

Rational Business Developer



VisualAge Generator to EGL Migration Guide

Version 8 Release 0

Rational Business Developer



VisualAge Generator to EGL Migration Guide

Version 8 Release 0

Note

Before using this document, read the general information under “Notices” on page 477.

Eighth Edition (January 2011)

© Copyright IBM Corporation 2004, 2011.

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

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. Use Adobe Acrobat Reader 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/software/rational/cafe/community/egl/vagen?view=documents>

Contents

Preface iii

Who should read this book iii

Related information. iii

Part 1. Migration overview. 1

Chapter 1. Migration overview 3

Terminology used in this book 3

What's new in EGL that requires migration? 4

Planning your migration 4

Determining whether you can migrate to EGL 9

VisualAge Generator features not available in

EGL 11

Terminology differences 12

References. 16

Chapter 2. Migration tool philosophy 19

Overview of the VisualAge Generator to EGL

migration tools 20

Migration tool terminology 20

Stage 1 details 21

Stage 2 details 24

Stage 3 details 25

Overview of single file migration 27

Migration challenges 29

Precise EGL syntax 29

When and how part names are resolved. 31

Common code scenarios 31

Techniques used by the VisualAge Generator to EGL

migration tool 35

Overview of techniques 35

Editor and build descriptor preferences 35

Program properties. 36

EGL build path and import statements 38

containerContextDependent Property. 40

EGL part name restrictions 41

Placing parts in EGL files. 42

Migrating with a program 45

Migrating with associated parts 46

Migrating without associated parts 46

Controlling the order for processing migration

sets 47

Overwriting and merging files 48

General rules. 50

Determining how to organize your EGL source code 53

Differences in product capabilities for organizing

your code 54

Organization capabilities provided by the

migration tool 56

Limitations and tradeoffs of EGL source code

organization techniques 57

What's new for the VAGen migration tool since EGL

5.1.2? 59

What's new for the VAGen migration tool since EGL

6.0 iFix? 60

What's new for the VAGen migration tool since EGL

6.0.0.1? 60

What's new for the VAGen migration tool since EGL

6.0.1? 61

What's new for the VAGen migration tool since EGL

6.0.1.1? 61

What's new for the VAGen migration tool since EGL

6.0.1.1 ifix003? 62

What's new for the VAGen migration tool since EGL

7.1? 64

Known restrictions for the migration tools 64

Stage 1 64

Stages 2 and 3 64

Syntax migration 64

Chapter 3. Handling ambiguous

situations 65

Handling ambiguous situations for data items. . . . 65

PACK data items with even length 65

Shared edits and messages 67

Map edit routine for shared data items 68

Fill characters for shared data items 70

Handling ambiguous situations for records. 70

Redefined records 70

Level 77 items in records 71

Alternate specification records 72

Different definitions with the same record name 74

Reserved words and UI record names 75

Handling ambiguous situations for tables 76

Reserved words and table names 76

Handling ambiguous situations for map groups and

maps 77

Reserved words and FormGroup names. 77

Map group and FormGroup requirements 77

Floating areas and starting positions 78

Map names and help map names 79

Numeric variable fields 81

Map variable fields and edit routines. 82

Map fields and the numeric hardware attribute 83

Map arrays and attributes 84

Unnamed map variable fields 85

Unprotected map constants 86

Fields at row=0, column=0 86

Handling ambiguous situations for programs 87

Program names and reserved words 87

Implicit data items in programs 88

Associated program parts 89

Program with EZEDLPCB in called parameter list 91

Intermediate variables required for migration . . 92

Handling ambiguous situations for functions,

including I/O statements. 94

DISPLAY I/O option for maps 94

I/O error routine 95

SQL I/O statements	96
SQL I/O and missing required SQL clauses.	98
SQL I/O and Execution time statement build	102
SQL I/O and !itemColumnName	103
SQL I/O with multiple UPDATE or SETUPD functions	104
DL/I I/O and comparison value items	105
Handling ambiguous situations for other statements	106
Implicit data items in statements	106
Level 77 items in statements	106
Table references in statements	107
MOVEA with a single row table as the source	107
Assignment statements	108
FIND statement	109
RETR statement	110
SET map PAGE statement	110
SET mapItem attributes	111
Checking for IN literal or scalar	112
Checking SQL and map items for NULL	113
I/O error values UNQ and DUP	114
I/O error value LOK	116
Handling ambiguous situations for EZE words	117
EZELTERM	117
EZESYS	118
EZEWAIT	120

Part 2. Migrating from VisualAge Generator 4.5 on Java to EGL . . . 121

Chapter 4. Stage 1 — Extracting from Java 123

Installing the Stage 1 migration tool on VisualAge for Java	123
Adding the migration feature	124
Creating the migration database	124
Setting Stage 1 preferences	124
Build Plans page	125
Mapping page	128
Renaming page.	129
Execution page.	130
Sample MigPreferences.xml file	133
Before you run the Stage 1 tool — hints and tips	135
Customizing the Stage 1 migration tool.	135
Specifying your character set information	136
Improving performance	137
Saving your workspace	138
Running the Stage 1 tool	138
Migration plans and high-level PLP projects	139
Creating a high-level PLP project	140
Creating a migration plan file manually	141

Part 3. Migrating from VisualAge Generator 4.5 on Smalltalk to EGL. 145

Chapter 5. Stage 1 — Extracting from Smalltalk 147

Installing the Stage 1 migration tool on VisualAge Smalltalk	147
Loading the migration feature.	148
Creating the migration database	148
Setting Stage 1 preferences	148
Build Plans page	149
Mapping page	151
Renaming page.	154
Execution page.	155
Sample MigPreferences.xml file	157
Deriving file names from your preferences	159
Before you run the Stage 1 tool — hints and tips	159
Customizing the Stage 1 migration tool.	159
Specifying your character set information	161
Improving performance	161
Saving your image	162
Running the Stage 1 migration tool	162
Repairing a migration database created using a previous version of the Stage 1 tool	165
Migration plans and high-level configuration maps	165
Creating a high-level configuration map	166
Creating a migration plan file manually	167

Part 4. Stages 2 and 3 — common migration steps 169

Chapter 6. Stage 2—Conversion to EGL syntax 171

Setting DB2 performance information	171
Setting your workbench preferences.	171
Start up parameters	171
Required EGL preferences	172
Suggested preferences	173
VAGen Migration preferences	174
Other suggested settings	179
Setting up the Stage 2 VAGen migration file	180
Running Stage 2	184
Running Stage 2 from the user interface	184
Running Stage 2 in batch mode	185

Chapter 7. Stage 3 — Import 189

Running the Stage 3 tool	189
Running Stage 3 in batch mode	192
Using the migration sets written to temporary directories	192

Chapter 8. Running migration in single file mode 195

Running single file migration using the user interface	195
Running single file migration using batch mode	197

Part 5. Completing the migration 201

Chapter 9. Completing your migration 203

Setting the Build Order preference	203
Exporting your preferences.	204
Saving a baseline for EGL projects and packages	204

Preliminary tasks for completing single file migration.	205
Common tasks for both Stage 1 — 3 and single file migration.	205
Reviewing your EGL source code.	205
Reviewing your EGL build descriptor parts	206
Reviewing your EGL linkage options parts	210
Reviewing your EGL resource associations parts	211
Establishing a bind control part to use as a template	212
Establishing a program-specific bind control part	214
Reviewing link edit commands	214
Reviewing your VGWebTransactions	215
Preparing for debugging	216
Installing the EGL server product for zSeries	217
Installing the EGL server product for VSE.	219
Converting VAGen preparation templates and procedures to EGL build scripts	219
Converting VAGen runtime templates	220
Converting the VAGen reserved words file	221
Generating and testing with COBOL generation	222
Generating and testing with Java generation	223
Reviewing your standards	224
Planning for dual maintenance of your source code	224
Eliminating the use of VisualAge Generator compatibility mode	225

Part 6. Language and runtime differences. 229

Chapter 10. Language and runtime differences 231

Language differences	231
Runtime differences	231
General differences	231
Differences in SQL support	233
Differences in DL/I support	235
Differences in debug	237
Differences in generated COBOL	238
Differences in generated Java	239
Differences between host and workstation environments	239
Differences between distributed CICS and native workstation environments	240
Differences between generated C++ and generated Java	243

Part 7. Appendixes 247

Appendix A. Reserved words 249

VisualAge Generator migration tool extended reserved words.	249
EGL reserved words	249
EGL enumeration words.	250
SQL reserved words	253
SQL reserved words requiring special treatment	254
Java reserved words	255

Appendix B. Relationship of VisualAge Generator and EGL language elements 257

General syntax conventions	258
Data item.	259
Record	265
Tables	281
Map groups	283
Maps	286
Programs.	300
Functions.	307
Statements	325
EZE words	339
Program flow EZE words	340
SQL EZE words	341
DL/I EZE words	341
Date and time EZE words	342
Other data EZE words	343
General function EZE words	345
String EZE words	346
Math EZE words	347
User interface EZE words	348
Object scripting EZE words.	349
Service routines	349
PSBs	350
Control parts	352
Generation options part	354
Conversion table names used in generation option parts	370
Conversion table names used in linkage table and resource associations parts	371
Linkage table parts	372
resource associations part	379
Link edit part	383
Bind control part	383
Symbolic parameters	384
Other generation information	386
Preparation templates and procedures	386
Runtime templates	388
Other runtime information	390
Runtime environment variables	390
vgj.properties	392

Appendix C. Messages from the migration tools. 395

Messages from the VisualAge Generator to EGL migration tool—Stage 1	395
Stage 1 common messages	395
Stage 1 on VisualAge for Java	399
Stage 1 on VisualAge Smalltalk	403
Messages from the VisualAge Generator to EGL migration tool— Stage 2.	405
Messages from the VisualAge Generator to EGL migration tool—Stage 3	425

Appendix D. Messages in the Problems view 427

Appendix E. IWN.xxx messages in the Problems view 435

IWN.VAL messages for the .egl files	435
IWN.VAL messages for the .eglbld file	448
Java messages for JSPs	450
Reference information for messages - name resolution and qualification rules	450
VisualAge Generator name resolution and qualification rules	450
EGL name resolution and qualification rules	452
Validation messages due to differences in name resolution and qualification rules	454

Appendix F. APARs required for VisualAge Generator 457

Appendix G. Migration database . . . 459

Creating the DB2 migration database	459
Using DB2 on Windows XP	459
DB2 authority requirements	459
Creating the migration database	459
Resetting the migration database for Stage 1	460
Cataloging a remote database using DB2	461
Uncataloging a remote database using DB2	462
Useful queries	462
Determining the number of parts in the migration database	463
Determining the number of parts migrated during Stage 2	464
Reviewing the EGL file names.	464

Queries to assist with specific error messages	465
Resetting the migration database for Stage 2	465
Backing up and restoring the migration database	466

Appendix H. Migration tool performance 467

Number of projects, packages, parts, and programs	467
Number of migration sets and other migration options	469
Processor speed	470
Number of lines in function parts	470
Clean Java workspace for Stage 1.	471
Disk space requirements.	471

Appendix I. VisualAge Generator and EGL interoperability 473

VisualAge Generator and EGL interoperability on z/OS CICS	473
VisualAge Generator and EGL interoperability on iSeries	473
VisualAge Generator and EGL interoperability for Web Transactions	475
Cross System Product interoperability	476

Notices 477

Trademarks	479
----------------------	-----

Index 481

Part 1. Migration overview

Chapter 1. Migration overview

Products with the Enterprise Generation Language (EGL) component are the successors to VisualAge[®] Generator. To move to EGL, you need to migrate your VisualAge Generator (VAGen) source code.

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.

Terminology used in this book

EGL is available with IBM[®] Rational[®] Business Developer, which can also be installed with other Eclipse-based products. This book uses the following terminology:

developer product

IBM Rational Business Developer, used as a stand alone product or installed with other products such as the following products:

- IBM Rational Application Developer
- IBM Rational Developer for System i[®]
- IBM Rational Developer for System z[®]

EGL development environment

The Workbench and other windows that you see after starting IBM Rational Business Developer.

EGL COBOL generator

Any of the products or features that provide EGL COBOL generation support:

- For System i, EGL COBOL generation is included with IBM Rational Business Developer.
- For System z, EGL COBOL generation is included with the Generation for System z feature of IBM Rational Business Developer. You must install this feature, but it is enabled automatically when you apply the license for IBM Rational Business Developer.
- For VSE, EGL COBOL generation is included with the Generation for VSE feature of IBM Rational Business Developer. You must install this feature, and then must enable the feature by purchasing and applying the license for IBM Rational Business Developer Extension for VSE.

EGL build server

Any of the products or features that provide the EGL build server support for generated COBOL programs:

- For System i, the build server is included with IBM Rational Business Developer, but must be uploaded and installed on System i.
- For System z, the build server is provided by IBM Rational COBOL Runtime for zSeries[®].
- For VSE, the equivalent of the build server is provided by preparation JCL templates included with the Generation for VSE feature and preparation JCL procedures included with IBM Rational COBOL Runtime for z/VSE[®].

