the record named on the embed statement was not available. **Solution:** Find the record named on the embed statement and copy the *tableNames* and *tableNameVariables* properties into the current SQL record.
EZE_DUPLICATE	**Problem:** The record named on a VAGen IF, WHILE, or TEST statement was not available during migration. **Solution:** Find the record named on the EGL *if* or *while* statement. Change EZE_DUPLICATE to one of the following: • *duplicate* for a non-SQL record • *unique* for an SQL record
EZE_NULL	**Problem:** The migration tool could not determine whether the item named on a VAGen IF, WHILE, or TEST statement is in an SQL record or on a map. **Solution:** Review the program and determine whether the item is in an SQL record or on a form. Replace EZE_NULL with *null* for an SQL item or *blanks* for a form field.

Table 133. VAGen migration text that causes EGL syntax errors (continued)

VAGen migration text in EGL syntax	Problem and Solution
EZE_SETPAGE();	**Problem:** The map named on a VAGen SET map PAGE statement was not available during migration. **Solution:** Find the map named on the // VAGen Info comment that accompanies the EZE_SETPAGE() statement. Change EZE_SETPAGE to one of the following: • *clearScreen()* for a display map • *pageEject()* for a printer map
EZE_UNKNOWN_PARTTYPE	**Problem:** The External Source Format stored in the migration database was not valid. The migration tool was not able to determine the part type and was not able to convert the part to EGL syntax. **Solution:** The part named on the EZE_UNKNOWN_PARTTYPE statement is not valid. If this problem only occurs for a few parts, try exporting External Source Format from VisualAge Generator and migrating these parts in single file mode. If you created your own tool to load the migration database, there might be a problem with the way the tool is loading External Source Format code into the migration database. See Appendix G, "Migration Database," on page 309 for some queries that might be useful in determining what is causing the problem.
EZE_UNKNOWN_RETURN_COLUMN	**Problem:** The VAGen table named on the VAGen RETR statement was not available during migration. **Solution:** Find the EGL Data Table named on the assignment statement and replace EZE_UNKNOWN_RETURN_COLUMN with the name of the second column in the table.
EZE_UNKNOWN_SEARCH_COLUMN	**Problem:** The VAGen table named on the VAGen FIND or RETR statement was not available during migration. **Solution:** Find the EGL Data Table named on the *if* statement and replace EZE_UNKNOWN_SEARCH_COLUMN with the name of the first column in the table.
EZE_UNKNOWN_SQLTABLE	**Problem:** The SQL record named as the I/O object was not available during migration. The migration tool was not able to determine the correct tables clause for the EGL I/O statement. **Solution:** Find the record named on the I/O statement and determine the correct tables clause from the record's *tableNames* and / or *tableNameVariables* properties.

Table 133. VAGen migration text that causes EGL syntax errors (continued)

VAGen migration text in EGL syntax	Problem and Solution
EZE_UNKNOWN_SQL_FORUPDATEOF	**Problem:** VisualAge Generator created a default *for update of* clause for the SQL UPDATE or SETUPD I/O option. The SQL record named as the I/O object was not available during migration. Therefore, the migration tool was not able to determine the correct *for update of* clause for the EGL I/O statement. **Solution:** Find the record named on the I/O statement and determine the correct *for update of* clause from the record's data items list. The default *for update of* clause in VisualAge Generator is the list of column names from the record in the same order as the items are listed in the record, but omitting the following: • Any column name that is listed in the EGL keyItems property for the record. • Any column name that is specified with the EGL isReadOnly=yes property. If the record named on the I/O statement embeds another SQL record, do the following: • Use the record named on the *embed* statement to determine the order of the columns and the isReadOnly=yes property. • Use the record named on the I/O statement (the embedding record) to determine the keyItems property. If the *for update of* clause is used in an EGL *prepare* statement, enclose the list of column names within double-quotes.
EZE_UNKNOWN_SQL_INSERTCOLNAME	**Problem:** VisualAge Generator created a default list of columns for the SQL ADD I/O option. The SQL record named as the I/O object was not available during migration. Therefore, the migration tool was not able to determine the correct list of column names for the EGL *add* statement. **Solution:** Find the record named on the I/O statement and determine the correct list of columns from the record's data items list. The default list of column names in VisualAge Generator is the list of column names from the record in the same order as the items are listed in the record, but omitting any column name that is specified with the EGL isReadOnly=yes property. If the record named on the I/O statement embeds another record, use the record named on the embed statement to determine the order of the columns and the isReadOnly=yes property. This list of column names is never used in an EGL *prepare* statement.
EZE_UNKNOWN_SQL_INTO	**Problem:** VisualAge Generator created a default list of data items for the *into* clause for the SQL INQUIRY, SETINQ, UPDATE, or SETUPD I/O option. The SQL record named as the I/O object was not available during migration. Therefore, the migration tool was not able to determine the correct *into* clause for the EGL I/O statement. **Solution:** Find the record named on the I/O statement and determine the correct list of items for the *into* clause. The default list of item names in VisualAge Generator is the list of items from the record in the same order as the items are listed in the record. If the record named on the I/O statement embeds another record, use the record named on the *embed* statement to determine the order of the data items. The *into* clause is never included on an EGL *prepare* statement.

Table 133. VAGen migration text that causes EGL syntax errors (continued)

VAGen migration text in EGL syntax	Problem and Solution
EZE_UNKNOWN_SQL_SELECT	**Problem:** VisualAge Generator created a default list of data items for the *select* clause for the SQL INQUIRY, SETINQ, UPDATE, or SETUPD I/O option. The SQL record named as the I/O object was not available during migration. Therefore, the migration tool was not able to determine the correct *select* clause for the EGL I/O statement. **Solution:** Find the record named on the I/O statement and determine the correct list of column names for the *select* clause. The default list of column names in VisualAge Generator is the list of column names from the record in the same order as the items are listed in the record. If the record named on the I/O statement embeds another record, use the record named on the *embed* statement to determine the order of the columns. If the *select* clause is used in an EGL *prepare* statement, enclose the list of column names within double-quotes.
EZE_UNKNOWN_SQL_VALUES	**Problem:** VisualAge Generator created a default list of data items to provide the values for the SQL ADD I/O option. The SQL record named as the I/O object was not available during migration. Therefore, the migration tool was not able to determine the correct list of item names for the EGL *add* statement. **Solution:** Find the record named on the I/O statement and determine the correct list of items from the record's data items list. The default list of item names in VisualAge Generator is the list of item names from the record in the same order as the items are listed in the record, but omitting any item name that is specified with the EGL isReadOnly=yes property. If the record named on the I/O statement embeds another record, use the record named on the *embed* statement to determine the order of the columns and the isReadOnly=yes property. The *values* clause is never used in an EGL *prepare* statement.

Appendix E. IWN.xxx messages in the Problems view

Some IWN.SYN, IWN.VAL, and IWN.XML messages are more likely to occur for EGL source code that was migrated from VisualAge Generator than for code that you develop completely within EGL. This section lists messages that have a special meaning for migrated code.

IWN.VAL.4300.e The part named *partName* could not be resolved or did not resolve to one of the following types: *partType*

Explanation: There are several situations in which this can occur:

- The specified control part does not exist.
- The specified control part is in a different project or package from the control part that has the error. The migration tool does not create import statements for control parts because control parts do not have associates in VisualAge Generator.
- The XML parser was not able to completely process the .eglbld file. In this case, the specified control part might exist in the same file as the control part that has the error. Check for message *IWN.XML.3999.e XML Validation Error - Attribute "xxxxx" was already specified for element "yyyyy"*. This message indicates that the attribute *xxxxx* is specified multiple times in the same control part.

User Response: If the specified control part does not exist, create the part or remove the reference to it.

If the specified control part is in a different project, update the EGL Build Path in the current project's properties to include the project where the specified control part resides. If the specified control part is in a different package, edit the current .eglbld file to add an import statement for the package where the specified control part resides.

If the message IWN.XML.3999.e occurs indicating that attribute *xxxxx* was already specified, edit the current .eglbld file so that there is only one specification for *xxxxx* in the control part. When you save the .eglbld file, the messages in the Problems view should be updated. Because the XML parser stops processing at the first duplicate attribute in the .eglbld file, you might have to resolve several errors before the entire file can be parsed. When all the IWN.XML.3999.e messages have been resolved, the specified control part should be available if it is in the same .eglbld file as the referencing control part.

IWN.VAL.4929.e The use statement for *formGroupName* in program *programName* does not resolve.

Explanation: The program specified a map group in VisualAge Generator but did not have any display or converse I/O options in the program, nor any maps as a called parameter. Another possibility is that the program specified a help map group, but none of the maps that the program uses specify a help map.

User Response: Change the program to do one of the following: Remove the use statement for the formGroup or add an import statement for the package containing the formGroup. You might need to create a formGroup part if one did not exist in VisualAge Generator.

IWN.VAL.5100.e sysVar is an invalid qualifier for the *xxxxx* system word.

Explanation: An EZE word that was valid in VisualAge Generator is not currently supported by EGL. The migration tool migrates the EZE word, making a "best guess" as to what the EGL replacement might be in the future. This preserves your program logic.

User Response: Edit the function and make logic changes so that this value is no longer used. Alternatively, create a new project to preserve migrated functions that cannot currently be used. Move all functions that contain VAGen values that are not currently supported to this new project.