EGL runtime server

Any of the products or features that provide the EGL runtime support for generated COBOL programs:

- For System i, the runtime server is included with IBM Rational Business Developer, but must be uploaded and installed on System i.
- For System z, the runtime server is provided by IBM Rational COBOL Runtime for zSeries.
- For VSE, the runtime server is provided by IBM Rational COBOL Runtime for z/VSE.

What's new in EGL that requires migration?

EGL includes major changes from and enhancements to VisualAge Generator. The following types of changes can affect your migration to EGL:

- Changes to the VAGen language, including many enhancements such as new data types, multidimensional structure field 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 an EGL build server instead of preparation JCL templates for COBOL generation.
- Changes to runtime behavior, including the use of the EGL runtime server.
- Changes to library management, including the ability to choose your own source code repository to interface with EGL.

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 factors:

- 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 EGL 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 5. Contact IBM Support to obtain the fix pack or check the VAGen Web site at:
<http://www.ibm.com/software/awdtools/visgen/support>
 Follow the link in the Download section. Also review Appendix F, “APARs required for VisualAge Generator,” on page 457 for additional VAGen APARs that might be necessary for your specific situation.
- Install or upgrade DB2[®] if it is not already available. You must use DB2 for the migration database. The migration tool requires DB2 Version 8.1 with Fix Pack 15 or DB2 Version 8.2 with Fix Pack 8. These two fix pack levels are equivalent. DB2 Version 9.x is supported for Stage 1 on Smalltalk and Stages 2 and 3. DB2 Version 9.x is not supported for Stage 1 on Java.
- Review the prerequisites for the developer product that you plan to use. In addition, review the prerequisites for your runtime environment, as in the following examples:
 - If you plan to generate COBOL for the z/OS[®] environment, be sure to review the prerequisites for IBM Rational COBOL Runtime for zSeries.
 - If you plan to generate Java for the UNIX System Services (USS) environment and plan to build in that environment, be sure to review the prerequisites for IBM Rational COBOL Runtime for zSeries.
 - If you plan to generate COBOL for the z/VSE environment, be sure to review the prerequisites for IBM Rational COBOL Runtime for z/VSE.
 - If you plan to generate for iSeries[®], be sure to review the prerequisites for the runtime component of your developer product.
 - If you plan to generate Java for a workstation environment, be sure to review the prerequisites for your developer product.
- Review the build descriptor options and symbolic parameters that you might need to accurately replicate the behavior of your programs in EGL. For more information, see “Reviewing COBOL generation build descriptor options” on page 208.
- Make key decisions about the scope of the pilot project, as in the following examples:
 - Determine whether you can freeze your VAGen development and maintenance during the actual migration. This technique enables you to migrate the production level of source code only. If you cannot freeze VAGen development and maintenance, be sure to include the following tasks in your pilot project:
 - Develop and test procedures for migrating your work-in-process source from VisualAge Generator to EGL.
 - Develop and test procedures for dual maintenance of common (shared) parts.
 - Choose a back end source code repository.
- Build a task list, resource assignments, and a schedule for the pilot project.
- Obtain education for the pilot team in the following areas:
 - Developer product environment
 - EGL language

- VisualAge Generator to EGL migration tools
- Your new source code repository
- Run the pilot project plan to perform the following tasks:
 - Install the developer product for the pilot team, and be sure to install it on a machine that has the same regional settings as you used for developing your VAGen programs. The following examples show installations with special requirements:
 - If you developed your VAGen programs on a German machine, you should install your 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 developer product on a Chinese machine using the same code page. This ensures that your DBCS characters are migrated correctly.
 - Determine how to organize your source code in EGL. Map this organization to the equivalent VAGen organization. See “Determining how to organize your EGL source code” on page 53 for considerations.
 - 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 19
 - Part 2, “Migrating from VisualAge Generator 4.5 on Java to EGL,” on page 121
 - Part 3, “Migrating from VisualAge Generator 4.5 on Smalltalk to EGL,” on page 145
 - Part 4, “Stages 2 and 3 — common migration steps,” on page 169
 - Part 5, “Completing the migration,” on page 201
 - Test your source code in the EGL development environment by carrying out the following tasks:
 - Plan and install the connectivity required to use the EGL debug facility. If you use any of the following capabilities when you test your VAGen programs using the Interactive Test Facility (ITF), you need to plan how to achieve comparable EGL debug capabilities:
 - Non-EGL programs that you need to call from ITF.
 - Access to DB2 databases.
 - Access to DL/I databases.
 - Access to VSAM files.
 - Create your EGL build parts for debug. This includes the build descriptor options, linkage options, and resource associations parts that you need for debug.
 - Test your source code using the EGL debug facility. Be sure to test each type of connectivity to your host environments.
 - Create your library management processes by carrying out the following tasks:
 - Select and install a source code repository.
 - Provide access to the source code repository 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. For example, you might need tools to assist with the following processes:

- Checkin and checkout procedures
- Version control procedures
- Retrieve source code from the source code repository and load a workspace or directory structure if you want to use batch generation
- Perform the following tasks in the iSeries COBOL target environment:
 - Follow the directions in the *Rational Business Developer EGL Server Guide for IBM i*.
- Perform the following tasks in the z/OS COBOL target environments:
 - Install and enable TCP/IP. TCP/IP is the only method for transferring output of COBOL generation to the z/OS host.
 - Install prerequisites for the IBM Rational COBOL Runtime for zSeries, including any changes to your COBOL compiler and runtime environment.
 - Install IBM Rational COBOL Runtime for zSeries, including the latest PTFs.
 - Create a new set of libraries to contain the output of COBOL generation and the results from the EGL build server.
 - If you use CICS® or IMS™, create a new region for testing the EGL-generated COBOL. This technique avoids accidentally intermixing your VAGen-generated code with the EGL-generated code and enables you to continue maintaining the VAGen code while you are running the pilot project.
 - Customize the EGL runtime server, including running the customization verification programs for all of your runtime environments.
 - Customize the EGL build server and pseudo-JCL build scripts. See “Converting VAGen preparation templates and procedures to EGL build scripts” on page 219 for details.
- Perform the following tasks in the VSE COBOL target environments:
 - Install and enable FTP. FTP is the only method for transferring output of COBOL generation to the VSE host.
 - Install prerequisites for IBM Rational COBOL Runtime for z/VSE, including any changes to your COBOL compiler and runtime environment.
 - Install IBM Rational COBOL Runtime for z/VSE, including the latest PTFs.
 - Create a new set of libraries to contain the output of COBOL generation and the results from the EGL build server.
 - If you use CICS, create a new region for testing the EGL-generated COBOL. This technique avoids accidentally intermixing your VAGen-generated code with the EGL-generated code and enables you to continue maintaining the VAGen code while you are running the pilot project.
 - Customize the EGL runtime server, including running the customization verification programs for all of your runtime environments.
 - Customize the preparation JCL templates and procedures. For details, refer to the VSE documentation listed in “References” on page 16.
- Perform the following tasks in the 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. Based on your original and new runtime platform, see the appropriate sections in Chapter 10, “Language and runtime differences,” on page 231 for a list of the platform differences. See this same chapter if you are changing from generating C++ to generating Java.
 - Obtain JDBC support from your vendor if you are currently using ODBC support.

- Generate and prepare your programs by carrying out the following tasks:
 - Review and modify your build parts (build descriptor, linkage option, resource association, link edit and bind control parts). Based on the build parts used for your runtime environment, see the appropriate sections in Chapter 9, “Completing your migration,” on page 203 for details of changes that cannot be handled by the VAGen Migration Tool.
 - If you modified the VAGen reserved words, create an EGL reserved words file.
 - 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.
- When you migrate certain types of programs, you must generate the EGL programs or relink the VAGen programs to provide interoperability for the VAGen and EGL programs. For details, see Appendix I, “VisualAge Generator and EGL interoperability,” on page 473.
- Use the following procedures for testing:
 - Test at least a representative sample of your generated programs to ensure you understand any runtime differences between VisualAge Generator and EGL. See “Runtime differences” on page 231 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, DataTables, and programs for each target environment. Also test your procedures for adding common code, forms, DataTables, and programs for each target environment.
 - Run a pilot change cycle using typical changes for several developers to ensure that your planned library management processes are acceptable.
 - Plan and test backup and recovery procedures for your source code repository.
- 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 users will notice after migration.
 - Final library management and change control process.
- Build a plan to complete your migration based on the findings from the pilot project.
- Provide education for the remaining developers on the following topics:
 - Developer product environment
 - EGL language
 - Your source code organization in EGL, including how the code is structured into EGL projects, packages, and files
 - Your new source code repository
 - Your new library management process
 - Your new generation process
 - Mentoring, as needed, during the first few weeks of development

Determining whether you can migrate to EGL

EGL is the strategic component in the Rational Developer products 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. Depending on your target environment and the types of programs you have developed with VisualAge Generator, you might need to consider other alternatives.

The EGL base product supports Java generation for the following VisualAge Generator target environments:

- Unix System Services (VAGen OS/390® target environment)
- iSeries (VAGen OS/400® target environment)
- Windows Native
- Linux on Intel platforms
- AIX® Native
- HP-UX
- Solaris

The EGL base product also supports COBOL generation for the following VisualAge Generator target environment:

- iSeries (VAGen OS/400 target environment)

The Generation for System z feature of IBM Rational Business Developer supports COBOL generation for the following VisualAge Generator target environments:

- IMS BMP
- IMS/VS
- z/OS Batch (VAGen MVS™ Batch target environment)
- z/OS CICS (VAGen MVS CICS target environment)

The Generation for VSE feature of IBM Rational Business Developer supports COBOL generation for the following VisualAge Generator target environments:

- VSE Batch
- VSE CICS

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 information:

- Special considerations for migrating to EGL — File and database access, Table 1 on page 9
- Special considerations for migrating to EGL — User interface, Table 2 on page 10
- “VisualAge Generator features not available in EGL” on page 11

The following tables list special considerations for supported environments.

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

VAGen file and database access	Special consideration
SQL	Supported in EGL.

Table 1. Special considerations for migrating to EGL — File and database access (continued)

VAGen file and database access	Special consideration
Serial, indexed, and relative records	Supported in EGL.
Message queue records	Supported in EGL.
DL/I	Supported in EGL for COBOL generation for the z/OS and VSE environments. Also supported for debug if the database is on IMS/VS. Not supported for debug if the database is on CICS or VSE, but check for any changes in the EGL documentation.
GSAM	Supported in EGL.
IMS Message Queues	Supported in EGL.
Btrieve	Not supported in EGL.
Local VSAM	Supported in the following cases: <ul style="list-style-type: none"> • Java generation for AIX. • COBOL generation. Not supported for debug or for Java generation for other environments.
Remote VSAM	Supported in the following cases: <ul style="list-style-type: none"> • Debug if the remote file is on z/OS. • Java generation for Windows if the remote file is on z/OS. • COBOL generation for CICS on z/OS and VSE.

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	Supported in EGL for both COBOL generation and Java generation depending on the environment.
JSP and Java servlets which use VAGen Java wrappers	<ul style="list-style-type: none"> • You can migrate your JSP and Java servlets to your new 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.
Java GUI applications or applets that do <i>not</i> use VAGen parts on the free form surface, but which use VAGen Java wrappers.	<ul style="list-style-type: none"> • You can migrate your Java applications or applets to your new developer product using the information provided by that 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.

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

VAGen user interface	Special considerations
Smalltalk GUI views or visual parts.	Not supported. The views with VAGen parts must be migrated to Java-based solutions. EGL does 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:

- Support for Java or Smalltalk GUIs.
- Certain runtime environments including:
 - MVS/TSO. You might want to consider z/OS CICS.
 - VM and VM Batch. You might want to consider z/OS CICS and z/OS Batch.
 - OS/2 and CICS for OS/2. You might want to consider using Java generation for Windows.
 - TX Series on distributed platforms (such as CICS for Windows, CICS for AIX, and so on). You might want to consider using Java generation for the equivalent native runtime environment (such as Windows, AIX, and so on).

However, if you plan to use Java generation, also determine if you use CICS specific functions that cannot be converted to native runtime environments. See “Differences between distributed CICS and native workstation environments” on page 240 for details of the differences.

- Specialized editors and lists such as a listing of the program produced during generation.
- Specialized functionality, including the following capabilities:
 - Searching for references in a selected set of parts and limiting the search list to a program and its associated parts.
 - Filtering parts by part type or by subtype. EGL provides a search capability so you might be able to search for a specific part type or subtype.
- Specialized debug support, including the following capabilities:
 - DL/I database I/O if the database is on z/OS CICS or VSE CICS.
 - Indexed on relative files if the file is anywhere other than a remote VSAM file on z/OS.
 - Calls to programs in the IMS environment that require static linkage (such as PL/I) and which perform IMS or DL/I calls. Calls to programs that require static linkage are permitted if there are no IMS or DL/I calls in the called program.
 - Calls to programs in the IMS environment that perform DL/I calls using something other than CBLTDLI or AIBTDLI (such as PLITDLI or ASMTDLI).
- VisualAge Generator Templates.