IWN.VAL.5101.e *mainFunctionName* It is invalid to use the *xxxxx* system word in this statement location.

Explanation: The program that uses that uses the specified main function in turn invokes other functions. One of the functions in the function invocation chain uses the specified system word in a statement. The migration tool always qualifies the EGL system words that are replacements for the VAGen EZE words. If the *xxxxx* system word is not qualified with sysLib, sysVar, mathLib, or strLib, the most likely causes are as follows:

- The VAGen program permitted implicit data items and the definition of *xxxxx* was automatically created

during generation. EGL does not permit implicit data items. The migration tool also does not create implicit data item definitions for you.

- The record, map, or table was not included in the migration set so the migration tool could not include the necessary import statement in the program.

User Response: Check whether the VAGen program allowed implicit items. If so, validate the program in VisualAge Generator. There will be a message on the VAGen View Messages list that provides the correct definition of the implicit data item. Add the definition for the data item to the declarations section of the program. If the VAGen program did not allow implicit items, create an associates list for the program in VisualAge Generator. From the associates list, use the VAGen References tool to search for the specified data item. The results of the References tool provide a clue to which record, map, or table might be missing from the migration set.

IWN.VAL.5168.e *xxxxx* **is not valid for use within an Is/Not expression.**

Explanation: In VisualAge Generator, the specified value was a valid value for EZESYS. This value has no corresponding value in EGL. The migration tool migrates the VisualAge Generator values to preserve your program logic.

User Response: Edit the function and make logic changes so that this value is no longer used. Alternatively, create a new project to preserve migrated functions that cannot currently be used. Move all functions that contain VAGen values that are not currently supported to this new project.

IWN.VAL.6506.e The *xxxxx* **SQL I/O statement allows only one** *yyyyy* **clause.**

Explanation: In VisualAge Generator, SQL keywords are permitted as column names. In EGL, certain SQL keywords are not permitted.

User Response: For the list of SQL keywords and techniques you can use to resolve this problem, see "SQL reserved words requiring special treatment" on page 174.

IWN.VAL.6620.e *functionName* **- The variable access** *xxxxx* **is ambiguous.**

Explanation: Determine if the problem occurs for a call statement and *xxxxx* is an unqualified data item. If the problem is for a call statement, check to see if the data item is in the Level 77 record associated with the program's *inputRecord* property. VisualAge Generator gives precedence to Level 77 items in the program's primary working storage record if an unqualified item is used on a CALL statement. However, EGL does not provide the same precedence for the call statement.

User Response: If the problem is for a call statement, you might be able to use the Level 77 record name as the qualification for this item. However, you must be sure that all programs that invoke this function use the same Level 77 record.

IWN.VAL.6695.e *functionName* **- The state** *XXXXX* **is not allowed for this item data reference.**

Explanation: If state is PROTECT, SKIP, INVISIBLE, BLINK, or a color, the data item is on a print form. VisualAge Generator tolerated setting these attributes for printer forms. EGL does not.

If state is EMPTY, determine if a form or record was not available during migration. EGL permits the definition of independent data items and assumes that if a type definition cannot be found the definition is for an item. EGL does not support the use of set empty for a data item.

User Response: If the problem is related to setting attributes on a print form, modify the function to remove the statement. Alternatively, if the same function is used with both a text form and a print form, you must create a copy of the function for use with print forms.

If the problem is related to a *set empty* statement, define or migrate the missing record or form.

IWN.VAL.7553.e *functionName* **- Argument** *n* **for** *systemFunctionName* **must be a string item, string constant or a string literal.**

Explanation: VisualAge Generator tolerated numeric (numc) items as character arguments in the EZE string functions. EGL does not support numeric items for the character arguments in the system string (strLib) functions.

User Response: Edit the program that uses the string function:

1. In the program, declare a new character item with the same length as the numeric item
2. In the function that invokes the string function, move the numeric item to the character item and then pass the character item as an argument to the system string function

This technique works even if you are using segmented programs. Aternatively, if you are not using segmented programs, you can define the new character item in the function that invokes the system string function.

IWN.XML.3997.e XML Validation Error - Attribute *"yyyyy"* **must be declared for element type** *"xxxxx"*.

Explanation: In VisualAge Generator, the option *yyyyy* is valid for element *xxxxx*. This combination is not supported by EGL. The migration tool migrates the

value even though it is invalid so there will be an error in the Problems view to remind you to resolve the problem.

User Response: Review the EGL online helps for the options that are valid for *xxxxx*. When you decide which option (or options) to use, you might need to open the build descriptor file with the Text Editor to be able to make the necessary change.

IWN.XML.3998.e XML Validation Error - Attribute "system" with value "*xxxxx*" must have a value from the list "ZOSCICS WIN USS ISERIESJ ZOSBATCH AIX LINUX".

Explanation: In VisualAge Generator, the value *xxxxx* is a valid target environment. This environment is not currently supported by EGL. The migration tool migrates the information for certain target environments to preserve the information for possible future use.

User Response: Create a new project to preserve migrated build descriptors that cannot currently be used. Move all build descriptor parts that contain VAGen values that are not currently supported to this new project. To move the part, use a text editor to open the build descriptor file to copy and paste the part that has the error.

IWN.XML.3999.e (first of two alternatives) XML Validation Error - Attribute "*xxxxx*" was already specified for element "*yyyyy*".

Explanation: Attribute *xxxxx* is specified multiple times in the same control part. The XML parser stops processing the .eglbld file. As a result, this error can mask other errors or cause additional errors to be listed in the Problems view.

User Response: Edit the current .eglbld file using the Text Editor so that there is only one specification for *xxxxx* in the control part. When you save the .eglbld file, the messages in the Problems view should be updated. Because the XML parser stops processing at the first duplicate attribute in the .eglbld file, you might have to resolve several errors before the entire file can be parsed.

IWN.XML.3999.e (second of two alternatives) XML Validation Error - The content of element type "*xxxxx*" must match "(*listOfValues*)".

Explanation: In VisualAge Generator, one or more of the values specified for a resource association is a valid value. This value is not supported by EGL. The migration tool migrates the value even though it is invalid so there will be an error in the Problems view to remind you to resolve the problem.

User Response: Review the EGL online helps for the options that are valid for *xxxxx*. When you decide which option to use, you might need to open the build descriptor file with the Text Editor to be able to make the necessary change.

Appendix F. Situations where incorrect External Source Format causes problems in creation of EGL

There are some situations in which the External Source Format produced after running the Stage 1 migration tool will cause problems when running the migration tool that produces EGL. Those situations, which are very rare and unlikely, are discussed here.

- Data item part
  - The map range edit (minimum and maximum values) can cause an exception in Stage 2 due to invalid External Source Format. The problem occurs if External Source Format is imported into VisualAge Generator and the data item is never modified. When the External Source Format is exported in Stage 1 of the migration, the map range edit is in an invalid format. Modify the data item in VisualAge Generator and save the item. For example, add and remove a blank from the item description. Alternatively, **before you run Stage 1 of migration,** install the fix for VisualAge Generator APAR PQ75621 or APAR PQ79914.
  - For VisualAge Generator on Java, data items with similar part names can cause unpredictable results when the # symbol is used in one of the names. For example, DATAITEM#A, DATAITEM@A, and DATAITEM$A can result in the wrong External Source Format code being stored in the migration database for DATAITEM#A. If you have used the # symbol in data item names, review the resulting EGL source code to ensure that the correct information was migrated. Alternatively, **before you run Stage 1 on Java,** install the fix for VisualAge Generator APAR PQ85794.
- Record part
  - For SQL records, the default key item can cause a problem where there is an invalid (or unprintable) character in the field. If you receive a message during Stage 2 indicating "NoSuchElementException", modify the record in VisualAge Generator, swipe through the default key item field and delete the unprintable character. Save the record and run Stage 1 again. Alternatively, **before you run Stage 1 of migration**, install the fix for VisualAge Generator APAR PQ89390.

Appendix G. Migration Database

Creating the DB2 migration database

Except where noted, the following instructions apply regardless of whether you are migrating from Java or Smalltalk. Even if you are migrating from Smalltalk, you must set the JDBC driver level on the machine where you plan to run Stage 2 and 3 of migration.

Setting the JDBC level for DB2 7.2

The migration tool requires that db2java.zip be at the JDBC 2.0 level. DB2 7.2 ships with two db2java.zip files -- one at the JDBC 1.1 level and one at the JDBC 2.0 level. To configure DB2 7.2 for JDBC 2.0, do the following:

1. Stop all DB2 processes.
   a. Navigate to the **Control Panel** and then select **Administrative Tools -> Services**.
   b. You might have to stop the *DB2 - DB2* process last if it does not stop on your first attempt.
2. Open a DOS command prompt window and navigate to directory that contains the *usejdbc2.bat* file. If you used the default install directory when you installed DB2 7.2, the file should be in the \SQLLIB\java12 directory.
3. Run the *usejdbc2.bat* file.
4. Start everything you stopped in the first step.

Setting the JDBC level for DB2 8.1 or higher

If you have DB2 8.1 or higher installed, the db2java.zip file is already at the correct level.

Using DB2 on Windows XP

The migration tool requires the following:

- The user ID that is used to access the migration database must not contain any blanks.
- The Windows user ID needs to have administrator authority, not limited authority, for the migration sets to be visible in the migration tool wizards in Stages 2 and 3.