Terminology differences

VisualAge Generator Developer on Java, VisualAge Generator Developer on Smalltalk, and EGL all use different terminology. The following tables relate the VAGen terminology to the EGL 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 splits 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	Type definition using a DataItem part
Nonshared data item	Nonshared data item	Type definition using a primitive data type
Data item part	Data item part	DataItem part
Record part	Record part	Record part Note: The migration tool converts all VAGen record definitions to EGL structured records to preserve VAGen behavior.
PSB part	PSB part	Record part with the PSBRecord stereotype.
User interface (UI) record	User interface (UI) record	Record part with the VGUIRecord stereotype.
Structure items (structure of fields in a record)	Structure items (structure of fields in a record)	Structure fields
Array (multiply occurring item in record or map)	Array (multiply occurring item in record or map)	Structure field array
Table part	Table part	DataTable part
Map group part	Map group part	FormGroup

Table 4. VAGen parts and concepts terminology differences (continued)

VisualAge Generator on Java	VisualAge Generator on Smalltalk	EGL
Map part: <ul style="list-style-type: none"> • display map • printer map 	Map part: <ul style="list-style-type: none"> • display map • printer map 	Form: <ul style="list-style-type: none"> • TextForm • printForm
Function part	Function part	Standalone function part. Note: The migration tool converts all VAGen function parts to EGL standalone function parts.
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)	<ul style="list-style-type: none"> • Smalltalk view and visual parts are not supported. • Java applications and applets are supported <i>if</i> 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.
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 <ul style="list-style-type: none"> • Shows error messages • Product closes only if you close <i>both</i> the Log and the Workbench • Workspace is <i>always</i> saved when you close the product 	System Transcript <ul style="list-style-type: none"> • Shows error messages • Product closes if you close <i>either</i> the System Transcript or the VisualAge Organizer • Image is <i>optionally</i> saved when you close the product 	Console <ul style="list-style-type: none"> • Shows messages. Problems view <ul style="list-style-type: none"> • Shows messages, especially those related to syntax validation. • Workspace is <i>always</i> saved when you close the product.
Workbench <ul style="list-style-type: none"> • Shows the projects and packages in the workspace. 	VisualAge Organizer <ul style="list-style-type: none"> • Shows the applications in the image. 	EGL and Web perspectives: <ul style="list-style-type: none"> • Navigator and Project Explorer views show the projects, source folders, packages, and files in the workspace. • Error markers are displayed in the Project Explorer view, but not in the Navigator view.
Scrapbook	Workspace	Scrapbook Page editor

Table 5. VAGen with IDE Windows terminology differences (continued)

VisualAge Generator on Java	VisualAge Generator on Smalltalk	EGL
Repository Explorer	Application Editions Browser	If you decide to use a repository, the repository might have a comparable concept.
VAGen Parts Browser <ul style="list-style-type: none"> • Three panes show package, part type, and VAGen parts • Filtering and sorting is included in the browser 	VAGen Parts Browser <ul style="list-style-type: none"> • Three panes show application, part type, and VAGen parts • Filtering and sorting is included in the browser 	EGL and Web Perspectives: <ul style="list-style-type: none"> • Navigator and Project Explorer views show the projects, source folders, packages and files in the workspace. • Outline view shows the parts within a file. • EGL Parts List view provides filtering and sorting.
VAGen options	VAGen preferences	EGL preferences
VAJava options	VASmalltalk preferences	Other 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 or File 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 product that you use. You can choose your own repository management system.
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: <ul style="list-style-type: none"> • Purge / Restore • Compact 	Library management: <ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> • Project owner • Package owner • Class owner 	Ownership: <ul style="list-style-type: none"> • 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: <ol style="list-style-type: none"> 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: <ol style="list-style-type: none"> 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: <ol style="list-style-type: none"> 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.
Package: <ol style="list-style-type: none"> 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: <ol style="list-style-type: none"> 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: <ul style="list-style-type: none"> • If you decide to use a repository, the repository might have a comparable concept.

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

VisualAge Generator on Java	VisualAge Generator on Smalltalk	EGL
<p>Class or Type:</p> <ul style="list-style-type: none"> Versioning the package or project automatically versions the included classes 	<p>Class:</p> <ul style="list-style-type: none"> You must version and release the class before you version the application 	<p>EGL part type:</p> <ul style="list-style-type: none"> No comparable concept in EGL.
<p>VAGen parts:</p> <ul style="list-style-type: none"> There is a date and time stamp for each part Packages containing duplicate part names <i>can</i> be added to the workspace. There is a duplicate parts tool to locate the duplicate parts 	<p>VAGen parts:</p> <ul style="list-style-type: none"> There is a date and time stamp for each part Applications containing duplicate part names <i>cannot</i> be loaded into the image. 	<p>EGL parts:</p> <ul style="list-style-type: none"> 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 the EGL build path for the project, the import statements for the file, 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 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.

References

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

- The Web site and news group for VisualAge Generator. The Web site is as at the following address:
<http://www.ibm.com/software/awdtools/visgen/>
- The Web site and forums for EGL. The Web site is as at the following address:
<http://www.ibm.com/software/rational/cafe/community/egl>
- The Web site and forum for the product that you are using.

The following resources also contain details beyond the scope of this Migration Guide:

- The online help system for EGL.

- One of the EGL server guides, which describes how to set up the build server and build scripts, as well as other reference material for creating your runtime environment. Choose one of the following books based on the COBOL runtime environment you use:
 - *IBM Rational COBOL Runtime Guide for zSeries Version 6.0.1* (SC31-6951-03).
 - *Rational Business Developer EGL Server Guide for IBM i Version 7.5* (SC31-6841).
- For VSE, the following volumes that explain the differences between VSE and z/OS support as well as the preparation process and runtime server support:
 - *Program Directory for Rational COBOL Runtime for z/VSE* (GI10-8803-00)
 - *Rational Business Developer V7.5 Generation for z/VSE feature Reference Manual* (SC19-2539-00)

The following white papers are also available to assist with migration:

- For migration from VisualAge Generator on Java:
 - *How to Consolidate Projects and Packages during Stage 1 of the VisualAge Generator on Java to Enterprise Generation Language Migration Tool.*
 - For information on how to modify the EGL file location algorithm used in Stage 1, see “Customizing the Stage 1 migration tool” on page 135.
- For migration from VisualAge Generator on Smalltalk:
 - *How to Consolidate Projects and Packages during Stage 1 of the VisualAge Generator on Smalltalk to Enterprise Generation Language Migration Tool.*
 - For information on how to modify the EGL file location algorithm used in Stage 1, see “Customizing the Stage 1 migration tool” on page 159.
- For migration from either VisualAge Generator on Java or VisualAge Generator on Smalltalk:
 - *How to Create Records for Implicit Items when Migrating from VisualAge Generator to Enterprise Generation Language.*
 - *Using the Rename User Exit in the VisualAge Generator to Enterprise Generation Language Migration Tool.*
 - *How to Modify the Package Name in the JSP when Migrating Web Transactions from VisualAge Generator to Enterprise Generation Language.*
- For migration from Cross System Product or VisualGen (Version 2.2 and earlier):
 - *Migrating from Cross System Product Version 4.1 to Enterprise Generation Language Version 7.5.1.* While this white paper is specific to Cross System Product Version 4.1, it includes an appendix that explains the differences between migrating from earlier versions of Cross System Product and from VisualGen (Version 2.2 and earlier).

All of the white papers are available in the VAGen Migration hub of the EGL Cafe at the following Web site:

<http://www.ibm.com/software/rational/cafe/community/egl/vagen?view=documents>

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:

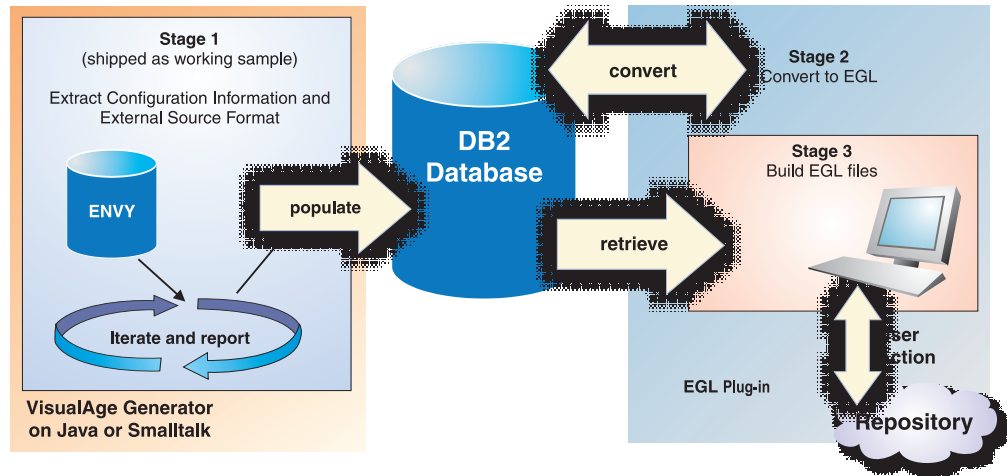
- Migration is from VisualAge Generator 4.5 using External Source Format that is produced by VisualAge Generator 4.5.
- The parts to be migrated are valid VisualAge Generator parts. Programs, tables, and map groups can be validated and/or generated in VisualAge Generator 4.5.

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 20. This is the primary technique for migrating your source code.
- Single File Migration, which is described in “Overview of single file migration” on page 27. 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 also determines the placement of each part in the EGL project, package, and file organization. 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 EGL 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 EGL 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 workspace.

After you have the projects in your workspace, you can then manage the source code using the tools provided by the source code repository that you have decided to use.

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 associates list for the program.

In VisualAge Generator, the following common techniques provide all the parts for generation:

- 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, including 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 use one of the following solutions:

- 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 139 for more details.

Stage 1 details

The Stage 1 tool is shipped as a sample program with the EGL 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 following basic steps are needed to run the Stage 1 tool:

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

Step 1

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

- How to filter Java project names so that only the projects you want to migrate are 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.
- 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.
- 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. This includes the following information:
 - 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.
 - Name to be used as a suffix to the migration set name if maps from the same map group are located in multiple Java projects or Smalltalk configuration maps. Similarly, this may be a name to be used as a suffix to the project name if maps from the same map group are located in multiple Java packages within a single project or in multiple Smalltalk applications within a single configuration map.
- The name of the migration database and the user ID and password that are needed for access to the database.
- 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.
- The names of a log file and a debug file and the level of detail information that you want included in the debug file.

Step 2

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

- **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 five Java projects and a sixth Java project that contains a PLP that specifies the versions of the other five projects. If you request multiple migration plans and three versions, then the Stage 1 tool creates three migration sets, one for each version of the Java Order Entry project that contains the PLP part. Similarly for Smalltalk, if you want to migrate three versions of a configuration map that reflects that code that makes up the Order Entry system, the Stage 1 tool creates three 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).

- **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.

The log file also includes messages for any UI record names that conflict with the EGL reserved word list or that starts with the # or @ symbol. UI records are renamed during Stage 2 migration.

- **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 database with the following information:
 - 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 showing how each migration set will be migrated during Stage 2 and 3.** The report shows the following information:
 - 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 is 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 the section “Placing parts in EGL files” on page 42 for information on how the VAGen parts are assigned to files during Stage 1 – 3 migration.

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.
- You might want to change the parts placement algorithm. See “Determining how to organize your EGL source code” on page 53 for considerations that might cause you to modify the parts placement algorithm for the Stage 1 migration tool.

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 are not 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 are not able to migrate your code.

See Chapter 4, “Stage 1 — Extracting from Java,” on page 123 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 147 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 plug-in `com.ibm.etools.egl.vagenmigration` and runs in the EGL 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 following basic steps are needed to run the Stage 2 tool:

1. You define rules and preferences to tell the Stage 2 tool what you want to migrate, including the following information:
 - 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 record that is named the same as the original working storage record and which contains all the non-level 77 items.
 - 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 three migration sets to migrate three different versions of the Order Entry system, you might want to migrate only 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 log on 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 things:
 - Retrieves parts for one migration set from the database.
 - Converts the External Source Format source code to EGL source code.
 - 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 migration set, the EGL source code is already available and is not converted again.
 - 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.
 - Iterates to process the next selected migration set.

You can run the Stage 2 migration tool in batch mode. See Chapter 6, “Stage 2—Conversion to EGL syntax,” on page 171 for details about running the Stage 2 tool in the EGL development environment. 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 EGL 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 following basic steps are needed to run the Stage 3 tool:

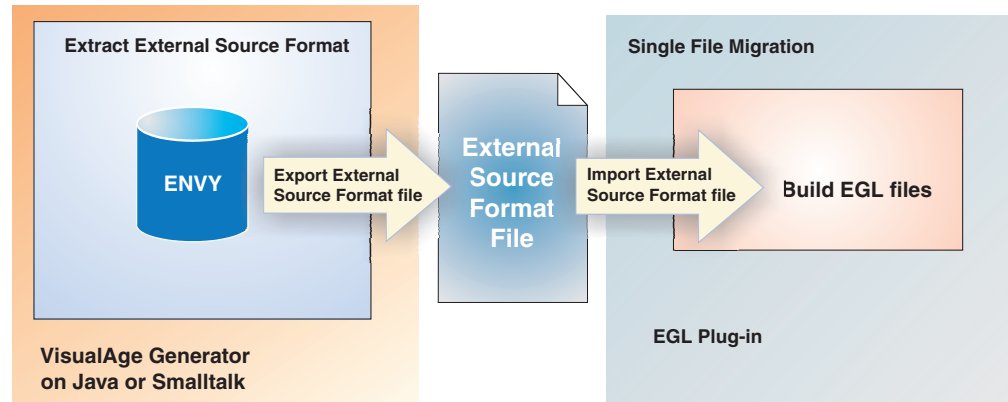
1. You define rules and preferences to tell the Stage 3 tool what you want to migrate, including the following information:
 - Which migration set or sets you want to migrate. For example, if you created three migration sets to migrate three different versions of the Order Entry system, you might have migrated all three 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 things:
 - 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.
 - Creates the EGL project and package structure in your workspace based on the information stored in the migration database during Stage 1.
 - 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 38 for details about the **import** statements.
 - 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 link edit parts.
 - Invokes the tool that optimizes the EGL project build order.
 - Refreshes the workspace so that EGL validation runs.
3. At this point you should follow these steps:
 - a. Optionally, version or commit the EGL projects into your source code repository to establish a baseline that reflects the code exactly as it was migrated.
 - b. 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 or the "to do" list produced in Stage 3.
 - c. Generate (without preparing) all programs and DataTables 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 can help detect generation problems early, without the overhead of the preparation process.
 - d. Version or commit the EGL projects into your source code repository to establish a new baseline that reflects any code changes you made to resolve problems.
 - e. Generate and test the migrated code. Depending on your runtime environment, this step might be required or optional. The advantage of generating and testing is that this step helps ensure that you migrated the correct version of your code, that the code migrated correctly, and that the code runs the same way as in the VAGen runtime environments. If you do not generate and test at this time, you might find differences in behavior the first time you make a change to the program and have difficulties in determining whether the differences are due to the change you just made or due to the migration process.

You can run the Stage 3 migration tool in batch mode. See Chapter 7, "Stage 3 — Import," on page 189 for details about running the Stage 3 tool in the EGL development environment. You cannot modify the Stage 3 migration tool.

Overview of single file migration

When you are first getting accustomed to EGL 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 migration tool provides a mechanism for you to migrate a few programs using *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 EGL development environment to create an EGL project and EGL package. You can then use the Import wizard to import the External Source Format file into EGL source. The single file migration tool does the following things:

- 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 of its associated parts 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 migration tool places the parts into files in the following way:
 - All control parts are placed in a file called *targetFile.eglbld*.
 - Each UI record is placed in a file by itself called *uiRecordName.egl*, where *uiRecordName* is the name of the UI record.
 - 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 migration tool places the parts into files in the following way:
 - 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. The single file migration tool does not 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 four 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 following techniques are available:
 - If you migrate one large External Source Format file containing several programs and their associates, you can only specify one target file name. Assuming you selected the Migration Preference to **Separate parts into EGL files**, the migration tool places the data items, functions, PSBs, and non-UI records into the single target file. Even if the programs share common parts so that the same associate appears multiple times in the file, the migration tool only migrates one definition of the part. Therefore, you do not have any duplicate parts using this technique. However, if the programs have numerous associated parts, the target file can be quite large.
 - If you migrate two External Source Format files to the same EGL package and the two files contain the same part and you specify *different* target file names, the migration tool creates duplicate parts. Consider what happens if Program1 and Program2 share some common parts and you migrate using the following steps:
 1. Migrate a file containing Program1 and its associates to a target file.
 2. Migrate a second file containing Program2 and its associates using a different target file.

In this case, the migration tool places the common parts in both target files. If you place both target files in the same package, there are duplicate part names in the package and EGL cannot resolve the part names. You can avoid this problem by migrating each program and its associated parts to a separate EGL package. This still results in duplicate parts in the workspace, but because they are in different packages, EGL is able to resolve the part references.
 - If you migrate two External Source Format files to the same EGL package and the two files contain the same part and you specify *the same* target file name, you are prompted to specify whether you want to overwrite the existing target file in the workspace.

- If you specify that you do not want to overwrite the existing target file, then any data items, functions, PSBs, and non-VGUI records in the second import are added to the target file. All the common parts in the second import result in duplicate parts within the target file.
- If you specify that you want to overwrite the existing target file, then any data items, functions, PSBs, and non-VGUI records in the second import completely replace the target file. This results in the loss of any parts included in the first import, but not included in the second import.
- If you selected the Migration Preference to **Separate parts into EGL files**, the migration tool overwrites the files created for Programs, FormGroups, and DataTables. If you did not select the preference, then these parts are placed in the target file and added or overwritten based on your response to the overwrite prompt.
- The migration tool always overwrites the files for VGUI records and .eglbl files.

As in the first technique, you can avoid this problem by migrating each program and its associated parts to a separate EGL package. This still results in duplicate parts in the workspace, but because they are in different packages, EGL is 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 External Source Format 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).
- Single file mode migration does not include the following processing that is included by Stages 2 and 3 migration:
 - Nesting forms within FormGroups.
 - Including multiple blank lines between the parts in the output files.

See “Migration challenges” on page 29 and “Techniques used by the VisualAge Generator to EGL migration tool” on page 35 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, as in the following 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 to make you aware that 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 **validatorDataTable** and the **validatorFunction** properties. If the edit table or the edit function is available during migration, the migration tool can determine whether to set the **validatorDataTable** or the **validatorFunction** property. However, if the part specified by the edit routine is not available, the migration tool cannot definitively determine whether to set the EGL **validatorDataTable** or **validatorFunction** 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 **validatorFunction** property. EGL validation displays an error message in the Problems view only if the **validatorFunction** 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 all of 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. For an overview of how the migration tool resolves ambiguous situations, see the following sections:

- “Migrating with a program” on page 45
- “Migrating with associated parts” on page 46
- “Migrating without associated parts” on page 46

See Chapter 3, “Handling ambiguous situations,” on page 65 for a complete list of the situations where the migration tool must do cross-part migration to achieve a correct migration and the techniques the migration tool uses 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.

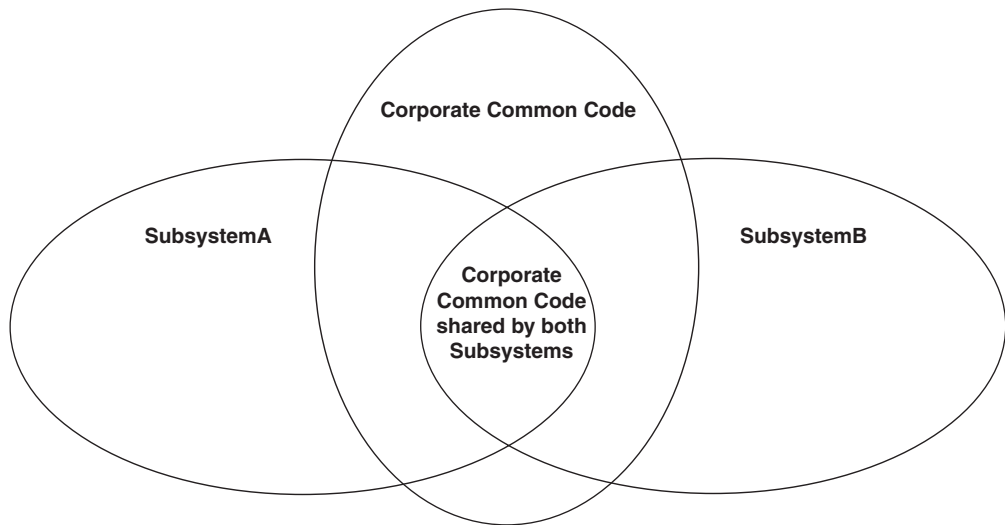
VisualAge Generator searches all parts in the workspace to find a particular part name. If there are duplicate part names in VisualAge for Java, then test and generation are blocked until the duplicate part problem is fixed. VisualAge for Smalltalk does not permit you to load duplicate parts into the 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.

When you migrate using Stage 1 to 3 migration, 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 38
- “containerContextDependent Property” on page 40

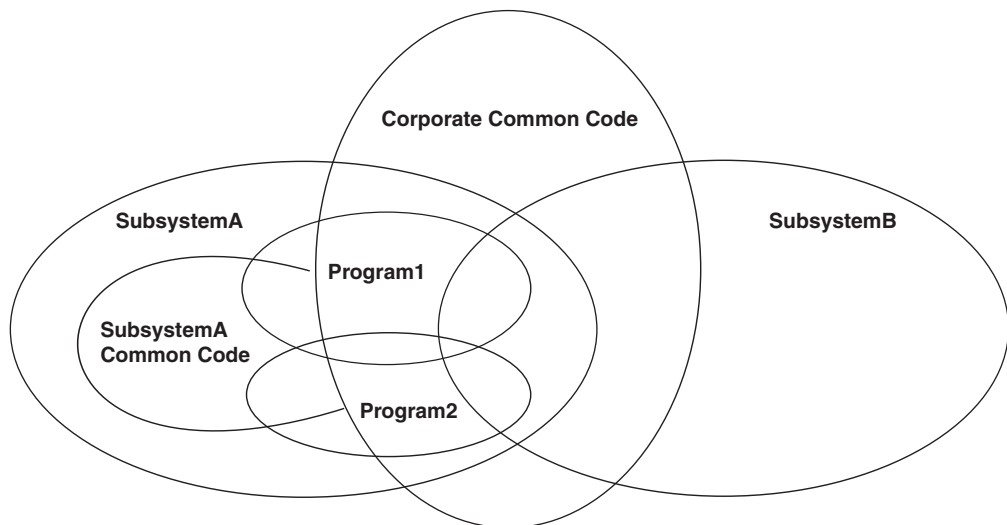
Common code scenarios

Common code 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 examples illustrate this idea:

- 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 in the following example:
 - 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 **converseLib.clearScreen()** for a text form
 - **print** and **converseLib.pageEject()** for a print form

In VisualAge Generator compatibility mode, the **display** statement is tolerated for a print form. However, even in VisualAge Generator compatibility mode, **clearScreen()** only applies to text forms and **pageEject()** only applies to print forms.

- 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 shared data item which is used as a 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 uses 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 DataItem part 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 uses 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 DataItem part definition.

Common code and the migration tool

Common code is used in multiple 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 make manually. For example, you might have an SQL record stored in a VAGen common project and used by functions in many different subsystem projects. When the migration tool converts an SQL function, the migration tool must be able to reference the SQL record that is the I/O option to properly convert the SQL function. Therefore, you must include the VAGen common project whenever you migrate the subsystem projects that use the SQL record.
- 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 need for you to change the EGL build path or add **import** statements.
- 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 results 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

The migration tool uses the following techniques to determine the corresponding EGL syntax and to preserve the VisualAge Generator behavior:

- Editor and build descriptor preferences
- Program properties
- EGL build path and **import** statements
- **containerContextDependent** property
- EGL part name restrictions
- Placing parts in EGL files
- Migrating with a program
- Migrating with associated parts
- Migrating without associated parts
- Controlling the order for processing migration sets
- 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 workspace. The EGL VisualAge Generator compatibility preference provides support for the following VAGen behaviors:

- Cycles in the EGL build path. This means that ProjectA can reference ProjectB in its build path at the same time that ProjectB references ProjectA in its own build path.
- Use of the hyphen (-) and national language characters @ and # in part names. However, these characters are not permitted as the first character of a name even in VisualAge Generator compatibility mode.
- The primitive data types NUMC and PACF.
- Defaulting the subscript to 1 for single dimension structure-field arrays.
- The **deleteAfterUse** property on a **use** declaration for a DataTable, which is the replacement for VAGen Keep After Use.
- A **display printForm** statement is implemented the same way 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.
- If you specify an even-numbered length for an item of primitive type DECIMAL, EGL increments the length by one except when the item is used as an SQL host variable in a WHERE clause or in the EGL **prepare** statement.

The EGL VisualAge Generator compatibility preference provides the following replacements for EZE data words:

- The **converseVar.segmentedMode** system variable, which is the replacement for EZESEGM.
- The **vgLib.getVAGSysType** system function, which provides the old VAGen values for EZESYS.
- The **vgVar.sqlIsolationLevel** system variable, which is the replacement for EZESQISL.

The EGL VisualAge Generator compatibility preference provides the following replacements for EZE function words:

- The **vgLib.connectionService** system function, which is the replacement for EZECONCT.

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

Note: If you think that you might want to eliminate the use of VisualAge Generator compatibility mode in the future, see “Eliminating the use of VisualAge Generator compatibility mode” on page 225 for details *before* you migrate. For example, if you need to eliminate hyphen, @, or # from your part names, you might want to use a Rename User Exit during migration.

There are migration preferences that enable you to minimize the use the migration tool makes of VisualAge Generator compatibility mode. For example, you can specify that the migration tool should not add the **vagCompatibility** = "YES" build descriptor option to every VAGen generation option part. For details, see “VAGen Migration preferences” on page 174.

Regardless of how you set the preferences, the migration tool *always* turns on VisualAge Generator compatibility mode when refreshing the workspace.