Creating the migration database

To create the migration database, do the following:

1. Make sure that DB2 and any other applications that use it are shut down. For example, shut down VisualAge Generator and the Rational Developer product.
2. Open a DB2 Command Window.
   - If you are migrating from Java, navigate to the *VisualAgeJava-installation-directory*\ide\vgmigration directory.
   - If you are migrating from Smalltalk, navigate to the *VisualAge-Smalltalk-installation-directory*.
3. Run the file named *SetupDatabase.bat*. This runs a file in the same directory called createdatabase.sql and saves the output to a file called createdatabase.out in the same directory. This creates a DB2 database called VGMIG, connects to

the database, and configures the database parameters. It might take up to a minute to create the database. Be sure to wait until all the commands finish executing.

Note:

- The first command that appears in the console might result in an error message. You can ignore this message. It simply means that the VGMIG database did not already exist.
- If you want to create a database with a name other than VGMIG, you must change all occurrences of VGMIG in createdatabase.sql to your desired database name. You must also remember to change VGMIG in your Stage 1 – 3 migration tool preferences.
- By default the VGMIG database is not password protected. If you need password protection, you must change the database to be password protected.

4. Run the file named *SetupTables.bat*. This runs a file in the same directory called createtables.sql and saves the output to a file called createtables.out in the same directory. This creates all the tables and views that the migration tool needs in the migration database. The tables are crated with a high-level qualifier (a schema) called MIGSCHEMA. It might take up to a minute to create the database. Be sure to wait until all the commands finish executing.

   Note:

   - The first commands that appear in the console might result in error messages. You can ignore these messages. They simply mean that the tables and views did not already exist.
   - If you want to create a schema with a name other than MIGSCHEMA, you must change all occurrences of MIGSCHEMA in createtables.sql to your desired schema name. You must all remember to change MIGSCHEMA in your Stage 1 – 3 migration preferences.
   - If you ever need to completely clean out the migration database, you can rerun the SetupTables.bat file from a DB2 Command Window.

5. Close the DB2 Command Window.

At this point the migration database, schema, tables, and views have been created. You are now ready to create a preferences file for the Stage 1 migration tool to use. If you are migrating from Java, see "Setting Stage 1 preferences" on page 90. If you are migrating from Smalltalk, see "Setting Stage 1 preferences" on page 108.

Resetting the migration database

If you need to reset the migration database (for example, due to changing your renaming rules), use one of the following techniques:

- Use the tool that deletes and recreates all the tables in the migration database.

  Use this tool in the following situations:

  – If you need to delete all your migration plans.
  – If you have migrated multiple versions of a Java project.
  – If you have migrated multiple versions of a Smalltalk configuration map.

  To run the tool that deletes and recreates all the tables, do the following:

  1. From a DB2 Command Window, navigate to the directory where **SetupTables.bat** is located.
     – For Java, this is your *VisualAge-for-Java-install-directory*\ide\vgmigration.

– For Smalltalk, this is your *VisualAge-Smalltalk-install-directory*.

2. Run **SetupTables.bat**.

- Use the tool that deletes a specified migration set. Use this tool if you need to delete only a few migration sets. To run the tool that deletes a specified migration set, do the following:
  1. Determine the migration set ID that you need to delete from the migration database as follows:
     a. Using the DB2 Control Center or an SQL query, look at the CONFIGPLAN table.
     b. Find the CONFIGPLANNAME that you want to delete.
     c. The migration set ID you need to specify is the value in the corresponding CONFIGPLANID column.
  2. From a DB2 Command Window, navigate to the directory where **deletemigsets.bat** is located.
     – For Java, this is your *VisualAge-for-Java-install-directory*\ide\vgmigration.
     – For Smalltalk, this is your *VisualAge-Smalltalk-install-directory*.
  3. Run the **deletemigsets.bat** file, using one of the following formats:
     – If you want to delete just one migration set, use the following format:

       ```
       deletemigsets n
       ```

       where *n* is the migration set ID that you want to delete.
     – If you want to delete several migration sets, use the following format:

       ```
       deletemigsets "n1,n2"
       ```

       where *n1* and *n2* are the migration set IDs that you want to delete.

Cataloging a remote database using DB2

You need the following information to catalog a remote database on DB2:

- Hostname or IP address of the remote machine where the database resides
- Port number and protocol on the client (for example: 60000/tcp)
- Node name (alias that describes the remote machine (for example: db2node)
- Database name
- Database alias (optional)

To establish a TCP/IP connection to a remote database using DB2, do the following:

1. Bring up a Command Prompt window on Windows or a Command Terminal on Linux.
2. To catalog the node, enter the following command all on one line:

   ```
   db2 catalog tcpip node nodeName remote [ hostName | ipAddress ]
       server [ svcename | portNumber ]
   ```

   You can enter either the hostName or the ipAddress. For example, to catalog a remote server on node db2node with the IP address 9.10.11.123 using port number 60000, enter the following command:

   ```
   db2 catalog tcpip node db2node remote 9.10.11.123 server 60000
   ```

3. To catalog the database, enter the following command all on one line:

   ```
   db2 catalog database databaseName
     [ as databaseAlias ] at node nodeName
   ```

The "as *databaseAlias*" is optional. If you do not specify *databaseAlias*, the alias will be the same as the database name. The *nodeName* must be the same *nodeName* you used in step 2.

For example, to catalog a remote database called SAMPLE so that it has the alias sam1 on node db2node, enter the following command:

```
db2 catalog database sample as sam1 at node db2node
```

4. To test the connection to the database, enter the following command all on one line:

   ```
   db2 connect to databaseAlias use userName using password
   ```

   If you did not specify an alias (*databaseAlias*) in step 3, use the database name. For example, to connect to database SAMPLE with the alias sam1 for user db2user who has a password db2password, enter the following command:

   ```
   db2 connect to sam1 user db2user using db2password
   ```

   If you did not specify sam1 as the database alias in step 2, then enter the following command:

   ```
   db2 connect to SAMPLE user db2user using db2password
   ```

5. You should see the Database Connect Information.

For additional assistance, go to the following Web site:

```
https://aurora.vcu.edu/db2help/db2i4/frame3.htm#idx
```

Uncataloging a remote database using DB2

You need the following information to uncatalog a remote database on DB2:

- Database alias or the database name if no alias was specified when you cataloged the database

To uncatalog a remote database using DB2, do the following:

1. Bring up a Command Prompt window on Windows or a Command Terminal on Linux.
2. To uncatalog the database, enter the following command, where *databaseAlias* is the database alias:

   ```
   db2 uncatalog database databaseAlias
   ```

   For example, to uncatalog database SAMPLE (which was given the alias sam1), enter the following command:

   ```
   db2 uncatalog database sam1
   ```

   If you did not specify a database alias when you cataloged the database, use the name of the database. For example, if you did not specify sam1 as the database alias, enter the following command:

   ```
   db2 uncatalog database SAMPLE
   ```

For additional assistance, go to the following Web site:

```
https://aurora.vcu.edu/db2help/db2i4/frame3.htm#idx
```

Useful Queries

If you modify the sample Stage 1 migration tool or develop your own Stage 1 migration tool, the following SQL queries might be useful in verifying your changes.

Note:

- These examples can run from a DB2 Command Window.

- These examples assume that you use the default migration database name (VGMIG) and the default schema (MIGSCHEMA).
- Unless noted otherwise, the entire DB2 command must be entered on one line. The commands shown later in this document might be on several lines due to space limitations.
- These examples require that you connect to the database first. To connect to the database, run the following:

  ```
  db2 connect to VGMIG
  ```

To determine if a VAGen part has been migrated:

```
db2 select configplanname, configplanversion, vgpartname, vgparttime, is_migrated
   from migschema.vgpart where vgpartname = 'yourPartName'
```

To verify the first few characters of the External Source Format for all parts in the migration database:

```
db2 select vgpartname, cast(vgesfsource as char(n)) from migschema.vgpart
```

n is a number between 1 and 256 and is the number of characters you want to display.

To determine if any parts in the migration database do not begin with valid External Source Format tags:

```
db2 select vgpartname, cast(vgesfsource as char(n))
    from migschema.vgpart  where vgesfsource not like ':%'
```

n is a number between 1 and 256 and is the number of characters you want to display.

To reset all parts in the migration database if you want to rerun Stage 2 and 3 of migration without rerunning Stage 1:

```
db2 update migschema.vgpart  set is_migrated = 'N', eglsource = NULL
db2 delete from migschema.translation_msgs
```

To backup the migration database:

```
db2 backup database vgmig to x:\mybackups\backupName
```

x:\mybackups\backupName is the drive and directory where you want the backup to be placed. Several subdirectories will be created under *x:\mybackups\backupName*.

To restore the migration database that you previously backed up:

```
db2 restore database vgmig from x:\mybackups\backupName REPLACE EXISTING
```

x:\mybackups\backupName is the drive and directory where you want the backup to be placed. Several subdirectories will be created under *x:\mybackups\backupName*.