Program properties

The migration tool includes the following program properties in 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 fields (VAGen data items) do not need to be qualified. The EGL rules for unqualified fields are similar to the VAGen rules. In most cases, the unqualified fields resolve to the same record, DataTable (VAGen table) or form (VAGen map) as in VisualAge Generator. The migration tool does not add qualifications. However conflicts can arise if a VAGen nonshared item or field on a map has the same name as a program, map, table, or function. For details, see “Reference information for messages - name resolution and qualification rules” on page 450.
- **throwNrfEofExceptions** = YES. The migration tool always includes this program property so that NRF (**noRecordFound**) and EOF (**endOfFile**) are treated as

error conditions. In EGL, NRF and EOF are not normally treated as error conditions. Therefore, **throwNrfEOFExceptions** = yes is required to preserve VAGen behavior.

- **handleHardIOErrors** = NO. The migration tool always includes this program property so that the default value for **vgVar.handleHardIOErrors** is set to 0. **vgVar.handleHardIOErrors** is the replacement for EZEFECE. The normal EGL default value for **vgVar.handleHardIOErrors** is 1. However, the VAGen default value for EZEFECE is 0. Therefore, **handleHardIOErrors** = no is required to preserve VAGen behavior.
- **V60ExceptionCompatibility** = YES. The migration tool always includes this program property so that exceptions do not propagate beyond the function in which they occur. There are additional side effects of setting the **V60ExceptionCompatibility** property to YES, all of which are consistent with VAGen behavior.
- **I4GLItemsNullable** = NO. The migration tool always includes this program property even though NO is the default value. The NO value is required to preserve VAGen behavior and performance.
- **textLiteralDefaultIsString** = NO. The migration tool always includes this program property so that text literals are treated as fixed-length CHAR, DBCHAR, or MBCHAR fields depending on the type of data in the literal. This provides better performance for EGL-generated COBOL programs and also ensures that text literals are passed to non-EGL programs in the same way as in VisualAge Generator.
- **localSQLScope** = YES. The migration tool always includes this program property even though YES is the default value. The naming conventions that the migration tool uses to create the result set ID and the **prepare** statement ID do not guarantee uniqueness across programs. Therefore, setting the **localSQLScope** property to YES is required to preserve VAGen behavior.

The migration tool includes the **@DLI** complex property for every DL/I or IMS program and sets the following property fields:

- **psb** = *psbVariableName*. The migration tool includes this program property to specify the name of the variable that provides the name of the PSBRecord part. The migration tool always uses "psb" as the variable name.
- **callInterface** = **DLICallInterfaceKind.CBLTDLI**. The migration tool includes this program property to ensure that CBLTDLI is used as the call interface. CBLTDLI provides the same call interface that is used by VisualAge Generator. EGL uses AIBTDLI as the default call interface. If you want to use AIBTDLI, you must add PCB Name information to your IMS PSBs and your EGL PSBRecord parts.
- **handleHardDLIErrors** = NO. The migration tool includes this program property so that the default value for **dliVar.handleHardDLIErrors** is set to 0. **dliVar.handleHardDLIErrors** is the replacement for EZEDLERR. The normal EGL default value for **dliVar.handleHardDLIErrors** is 1. However, the VAGen default value for EZEDLERR is 0. Therefore **handleHardDLIErrors** = NO is required to preserve VAGen behavior.
- **psbParm** = *psbData*. If the VAGen program includes EZEDLPSB as a called parameter, the migration tool includes the **psbParm** property to indicate that the entire PSB is passed to the program.
- **pcbParms** = [*list of PCB parameters*]. If the VAGen program includes EZEDLPCB[n], where *n* is a numeric literal, as a called parameter, the migration tool includes the **pcbParms** property to provide the mapping of the input PCB parameters to the PCBs in the PSBRecord part for the program.

Because the edit routines for UI records might have functions that perform I/O or that call another program, the migration tool always includes the following properties to preserve VAGen behavior when creating a VGUI record:

- **throwNrfEofExceptions**
- **handleHardIOErrors**
- **V60ExceptionCompatibility**
- **I4GLItemsNullable**
- **textLiteralDefaultIsString**
- **localSQLScope**

EGL build path and import statements

EGL enables you to have multiple definitions for a part name in the workspace at the same time. The EGL build path for a project limits the other projects that are 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 are 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 DataItem part references if you only have one definition of the DataItem part 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. EGL requires the build path and **import** statements to be specified in the following way:

- The EGL build path property for projects in SubsystemA needs to include the project that provides the definition of RECORDX for SubsystemA.
- 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 are 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 are considered. Even if RECORDX uses DataItem part ITEM1 as a type definition 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 through 3 migration, the migration tool performs the following processing based on the parts in the migration set:

- The migration tool sets the EGL build path for each project based on the parts the project needs to reference in other projects.
- The migration tool includes most **import** 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 associated parts that were determined by VisualAge Generator during Stage 1 of migration.

- The migration tool adds **import** statements for DataItem parts. During Stage 2 migration, the migration tool determines if a DataItem 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 adds **import** statements for UI records. During Stage 2 migration, the migration tool determines if any field in the UI record specifies program link information. If so and the referenced program and first UI record are included in the migration set, then the migration tool updates the migration database to include the program and first UI record as associates of the UI record.
- The migration tool adds **import** statements for functions. During Stage 2 migration, the migration tool determines if any statement in the function explicitly references a table. If so and the referenced table is included in the migration set, then the migration tool updates the migration database to include the table as an associate of the function.
- 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, you can use an entry in a linkage options part to specify the name of the package where the program is located or use the **programPackageName** build descriptor option to force all the generated Java programs to be placed in the same runtime package.
 - 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 probably need to do some reordering of the **nextBuildDescriptor** values (VAGen /OPTIONS). This reordering in turn requires modification of any imports the migration tool might have done.

Note: The Stage 1 migration tool analyzes the parts in the migration set to determine the associates for each part. To ensure that only parts for the migration set are included in the analysis, the Stage 1 migration tool deletes any Java projects from the workspace before loading the migration set specified by a high-level PLP project. Similarly, the Stage 1 migration tool deletes any Smalltalk configuration maps before loading the migration set specified by a high-level configuration map. Because the analysis of associates is limited to the migration set, the migration tool does not set the EGL build path property to specify EGL projects that are not included in the migration set. In addition, the migration tool does not include **import** statements for EGL packages that are not included in the migration set.

Single file mode migration does not include the following processing that is included by Stages 2 and 3 migration:

- Setting the EGL build path because all parts in single file migration are placed in the same project.

- Including **import** statements because all parts in single file migration are placed in the same package.

containerContextDependent Property

Note: This section describes a capability that is only partially implemented in EGL version 7.1. If you specify the **containerContextDependent** property 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 that uses the invoking function. At this time, the **containerContextDependent** property has no effect on name resolution for record or DataItem parts.

The following description reflects what is intended for the final implementation.

As described in “EGL build path and import statements” on page 38, 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 information:

- 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 set the **containerContextDependent** property to 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. Setting the **containerContextDependent** property to 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 the definition of RECORDX for that subsystem.

The **containerContextDependent** property is also supported for record parts. For example, SubsystemA and SubsystemB might both use the same definition of RECORDZ. However, RECORDZ uses a type definition that references the DataItem part called ITEM1. The subsystems have different definitions of ITEM1. In this case, you can specify **containerContextDependent** = yes for RECORDZ so

that EGL validation does 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 your 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 part name restrictions

EGL part and variable names have more restrictions than VAGen part and field names. EGL has the following restrictions:

- EGL has a reserved word list. EGL parts and variables must not be named the same as an EGL reserved word.
- EGL does not permit the use of the # or @ symbol as the first character of an EGL part name, even when the VisualAge Generator compatibility preference is selected.
- If an EGL part or variable name has the same name as an EGL property, annotation, enumeration, or library name, EGL name resolution gives precedence to the part or variable name.

To minimize conflicts, the migration tool creates an extended reserved word list that includes all EGL reserved words, as well as all EGL property, annotation, enumeration, and library names. If a VAGen part or field name is on this extended reserved word list or starts with the # or @ symbol, the migration tool uses the following rules to rename the part or field 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 Preferences that you can specify for Stage 2 or single file mode migration. Similarly, the migration tool renames fields by prefixing the field name with the **Renaming prefix**.

Note: For the purposes of renaming, the migration tools treat a UI record the same as other records. In addition, if a UI record must be renamed, the Stage 3 migration tool changes the name of the .egl file that contains the UI record so that the file name matches the new name for the record. The migration tool sets the **alias** property for the VGUI record to the original VAGen UI record name so that the EGL-generated record name matches the original VAGen UI record.

- The migration tool does not rename control parts, except in the following situations:
 - 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 migration tool extended reserved word list. If you do not rename these parts before you migrate, the Stage 2 migration tool (or single file mode) also issues an error message. EGL validation also displays an error message in the Problems view. You can correct the problem in EGL by renaming the Program, FormGroup, or DataTable and optionally using the EGL `alias` property.

Note: The Stage 2 migration tool issues a warning message for any UI record that is renamed by the tool. Because the Stage 3 migration tool also renames the .egl file, there is no error in the Problems view for UI records.

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 conditions:

- The type of part:
 - Generatable part -- program, table, map group, or UI record
 - Control part -- generation options, resource associations, linkage table, link edit, or bind control
 - Other migratable parts -- data item, map, function, PSB, and records other than UI records.
- 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 conditions:
 - 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 according to the following rules:

- All control parts are placed in a single file called *eglPackageName.egl*bld, 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 according to the following rules:
 - 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.
- Each UI record is placed in a file called *uiRecordName.egl*, where *uiRecordName* is the name of the UI record.
- All the remaining parts are placed according to the following rules:
 - If the part is **used by only one** program in the migration set, the part is placed in a file according to the following rules:
 - 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 or in other generatable parts. 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 in a file in the original package for the part. This file is named *commonParts.egl* by default, but you can change the name through the Common Parts Stage 1 preference.
 - If the part is **used by several** programs or multiple generatable parts in the migration set, then the part is placed in the file called *commonParts.egl* within the 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 in a file according to the following rules:
 - 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 in 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 a file 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 a file within the EGL package that corresponds to the original Smalltalk application that contains the part. This file is named `unusedParts.egl` by default, but you can change the name through the Unused Parts Stage 1 preference.
- The following special considerations apply:
 - The migration tool places any function used as an edit routine in a map, UI record, or data item part in the `commonParts.egl` file. The migration tool also places all the associates of the edit routine function in the `commonParts.egl` file. This technique ensures that edit routine functions used by parts other than a program are visible outside the program file. If you later decide to nest functions within a program, you can move the **end** statement for the program after the last function in the program file without being concerned about the visibility of functions that are used as edit routines.
 - The migration tool places any shared item that is used in a table or UI record in the `commonParts.egl` file. The migration tool never places shared items in the same file with a table or UI record.

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. Alternatively, migrate the most recent version of the migration set through Stage 1 so that any part edition that has not changed since earlier migration set versions is placed into an EGL file based on the current usage of the part. For example:

- A part that was previously used by only one program might now be used by several programs. Migrating the most recent version of the migration set first causes the part to be placed into a common parts EGL file rather than placed with the program that was the original (and sole) user of the part.
- A part that was previously not used by any programs might now be used by one or more programs. Migrating the most recent version of the migration set first causes the part to be placed based on the current usage of the part, rather than placed in the `unusedParts.egl` file.

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 or by the first few characters of the part name.
- 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.
- If you used a strategy of placing one program per Java package and one package per project, you might want to combine packages or projects to reduce the number of EGL projects and packages you need to manage in your source code repository. Similarly, if you used a strategy of placing one program per Smalltalk application and one application per configuration map, you might want to combine applications or configuration maps when creating your EGL projects and packages.

For examples of modifying the Stage 1 tools, see the Stage 1 white papers listed in “References” on page 16. In addition, the Stage 1 tools have some built-in customization capability. For more information on the Stage 1 tool on Java, see “Customizing the Stage 1 migration tool” on page 135. For more information on the Stage 1 tool on Smalltalk, see “Customizing the Stage 1 migration tool” on page 159.

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 and its associated parts to migrate determines the EGL syntax for any ambiguous situation within that program or its associated parts. 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 the parts associated with it.

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 **converseLib.clearScreen()** system library function. If ProgramB migrates first, the migration tool creates the EGL source to use the **print** statement and the **converseLib.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 cannot 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 associated parts 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 **validatorDataTable** property. If there is a function that is named the same as the edit routine, then the migration tool migrates to the **validatorFunction** 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 field 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 migration set.

Migrating without associated parts

Sometimes even when a program is available, not all of its associated parts 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 is 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. In this case, the migration tool makes use of other information. The tool considers the following factors when determining whether to use the **validatorDataTable** or **validatorFunction** 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.

The presence of any of these factors enables the migration tool to make an intelligent choice between setting the **validatorDataTable** or **validatorFunction** property. If there is nothing to make a definitive choice, the migration tool uses the **validatorFunction** 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 **converseLib.clearScreen()** function for a text form and an EGL **converseLib.pageEject()** function for a print form. However, both choices are equally probable. Therefore, the migration tool creates intentionally invalid syntax and converts to **converseLib.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 debug and generation treat RecordA and RecordB as separate data areas. The program does not run the same way that it did 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.