Appendix H. Migration tool performance

There are many factors that affect the performance of the migration tool. These factors include the following:

- General performance for Stage 1, 2 and 3
  - Memory.
  - Processor speed.
  - Local or remote DB2 database. For remote databases, the speed of the network connection is critical.
  - Number of Java projects and packages or number of Smalltalk configuration maps and applications.
  - Number of parts and their distribution by part type.
  - Number of lines in function parts.
- Performance for Stage 1
  - Clean Java workspace or Smalltalk image before starting migration.
  - Local or remote Java repository or Smalltalk library. For remote repositories or libraries, the speed of the network connection is critical. Some customers improve the processing speed by copying their repository to the machine where migration is running.
  - Complexity of renaming rules.
  - Whether the migration set already exists in the migration database. If you are recreating a migration set with a different set of renaming rules, it is more efficient to recreate the SQL tables by running setuptables.bat than to have the Stage 1 migration tool clean out the original migration set. Recreating the SQL tables is only practical if there are no other migration sets in the migration database.
- Performance for Stage 2 and 3
  - Setting for the VMArgs option in the .ini file. When the VMArgs option is not set, the refresh and rebuild of the workspace in Stage 3 does not run reliably. You must set this option. For details of how to set this option, see "Start up parameters" on page 127.

Given the number of factors involved, there is no specific formula that can predict migration run time. However, the following sections provide some antedotal guidance on how long the various stages of migration might take:

- Number of projects, packages, parts and programs
- Processor speed
- Number of lines in function parts
- Clean Java workspace for Stage 1

Number of projects, packages, parts, and programs

The next table provides information on how long the various stages of migration might take. These tests were run using Windows 2000 with a 1.0 gigaHertz of memory and a 1.1 gigaHertz processor speed.

Table 134. Effect of migration set size on migration times

Test case	Number of projects	Number of packages	Number of parts	Number of programs	Stage 1 time (in hours)	Stage 2 time (in hours)	Stage 3 time to write files (in hours)	Stage 3 time to refresh or build (in hours)
1	1	98	18,403	1,246	2.0	5.3	3.1	0.6
2	1	10	2,771	147	1.2	1.0	0.4	0.2
3	3	29	2,660	33	0.3	0.2	0.2	0.1
4	3	44	12,048	249	2.5	2.0	3.5	0.5
5	4	92	10,601	25	1.5	0.9	1.0	0.2
6	5	35	8,238	100	0.8	0.9	1.2	0.3
7	474	570	11,280	162	3.0	1.6	1.8	0.9

Here are some general observations based on Table 134:

- Test cases 2 and 3 have similar numbers of parts, but test case 2 takes 4 times as long to run in Stage 1 and 5 times as long to run in Stage 2. Test case 2 has 4.5 times as many programs to analyze for associates during Stage 1 and 2. So the number of programs has an impact on the run times.
- Test cases 5 and 7 have similar numbers of parts, but test case 7 takes twice as long in Stage 1. Test case 7 has many more programs to analyze for associates during Stage 1. In addition, test case 7 has relatively few data items compared to test case 5. In VAGen, data items never have associates so the Stage 1 migration tool does not attempt to determine associates for the data items. Test case 7 also takes nearly twice as long to write the files in Stage 3. There are more generatable parts for which to analyze associates. Because each generatable part is in its own file, there are more files to write. Proportionally fewer parts are placed into the CommonParts.egl and UnusedParts.egl files than in test case 5. The files are smaller and there are more of them.
- Test case 1 has 50% more parts than test case 7, but test case 7 takes 50% longer to refresh and rebuild the workspace. Test case 7 has many more projects and packages to analyze during the build. In addition, in test case 7 there are 162 project cycles detected during the build process, so this contributes to the longer build times.

Based on the information in Table 134, if you have several hundred or even 1000 programs you might want to migrate them as a single migration set even if the programs are split across several subsystems. This assumes that the subsystems all use the same version of your common projects and that the subsystems do not have any duplicate part names.

Processor speed

Table 135 on page 317 shows the impact of changing the processor speed from 1.1 gigaHertz to 1.6 gigaHertz. Both machines had 1.0 gigaHertz of memory. The 1.1 gigaHertz machine used Windows 2000; the 1.6 gigaHertz machine used Windows XP. The Test case numbers are the same as in Table 134. The processor speed has a significant impact, particularly on the Stage 1 and 2 processing time.

Table 135. Effect of processor speed on migration times

	1.1 gigaHertz				1.6 gigaHertz			
Test case	**Stage 1 time (in hours)**	**Stage 2 time (in hours)**	**Stage 3 time to write files (in hours)**	**Stage 3 time to refresh or build files (in hours)**	**Stage 1 time (in hours)**	**Stage 2 time (in hours)**	**Stage 3 time to write files (in hours)**	**Stage 3 time to refresh or build files (in hours)**
2	1.2	1.0	0.4	0.2	0.4	0.3	0.3	0.1
7	3.0	1.6	1.8	0.9	1.6	0.6	0.9	0.9

Based on Table 135, you might want to use a machine with faster processor speed during migration.

Number of lines in function parts

At one point in time, VisualAge Generator had a problem that resulted in a series of blank lines being inserted into functions. In some cases, as many as 32,000 blank lines were inserted. These extraneous blank lines have a severe impact on performance. Table 136 shows the impact of the number of lines in a function on the migration times. The test case contains 2 projects, 17 packages, 919 parts, and 87 programs. There were 497 functions, 8 of which had numerous blank lines (suspected to be in the 30,000 range). All times are in minutes.

Table 136. Effect of in-function lines on migration times

	Before removing blank lines in VAGen				After removing blank lines in VAGen			
Test case	**Stage 1 time**	**Stage 2 time**	**Stage 3 time to write files**	**Stage 3 time to refresh or build files**	**Stage 1 time**	**Stage 2 time**	**Stage 3 time to write files**	**Stage 3 time to refresh or build files**
8	40	25	1	3	12	11	1	3

There is a dramatic difference in the Stage 1 and 2 processing time just from removing the extraneous blank lines before starting migration. If you know of functions that have large numbers of blank lines, you should eliminate them before migration. However, due to the rarity of the problem in VisualAge Generator, it is probably not cost effective to search for these functions prior to migration. If there are more than 3 consecutive blank lines, the migration tool automatically eliminates the additional blank lines during Stage 2 migration. Therefore, there is no change in the processing time for Stage 3. There is improved performance in EGL due to the elimination of these blank lines.

Clean Java workspace for Stage I

The next table shows the impact of having a clean workspace at the start of Stage 1 migration. Test case 3 is the same test case 3 as in Table 134 on page 316. Test case 9 contains 7 versions of a migration set. The first version contains 3 projects, 4 packages, 30 parts and no programs. The last version contains 6 projects, 11 packages, 66 parts, and 7 programs. All times are in minutes.

Table 137. Effect of clean Java workspace on migration times

Test case	Stage 1 Time without Clean Workspace	Stage 1 Time with Clean Workspace
3	17	12
9	11	1

Based on Table 137, you should consider starting with a clean workspace if you are migrating from VisualAge Java. For details on how to start with a clean Java workspace, see “Improving performance” on page 99.

For VisualAge Smalltalk, similar time savings are likely. Therefore, if you are migrating from VisualAge Smalltalk, you should also consider starting with a clean image. For details on how to start with a clean Smalltalk image, see “Improving performance” on page 117.

Notices

Note to U.S. Government Users Restricted Rights - Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
SWS General Legal Counsel
Department TL3 Building 062
P. O. Box 12195
Research Triangle Park, NC 27709-2195

IBM has made reasonable efforts to ensure the accuracy of the information contained in this publication. If a softcopy of this publication is provided to you with the product, you should consider the information contained in the softcopy version the most recent and most accurate. However, this publication is presented "as is" and IBM makes no warranties of any kind with respect to the contents hereof, the products listed herein, or the completeness or accuracy of this publication.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

IBM may change this publication, the product described herein, or both.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries:

- AIX
- CICS
- DB2
- IBM
- IMS
- iSeries
- MVS
- OS/2
- OS/400
- Rational
- VisualAge
- z/OS

Intel is a trademark of Intel Corporation in the United States, other countries, or both.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft®, Windows, and Windows NT® are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX® is a registered trademark of The Open Group in the United States and other countries.

Other company, product or service names, may be trademarks or service marks of others.

Index

F

G

H

I

J

L

M

O

P

R

S

T

U

V

W

Readers' Comments — We'd Like to Hear from You

Rational Application Developer
VisualAge Generator to EGL Migration Guide
Version 6 Release 0

Publication No. SC31-6830-00

We appreciate your comments about this publication. Feel free to comment on specific errors or omissions, accuracy, organisation, subject matter, or completeness of this book. The comments you send should pertain to only the information in this manual and the way in which the information is presented.

For technical questions and information about products and prices, please contact your IBM branch office, your IBM business partner, or your authorized remarketer.

For general questions, please call "Hello IBM" (phone number 01803/313233).

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Comments:

Thank you for your support.

To submit your comments:

- Send your comments to the address on the reverse side of this form.
- Send a fax to the following number: 1-800-227-5088(US and Canada)
- Send your comments via email to: kfrye@us.ibm.com

If you would like a response from IBM, please fill in the following information:

Name	Address
Company or Organization	
Phone No.	E-mail address

Readers' Comments — We'd Like to Hear from You
SC31-6830-00

IBM

Cut or Fold
Along Line

Fold and Tape | **Please do not staple** | Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
Information Development
Department G7IA / Bldg. 503
P.O. Box 12195
Research Triangle Park, NC
27709-2195

Fold and Tape | **Please do not staple** | Fold and Tape

SC31-6830-00

Cut or Fold
Along Line

Program Number: 5724-J19

Printed in USA

SC31-6830-00