Controlling the order for processing migration sets

The Stage 1 migration tool processes migration sets in the following order:

- If you specify a .pln file or a directory containing .pln files, the Stage 1 migration tool processes the migration sets within a single .pln file in the order in which they are listed within the file. If there are multiple .pln files in a directory, the Stage 1 migration tool processes the files in alphabetical order.
- If you do not specify a .pln file or a directory containing .pln files, the Stage 1 migration tool processes the Java high-level PLP projects that match the specifications in alphabetical order. If multiple versions of the same Java high-level PLP project are requested, the Stage 1 migration tool processes the versions in order based on the date/time stamp. Similarly, the Stage 1 migration tool processes the Smalltalk high-level configuration maps that match the specifications in alphabetical order. If multiple versions of the same Smalltalk high-level configuration map are requested, the Stage 1 migration tool processes the versions in order based on the date/time stamp.
- If you need more control over the order in which migration sets are added to the migration database, run the Stage 1 tool multiple times.

The Stage 2 and 3 migration tools process migration sets in the following order:

- The Stage 2 and 3 migration tools process the migration sets in the same order they are listed in the .vgmig file. By default, if all the migration set names are unique, this is the same order in which the migration sets are listed in the VAGen Migration wizard and is the same order in which the migration sets were added to the migration database in Stage 1.
- If you need to change the order, clear **Migrate now** and save the .vgmig file. Double-click the .vgmig file to change the order and then save the file. Right-click the .vgmig file and then click **Start Migration**. Alternatively, run Stage 2 and 3 multiple times, specifying just one migration set each time, in the order in which you want the migration to occur.

Note: You should not process multiple versions of a migration set using online mode.

Overwriting and merging files

The Stage 2 and 3 migration wizards provide the following related preferences that control processing for multiple versions of the same migration set:

- 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.

- For the purposes of the **Migrate remaining VAGen parts** preference, UI records are treated like other records. This preference does not consider UI records to be generatable parts.
- 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. Clearing the **Migrate remaining VAGen parts** preference can be useful if you are migrating a subsystem project and a common project in a single migration set. In this situation, the migration tool migrates projects in the following way:
 - 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 to clearing **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 the **Migrate remaining VAGen parts** preference, 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 to 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 select **Override existing files**. By selecting both options you ensure that all the parts in the migration set are included in the EGL file.

The **Import into workspace** preference 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 the **Override existing files** preference, 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 the **Override existing files** preference if you decide to change your VAGen Migration Preferences or Database I/O Preferences or to modify your **Rename User Exit** 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. Selecting the **Override existing files** preference 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 select the **Migrate remaining VAGen parts** preference. 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 clear the **Override existing files** preference, 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. Clearing the **Override existing files** preference is useful only if you clear the **Migrate remaining VAGen parts** preference 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 the **Import into workspace** preference when you are migrating multiple versions of a migration set. However, the better technique is to clear **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.

The **Save migrated files to temporary directory** preference 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 EGL development environment. Therefore, due to the large amount of memory resource required, use it only in 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 following list summarizes important rules that are discussed in other sections:

- Cross-part migration is required for a good conversion from VisualAge Generator to EGL. Therefore, you must include common (shared) parts with every migration set. The cross-part migration affects the migration of your parts in the following ways:
 - Stage 1 controls the parts placement. The first migration set in which a part edition occurs for a data item, record, PSB, or function determines whether the part is placed in the same file with a program, in a common parts file, or in an unused parts file based on other parts included in that migration set. The placement for the part edition is used for all subsequent migration sets until you clean out the migration database.
 - Stage 2 controls the conversion of the parts to EGL. The first migration set in which a part edition occurs determines the conversion to EGL source using cross-part migration based on other parts included in that migration set. The conversion for the part edition is used for all subsequent migration sets until you reset the migration information for the part edition.

- Stage 3 always turns on the VisualAge Generator compatibility preference and turns off the Build automatically preference before starting the build that occurs at the end of migration. The migration tool does not restore these preferences to their original values.
- For Stage 3, migration of multiple versions of a migration set or of migration sets that contain different versions of the same project is only supported in batch mode.

The following general rules also apply:

- The migration tool adds a few new item variables to each program. These variables are needed to support the EGL replacement for EZE words or other statements. Adding the variables to the program permits the variables to be used by any function in the program. See “Intermediate variables required for migration” on page 92 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 or incorrect information, as in the following examples:
 - 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. This preserves the unsupported values to facilitate finding places where you might need to update your logic if you change from an unsupported environment such as TSO to a supported EGL environment such as ZOSCICS.
 - The EZESCRPT special function word is converted to EZE_SCRIPT. There is no corresponding EGL replacement. Because EZESCRPT could not be used in an IF, WHILE, or TEST statement, the logic structure of your function is preserved. The migration tool issues an error message. EGL validation does not display an error message 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, you can use an entry in a linkage options part to specify the name of the package where the program is located, or use the **programPackageName** build descriptor option to force all the generated Java programs to be placed in the same runtime package.

- 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 probably need to do some reordering of the **nextBuildDescriptor** values (VAGen /OPTIONS). This reordering in turn requires modification of any imports the migration tool might have done.

Note: The Stage 1 migration tool analyzes the parts in the migration set to determine the associates for each part. To ensure that only parts for the migration set are included in the analysis, the Stage 1 migration tool deletes any Java projects from the workspace before loading the migration set specified by a high-level PLP project. Similarly, the Stage 1 migration tool deletes any Smalltalk configuration maps before loading the migration set specified by a high-level configuration map. Because the analysis of associates is limited to the migration set, the migration tool does not set the EGL build path property to specify EGL projects that are not included in the migration set. In addition, the migration tool does not include **import** statements for EGL packages that are not included in the migration set.

- EGL does not permit implicit items. VisualAge Generator permits implicit items, but their use is not generally considered to be a good practice. 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 uses implicit items. If implicit items are used, VAGen validation provides a message with the correct definition of the item. If you do not create a definition for the implicit item before you migrate, EGL validation displays an error message in the Problems view and you can correct the problem in EGL. See “References” on page 16 for information about a white paper that can help you create the implicit items before you run Stage 1 of migration.
- 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 40 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. EGL validation displays an error message 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 the Tables and Additional Records list for a program, the migration tool does not remove it and does not issue a message. EGL validation displays an error message in the Problems view. Similarly, if the same table is listed twice in the Tables and Additional Records list for a program, the migration tool does not remove the extra table and does not issue a message. EGL validation displays an error message in the Problems view.
 - If a record is not used in a program, but is listed in the Tables and Additional Records list for the program, the migration tool does not remove it and does not issue a message. EGL validation does not display an error message in the Problems view. Similarly, if a table is not used in a program, but is listed in the Tables and Additional Records list for a program, the migration tool does not remove the table and does not issue a message. EGL validation does not display an error message in the Problems view.
 - If a working storage record is listed in the Tables and Additional Records list for a program and the record contains level 77 items only, the migration tool does not remove the record and does not issue a message. EGL validation displays an error message 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 and 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, EGL validation displays an error message in the Problems view indicating that the FormGroup specified in the **use** declarations for the program 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:
 - If a CALL statement specifies an unqualified item as an argument, and there are multiple definitions for this item name within a program, VAGen gives precedence to the level 77 item in the primary working storage record for the program. EGL requires that the item be qualified. The migration tool does not add the qualification for you and does not provide a warning message. EGL validation displays an error message in the Problems view.

Determining how to organize your EGL source code

Before you attempt to organize your source code for EGL, you need to understand the following concepts:

- The differences between the VisualAge Generator and EGL products, particularly in the following areas:
 - The facilities the products provide for organizing code.
 - The way the products determine which parts to consider during development, test, and generation.
 - How the products track the changes you make to a part.
- The capabilities provided by the migration tool to help you achieve the final organization you want in EGL.
- The limitations and tradeoffs of various source code organization techniques in EGL.

Differences in product capabilities for organizing your code

VisualAge Generator and EGL provide different methods for organizing your source code. VAGen on Java uses projects and packages. VAGen on Smalltalk uses configuration maps and applications. EGL uses projects, packages, and files.

VAGen on Java code organization

In VAGen on Java, the source code is organized into Java projects and packages. For example, you might have ProjectA that includes all the packages that contain code unique to SubsystemA, ProjectB that includes all the packages that contain code unique to SubsystemB, and ProjectCommon that includes all the packages that are shared by multiple subsystems.

The Java projects that you add into your workspace determine the source code that is considered when you develop, test, or generate your VAGen programs. You can use a project that contains a Project List Part (PLP) to point to other projects that must be added to your workspace at the same time. For example, in addition to specifying all the packages that contain code unique to SubsystemA, ProjectA can include a PLP part that specifies that ProjectCommon is a VAGen required project. Specifying the VAGen required project ensures that whenever you load ProjectA into your workspace, the correct version of ProjectCommon and all the package versions it contains is also loaded so that you have all the parts needed to develop, test, and generate. You can add projects that contain duplicate part names into your workspace, but you cannot test or generate if there are duplicate parts in your workspace.

When you make a change to a part, VisualAge Generator creates a new edition of the part in your workspace and in the ENVY repository. VisualAge Generator uses a technique called versioning to freeze the code at a known level. The ENVY repository stores all the versions of a Java project or package, but you can only have one version in your workspace at a given time. Tools provide a way of comparing the version in your workspace with previous versions in the ENVY repository to see what has changed at the project, package, or part level. To keep track of changes, you can use different versions of the same Java project for development, each level of test, or production. An alternative technique for tracking changes is to have one project for development, one for each level of test, and one for production.

VAGen on Smalltalk code organization

In VAGen on Smalltalk, the source code is organized into Smalltalk configuration maps and applications. For example, you might have ConfigurationMapA that includes all the applications that contain code unique to SubsystemA,

ConfigurationMapB that includes all the applications that contain code unique to SubsystemB, and ConfigurationMapCommon that includes all the applications that are shared by multiple subsystems.

The Smalltalk configuration maps that you load into your image determine the source code that is considered when you develop, test, or generate your VAGen programs. Configuration maps provide an easy way of specifying which Smalltalk applications to load into your image. For example, in addition to specifying all the Smalltalk applications that contain code unique to SubsystemA, ConfigurationMapA can specify that ConfigurationMapCommon is a required configuration map. Specifying the required configuration map ensures that whenever you load ConfigurationMapA into your image, the correct version of ConfigurationMapCommon and all the application versions it contains is also loaded so that you have all the parts needed to develop, test, and generate. You cannot load two applications or configuration maps into your image if they contain duplicate part names.

When you make a change to a part, VisualAge Generator creates a new edition of the part in your image and in the ENVY manager. VisualAge Generator uses a technique called versioning to freeze the code at a known level. The ENVY manager stores all the versions of a Smalltalk configuration map or application, but you can only have one version loaded into your image at a given time. Tools provide a way of comparing the version in your image with previous versions in the ENVY manager to see what has changed at the configuration map, application, or part level. To keep track of changes, you can use different versions of the same Smalltalk configuration map for development, each level of test, or production. An alternative technique for tracking changes is to have one Smalltalk configuration map for development, one for each level of test, and one for production.

EGL code organization

In EGL, the source code is organized into projects, packages, and files. For example, you might have ProjectA that includes all the packages that contain code unique to SubsystemA, ProjectB that includes all the packages that contain code unique to SubsystemB, and ProjectCommon that includes all the packages that are shared by multiple subsystems. This is similar to VAGen on Java, but differs in two important ways:

- EGL does not have a concept similar to the PLPs in VAGen on Java. Instead, you must determine which projects to load into your workspace so that all the parts necessary to develop, debug, and generate are available.
- The EGL files provide a more detailed organization of your source code. For example within ProjectA, you might have one package for data that is shared by the programs in SubsystemA. This package might include all the data item parts for SubsystemA and the records that are used by multiple programs in SubsystemA. You might organize this package into files in several ways, including any of the following schemes:
 - One file that contains all the data items and records.
 - One file that contains all the data items and another file that contains all the records.
 - One file that contains the data items that start with the letters A through M, another file that contains the data items that start with the letters N through Z, and one file that contains all the records.

EGL requires that Program, DataTable, and FormGroup parts must each be in a unique file, but the file can also contain other parts. For example, you might have a file for ProgramX that contains the ProgramX part as well as functions and records that are unique to ProgramX.

When you create an EGL project, you use the EGL Build Path to specify any other projects to consider when you develop, test, or generate your EGL programs, DataTables, or FormGroups. The EGL Build Path for a project limits which other projects are considered when searching for a part name. EGL also uses **import** statements within each file to determine which packages to include from within the projects listed in the EGL Build Path when searching for a part name. You can have duplicate part names in your workspace, but the part names within the EGL Build Path and the set of **import** statements must be unique.

When you make a change to a part and save the file, EGL stores the file into the file system and replaces the previous file. You use a source code repository to retain multiple versions of the code. The source code repository provides tools such as checkout and checkin, version control, and comparison tools so that you can compare what is in your workspace with other versions of the code in the source code repository. The source code repository also enables developers to share their changes. There are a number of source code repositories that you can use with EGL. Some examples are CVS and IBM Rational ClearCase®. Regardless of the source code repository you select, you can only have one version of a project, package, or file loaded into your workspace at a given time.

Organization capabilities provided by the migration tool

The Stage 1 migration tool determines the placement of each part in the EGL project, package, and file organization. By default, the Stage 1 migration tool for Java preserves your VAGen on Java project and package structure by converting each VAGen on Java project to an EGL project and converting each VAGen on Java package to an EGL package. Similarly, the Stage 1 migration tool for Smalltalk preserves your VAGen on Smalltalk configuration map and application structure by converting each VAGen on Smalltalk configuration map to an EGL project and converting each VAGen on Smalltalk application to an EGL package. The only exception to this default preservation policy occurs if maps and their corresponding map group are in multiple Java projects or packages or multiple Smalltalk configuration maps or applications. In the exception case, the migration tool merges the maps and their map group into a new EGL package or project, depending on the original placement of the maps and map group. See “Placing parts in EGL files” on page 42 for details of the exception case and how the Stage 1 migration tool assigns VAGen parts to files.

There might be situations, such as those described in “Limitations and tradeoffs of EGL source code organization techniques” on page 57, in which you want to organize your EGL projects, packages, and files differently from the default used by the Stage 1 migration tool. For this reason, the Stage 1 migration tool is shipped as a sample program. You can modify the Stage 1 tool to better suit your environment. The following documentation and white papers provide sample modifications to the Stage 1 tool:

- *File Location* sections:
 - For information about the Stage 1 tool on Java, see “Customizing the Stage 1 migration tool” on page 135.
 - For information about the Stage 1 tool on Smalltalk, see “Customizing the Stage 1 migration tool” on page 159.

- *Project Consolidation* documents:
 - *How to Consolidate Projects and Packages during Stage 1 of the VisualAge Generator on Java to Enterprise Generation Language Migration Tool.*
 - *How to Consolidate Projects and Packages during Stage 1 of the VisualAge Generator on Smalltalk to Enterprise Generation Language Migration Tool.*

The two *File Location* sections show examples of modifying the parts placement algorithms for common and unused parts to split the commonParts.egl and unusedParts.egl files into smaller files based on the part type and the first character of the part name. They also include information about how to move all the data item parts to a new project. The two *Project Consolidation* white papers show examples of merging multiple Java projects or Smalltalk configuration maps into a single EGL project and merging multiple Java packages or Smalltalk applications into a single EGL package.

The white papers are available in the VAGen Migration hub of the EGL Cafe at the following Web site:

<http://www.ibm.com/software/rational/cafe/community/egl/vagen?view=documents>

Limitations and tradeoffs of EGL source code organization techniques

EGL works best when you organize your projects along functional lines. This technique can help minimize the number of EGL projects you need in your workspace, thereby improving performance. Your VAGen on Java projects and packages or your VAGen on Smalltalk configuration maps and applications might already be organized along functional lines. In this case, you might be able to use the default Stage 1 migration tool. However, you need to consider other factors that might cause you to alter the default EGL project and package strategy or the default file placement algorithm used by the Stage 1 migration tool.

The main factor to consider is that **the size of EGL projects, packages, and files matters**. The size can affect EGL performance in any of the following areas:

- Time required for opening or saving a file.
- EGL build time that is required to validate the parts. If you turn on the workbench preference to Build Automatically, an EGL build occurs whenever you save a file. If you do not turn on this preference, an EGL build occurs when you request it. As a minimum, an EGL build must occur before you attempt to debug or generate your code.
- EGL generation time that is required to generate the Java or COBOL source code.

The following table shows the limitations and tradeoffs of project, package, and file sizes.

Table 9. Limitations and tradeoffs of project, package, and file sizes

	Limitations	Advantage of Smaller	Advantage of Larger
Project	<ul style="list-style-type: none"> • Maximum of 1500 projects. • Some source code repositories might have a smaller maximum. 	<ul style="list-style-type: none"> • If you only need a subset of the projects, then the workspace is smaller so the EGL build time might be quicker. • Smaller projects load quicker from the source code repository. • Minimizes the chance that two developers need to make changes to the same project at the same time. 	<ul style="list-style-type: none"> • Fewer projects in the EGL Build Path. • Less likelihood of cycles and fewer cycles in the EGL Build Path. • Fewer projects to scroll through. • Fewer projects to load from the source code repository.
Package	<ul style="list-style-type: none"> • Theoretical maximum of 65K parts on a drive formatted as FAT32, assuming that the part names are no longer than 8 characters in length. Longer part names reduce this maximum. 	<ul style="list-style-type: none"> • An import statement of the form: <code>import package.*</code> includes fewer part names so the EGL build time might be quicker. • Minimizes the chance that two developers need to make changes to the same package at the same time. 	<ul style="list-style-type: none"> • Fewer import statements in each file. • Fewer packages to scroll through.
File	<ul style="list-style-type: none"> • Performance to open or save a file degrades for large file sizes. • No maximum size, but practical size is <200K. 	<ul style="list-style-type: none"> • When you save a file, there are fewer parts to check for changes so the EGL build time might be quicker. • With good naming conventions, you can quickly find the file containing a specific part. • Minimizes the chance that two developers need to make changes to the same file at the same time. 	<ul style="list-style-type: none"> • Fewer files to scroll through.

Given the tradeoffs in Table 9, you should avoid creating giant EGL projects, packages, or files. Conversely, you should avoid creating lots of tiny EGL projects,

packages, or files. Both extremes can adversely affect performance. For example, a file that contains 30,000 data items is probably very large and might take several minutes to open or save. Conversely, having 30,000 files, each with just one data item, results in too many files to reasonably scroll through in the various workbench views. A compromise approach is better. For example, one possible compromise if the first character of the data item names is evenly distributed across the alphabet, is to create multiple files, with each file containing all the data items that start with the same character. This technique creates smaller files that are quicker to open, but minimizes the total number of files.

In addition, consider the following factors when structuring your EGL projects, packages, and files:

- Consider your source code repository. The following list includes some of the factors you might need to consider based on the source code registry you select:
 - Whether the source code repository supports checkout and checkin at the project, package, or file level.
 - Whether the source code repository supports version control at the project, package, or file level.
 - How the source code repository deals with the situation in which several developers might need to make changes to the same project, package, or file at the same time and how likely that situation is to occur given the organization of your EGL projects, packages, and files.
- If you use a source code repository, it might have support for the concepts of ownership or access control. If so, you need to consider the following factors:
 - Your ownership strategy should reflect how development and maintenance responsibilities are divided. Organizing your source code along functional lines is useful if one person or group is responsible for developing and maintaining a functional area.
 - Whether there are restrictions that limit access to certain programs or parts. For example, access to a program that writes payroll checks might be limited to just one or two developers. In this case, you might need to put the program in an EGL project by itself so that you can restrict access to that project.

What's new for the VAGen migration tool since EGL 5.1.2?

Note: If you created a migration database using EGL 5.1.2, you must create the database again by following the instructions in “Creating the DB2 migration database” on page 459.

The following functions are new or improved since EGL 5.1.2 General Availability. Depending on your level of maintenance for the WebSphere® Developer 5.1.2 product, you might already have some of the Stage 1 enhancements.

- Stage 1 has been changed in the following ways:
 - Updated DDL for the migration database. You must rerun SetupTables.bat and rerun Stage 1 to place your source code in the migration database.
 - Externalized the EGL reserved word list and included the reserved word list version in the log information.
 - Added new checkStage1.bat file to check the results of Stage 1 for errors.
- Both Stage 2 and 3 and single file mode migration now include the following changes:
 - Support for the new EGL syntax.

- Optionally invoke a user exit to enable you to rename parts during Stage 2 migration or during single file migration. For example, you can write a user exit to change a hyphen (-) to an underscore (_).
- New VAGen Syntax Migration Preferences to enable you to specify the following options for fields in SQL records:
 - The migration tool should omit the **column** property if the SQL column name is the same as the item name.
 - The migration tool should always omit the **isSQLNullable** = YES property.
 - The migration tool should only include the **isReadOnly** = YES property if there is only one SQL table specified for the SQL record and the VAGen Read Only property is set to yes.
- New VAGen Syntax Migration Preference to enable you to specify that the migration tool should change the decimal comma to a decimal point, even if your workstation uses a locale that defaults to the decimal point.
- Stage 2 to 3 migration now also does the following things:
 - Add **import** statements for DataItem parts that specify a **validatorFunction** or **validatorDataTable** property.
 - Sort parts in a file by the EGL part name within the part type.

The following EGL changes provide better support for migrated VAGen programs:

- Numeric variables can now be used in string concatenation. This provides support for the VAGen SQL I/O Execution time statement build option that is migrated to the EGL **prepare** statement.
- The Java runtime environments now permit you to change the EGL product messages similar to the capability provided in VisualAge Generator.

What's new for the VAGen migration tool since EGL 6.0 iFix?

Note: If you created a migration database using EGL 6.0 iFix, you must create the database again by following the instructions in "Creating the DB2 migration database" on page 459.

The following functions are new or improved since EGL 6.0 iFix:

- Both Stage 2 and 3 and single file mode migration now include the following changes:
 - Better support for the user exit that enables you to rename parts during Stage 2 migration or during single file migration.
 - New VAGen Migration Preferences to enable you to minimize the use of VisualAge Generator compatibility mode.

What's new for the VAGen migration tool since EGL 6.0.0.1?

Note: If you created a migration database using EGL 6.0.0.1, you must create the database again by following the instructions in "Creating the DB2 migration database" on page 459.

The following functions are new or improved since EGL 6.0.0.1:

- Support for migrating Web transaction programs and UI records. Also support for migrating generation options related to Web transactions.
- Support for migrating DL/I records, DL/I function I/O, PSB parts, EZEDL* special functions words, and the CSPTDLI special function word. Also support

for migrating generation options, resource association options, and linkage table options related to the IMSVS and IMSBMP environments, including GSAM and message queue support.

What's new for the VAGen migration tool since EGL 6.0.1?

Note: If you created a migration database using EGL 6.0.1, you must create the database again by following the instructions in “Creating the DB2 migration database” on page 459.

The following functions are new or improved since EGL 6.0.1:

- Stage 1 Smalltalk now supports subapplications. See “Mapping page” on page 151 for details on the new Collapse subapplication preference.
- Syntax migration improvements:
 - For single file mode, parts are now sorted by part name within part type in each file.
 - Record declarations within programs are now sorted by record name.
 - SQL comparison operations using the not sign are now converted to code page independent operators.
 - DL/I record declarations are not automatically added to a program based on the PSB for the program. The only DL/I record declarations that are added to the program are the ones necessary for the I/O options used by the program.
 - Maps containing arrays now use the **indexOrientation**, **columns**, **linesBetweenRows**, and **spacesBetweenColumns** properties to provide position information for standard arrays. This enables you to use the EGL Form Editor for the maps.
 - The migration tool always sets the **sign** property when migrating data items, fields on maps, and nonshared fields in UI records.
- Stage 2 and 3:
 - The migration tool adds an **import** statement if a function references a table.
 - Better merging logic if multiple migration sets are migrated at the same time or when parts are renamed using the **Renamer user exit** preference.
 - Performance improvements.

What's new for the VAGen migration tool since EGL 6.0.1.1?

Note: If you created a migration database using EGL 6.0.1.1, you must create the database again by following the instructions in “Creating the DB2 migration database” on page 459.

The migration tool itself has not changed significantly for this interim release (6.0.1.1 ifix3) of the *Migration Guide*. However, the *Migration Guide* has been updated with the following new information:

- Chapter 1 has been changed in the following ways:
 - Updated and expanded the section on “Terminology used in this book” on page 3 based on the availability of IBM Rational COBOL Generation Extension for zSeries and IBM Rational COBOL Runtime for zSeries.
 - Updated and expanded the section on “Planning your migration” on page 4.
 - Added the section on “References” on page 16. This section also includes a list of the available white papers related to migrating from VisualAge Generator to EGL.

- Chapter 2 has added the following sections:
 - “Determining how to organize your EGL source code” on page 53
- Chapter 9 has added the following sections:
 - “Converting VAGen preparation templates and procedures to EGL build scripts” on page 219
 - “Converting VAGen runtime templates” on page 220
 - “Converting the VAGen reserved words file” on page 221
 - “Installing the EGL server product for zSeries” on page 217
 - “Planning for dual maintenance of your source code” on page 224
- Chapter 10 has added the following sections:
 - “Differences in SQL support” on page 233. This consolidates information on SQL previously scattered throughout the chapter and expands the information based on customer experience.
 - Information about the migration of VAGen runtime environment variables and runtime properties.
- Appendix B added the following sections:
 - “Preparation templates and procedures” on page 386
 - “Runtime templates” on page 388
 - “Runtime environment variables” on page 390
 - “vgj.properties” on page 392
- Appendix E has been changed in the following ways:
 - Expanded the explanation for some messages and added new messages based on customer experience and enhancements to the EGL validation messages.
 - Added section “Reference information for messages - name resolution and qualification rules” on page 450. This section provides information that is useful in resolving messages that result from the differences between VisualAge Generator and EGL for name resolution when there is a field with the same name as another record, form, or DataTable.
- Appendix F has updated the list of APARs required for VisualAge Generator.

What's new for the VAGen migration tool since EGL 6.0.1.1 ifix003?

The following functions are new or improved since EGL 6.0.1.1 ifix003:

- Both Stage 1 tools now correctly place functions and data items that are referenced by multiple generatable parts in the commonParts.egl file. In addition, the Stage 1 tool on Smalltalk has been updated to add message tables as associates of programs that specify a message table prefix. This problem did not occur for the Stage 1 tool on Java.
- Both Stage 1 tools have been completely rewritten to improve the migration of large migration sets. This results in the following differences when running the Stage 1 tools:
 - If you previously captured a Stage 1 database, you can run Stage 2 and 3 with that existing database. This enables you to obtain the new EGL Version 7.1 syntax for your existing database.
 - If you want to run the new Stage 1 tools, you must follow these steps:
 - Install DB2 Version 8.1 with Fix Pack 15 or DB2 Version 8.2 with Fix Pack 8. These two fix pack levels are equivalent.
 - From a DB2 Command Prompt window, run setupDatabase.bat and setupTables.bat. You must run the two .bat files after installing the correct level of DB2.

- If you restore a Stage 1 database from prior to EGL V7.1 and then decide to run Stage 1 again, you must run `setupDatabase.bat` and `setupTables.bat` again because the restore of the old database resets the database definition to what was used for the earlier version of the migration tool.
- If you previously used the Stage 1 migration tool on Java, you need to perform the following additional steps after you install the new version of the Stage 1 tool:
 1. In the VAGen on Java Workbench window, click the **Projects** tab.
 2. Navigate to the **IBM VisualAge Generator EGL Migration** project.
 3. Expand the migration project and then expand the **com.ibm.vgj.mig** package.
 4. Within the package, right-click the **VAGenToEGLMigration** class and follow these steps:
 - a. Click **Properties** on the pop-up menu.
 - b. Click the **Class Path** tab.
 - c. Next to the **Project** path section, click **Edit**.
 - d. In the Class Path window, ensure that **IBM VisualAge Generator Runtime** is selected.
 - e. Click **OK** to close the Class Path window.
 - f. Click **OK** again to close the Properties window.
 5. Repeat step 4 for the **PreferencesUI** class.
- If you previously used the white papers to customize the Stage 1 tool on Java, note the following changes:
 - If you modified the file location, see “Customizing the Stage 1 migration tool” on page 135 for information on how to use the built-in customization to change the file location.
 - If you consolidated projects or packages, your customization code still applies, provided you limited your changes to the **IBM VisualAge Generator EGL Migration** project, the **com.ibm.vgj.mig** package, the Preferences class, and the **applyRenameRulesTo(String,String)** method as described in the white paper titled *How to Consolidate Projects and Packages during Stage 1 of the VisualAge Generator on Java to Enterprise Generation Language Migration Tool*.
- If you previously used the white papers to customize the Stage 1 tool on Smalltalk, note the following changes:
 - If you modified the file location, see “Customizing the Stage 1 migration tool” on page 159 for information on how to use the built-in customization to change the file location.
 - If you consolidated projects or packages, your customization code still applies, provided you limited your changes to the **HptEGLMigrationBaseApp** application, the **VAGenToEGLMapper** class, the **API** class category and the **convertProjectNameToWSED** and **convertApplicationToWSEDPackageName** public methods as described in the white paper titled *How to Consolidate Projects and Packages during Stage 1 of the VisualAge Generator on Smalltalk to Enterprise Generation Language Migration Tool*.
- Syntax migration in Stage 2:
 - The EGL source produced by the migration tool has been updated for the EGL language changes in EGL V7.1.

- Migration of SQL functions that specify the Execution time statement build option has been improved for situations in which default SQL is used and for situations in which a soft error occurs on the EGL **prepare** statement.

What's new for the VAGen migration tool since EGL 7.1?

There are no changes to the migration tool since EGL 7.1.

Known restrictions for the migration tools

Stage 1

- Stage 1 migration is not supported in the Linux or Vista environments.
- DB2 Version 9.x is not supported for Stage 1 migration in the Java environment.

Stages 2 and 3

- 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 EGL development environment each time you switch.
- Stage 2 and 3 are not supported 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 **defaultSelectCondition** or SQL clause as described in “SQL reserved words requiring special treatment” on page 254. This section provides a list of SQL keywords and details of the syntax required for the SQL **defaultSelectCondition** and SQL I/O statements.
- See “containerContextDependent Property” on page 40 for details on limitations on your use of this property.

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 associated parts, 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 associated parts is the preferred way of migrating with associates because it guarantees the maximum information. The tables in this chapter explain the differences between migrating with and without associated parts for the following part types:

- 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 part 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 the Execution time statement build option, test and generation create a temporary variable with the even length.

EGL: The DECIMAL primitive type is the replacement for the VAGen PACK type. 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 10. PACK data items with even length

Migrating with the associated part	Migrating without the associated part
<p>The migration tool migrates PACK data items based on the VAGen Do not honor evensql=y for items or variables migration preference.</p> <p>If the preference is not selected, the migration tool does the following things:</p> <ul style="list-style-type: none"> • 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. <p>If the preference is selected, the migration tool always uses an odd length (or 18 if the item is the maximum length) for all items or variables. The tool issues a warning message for any data item that specifies evensql=y.</p>	<p>The migration tool does the same things as mentioned in the <i>Migrating with the associated part</i> column.</p>

Table 10. PACK data items with even length (continued)

Migrating with the associated part	Migrating without the associated part
<p>Potential Problem 1: A problem arises if you select the preference so that evensql=y is migrated to an odd length. In this situation, there might be a runtime performance impact due to using host variable lengths that do not match the SQL column definition.</p> <p>Solution 1: Review the migration log information for any DataItem part that specified evensql=y. Review any SQL table and view definitions to determine whether the definitions match the DataItem part definition. Also review any SQL WHERE clauses and prepare statements that use variables that specify the DataItem part as a type definition.</p> <p>Potential Problem 2: A problem also arises if you clear the preference so that evensql=y is honored during migration and you later decide to 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.</p> <p>Solution 2: Review all DECIMAL DataItem part definitions and primitive fields in EGL records for even length items. Assess whether overflow might occur for any of these items.</p>	<p>Potential Problems: The same potential problems and solutions as listed in the <i>Migrating with the associated part</i> column apply.</p>

Shared edits and messages

VisualAge Generator: A shared DataItem part definition can specify default edits and messages for both maps and UI records.

EGL: There is only one set of edit and message properties for a DataItem part definition. The migration tool merges the map and UI properties for the data items.

Associated part needed for migration: Not applicable.

Table 11. Shared edits and messages

Migrating with the associated part	Migrating without the associated part
<p>The migration tool merges map and UI edits in the following ways:</p> <ul style="list-style-type: none"> For <i>each</i> edit or message, the migration tool converts the edits based on the first criterion in the following list that applies: <ul style="list-style-type: none"> 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 an associated UI edit, the migration tool converts the UI message to the corresponding EGL property. If the map message is specified without an 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. For a numeric data item, the tool migrates the sign edit in the following way: <ul style="list-style-type: none"> If the sign edit is not specified, the migration tool sets the EGL sign property to leading if any UI edit is specified. If no UI edits are specified, the migration tool sets the EGL sign property to none. 	<p>Except as noted later in “Map edit routine for shared data items” on page 68, the migration tool migrates the default edits and messages in the same way both with and without the associated parts.</p>
<p>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.</p> <p>Note: If you never used VAGen Web Transactions, only map edits should exist in VisualAge Generator and you should not have a problem.</p> <p>Possible Solution: Other than adding a comment to the DataItem part definition to list the VAGen map item edits and messages, there is nothing you can do for the DataItem part 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.</p> <p>Potential Problem 2: A problem can also arise if you use the item in an EGL VGUI record. The item might have some additional edits or messages that were migrated from the VAGen map edits.</p> <p>Solution: Always review the edits for fields defined with a type definition in a VGUI record.</p>	<p>The same potential problems mentioned in the <i>Migrating with the associated part</i> column apply. You can use the same solutions.</p>

Map edit routine for shared data items

VisualAge Generator: A shared data item definition can have a map 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 **validatorDataTable** and a **validatorFunction**. The data item can also have both a **validatorDataTableMsgKey** and a **validatorFunctionMsgKey**.

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

Table 12. Map edit routine for shared data items

Migrating with the associated part	Migrating without the associated part
<p>The first time the shared data item is migrated, the migration tool converts the map edit routine based on the first criterion in the following list that applies:</p> <ul style="list-style-type: none"> • If the editRoutineName is EZEC10 or EZEC11, the migration tool sets the validatorFunction property to the EGL equivalent system library function. The migration tool also sets the validatorFunctionMsgKey to the edit message, if any. • If the editRoutineName is a function, then the migration tool sets the validatorFunction property. The migration tool omits the validatorFunctionMsgKey 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 validatorDataTable property. The migration tool also sets the validatorDataTableMsgKey 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 <i>not</i> specified but the edit message <i>is</i> specified, the migration tool sets the validatorDataTableMsgKey to the edit message. <p>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 the edits on the map differ from what was specified in the shared item definition.</p>	<p>If a function or table with the same name as the editRoutineName is not available, the migration tool converts the map edit routine based on the first criterion in the following list that applies:</p> <ul style="list-style-type: none"> • If the editRoutineName is EZEC10 or EZEC11, the migration tool sets the validatorFunction property to the EGL equivalent system library function name. The migration tool also sets the validatorFunctionMsgKey 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 validatorFunction property. The migration tool omits the validatorFunctionMsgKey because it is not used in VisualAge Generator. • If an edit message is specified, the migration tool sets the validatorDataTable and validatorDataTableMsgKey. • Otherwise, the migration tool sets the validatorFunction property and issues an error message. <p>If the edit routine <i>is not</i> specified but the edit message <i>is</i> specified, the migration tool sets the validatorDataTableMsgKey to the edit message.</p>
<p>Potential Problem: A problem only arises if a function and DataTable 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.</p> <p>Possible Solution: Rename the DataTable part so there are no longer two parts with the same name. Specify both a validatorFunction and validatorDataTable property for the DataItem part. If you add the item to a text form, delete the validatorFunction or validatorDataTable property, whichever is not needed for that particular form.</p> <p>Disadvantage: You must modify your programs and forms to use the new DataTable name.</p>	<p>Potential Problem 1: A problem arises if the migration tool guesses incorrectly.</p> <p>Possible Solution: Correct the data item definition.</p> <p>Potential Problem 2: The same problem listed under the <i>Migrating with the associated part</i> column can also occur. You can use the same solution.</p>

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 DataItem part. The migration tool merges the map and UI **fillCharacter** properties for the data items.

Associated part needed for migration: Not applicable.

Table 13. Fill characters for shared data items

Migrating with the associated part	Migrating without the associated part
<p>The first time the shared data item is migrated, the migration tool converts the fill character based on the first criterion in the following list that applies:</p> <ul style="list-style-type: none">• If the UI edits specify a fill character, the migration tool migrates the character to the EGL fillCharacter property. The tool converts N to N because null fill is not valid for UI records in VisualAge Generator.• If the map edits specify a fill character, the migration tool migrates the character to the EGL fillCharacter property. The tool converts N to null fill.• If neither the UI edits nor the map edits specify a fill character, the migration tool does not set the EGL fillCharacter property.	<p>The migration tool does the same thing as mentioned in the <i>Migrating with the associated part</i> column.</p>
<p>Potential Problem: A problem only arises if you add a field using a type definition to a DataItem part that was migrated using one type of edits to a different type of user interface (form or VGUI record).</p> <p>Possible Solution: Always review the fillCharacter property when adding a new field to a form or VGUI record.</p>	<p>The same problem listed under the <i>Migrating with the associated part</i> column can occur. You can use the same solution.</p>

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 14. Redefined records

Migrating with the associated part	Migrating without the associated part
<p>When migrating a redefined record, the migration tool does the following things:</p> <ul style="list-style-type: none"> • 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. 	<p>When migrating a redefined record, the migration tool does the same thing mentioned in the <i>Migrating with the associated part</i> column.</p>
<p>When migrating a program, if RecordA is available, the migration tool does the following things:</p> <ul style="list-style-type: none"> • 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. 	<p>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 <i>not</i> include the redefines property in the declaration for RecordA.</p>
<p>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.</p>	<p>Potential Problem: A problem arises if the VAGen program uses RecordA as a redefinition. Immediately after migration, the program is not a valid EGL program because the definition for RecordA and the import statement are missing. EGL validation displays an error message in the Problems view. If you migrate RecordA and add the import statement to the program, this converts the program into a valid EGL program. However, two data areas are created: one for RecordA and one for RecordB. EGL does not detect this change during validation or generation. The program does <i>not</i> run the same way that it did in VisualAge Generator.</p> <p>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 RecordA is a redefinition for RecordB, update the program to include the redefines property for the declaration of RecordA.</p> <p>Considerations: The same considerations for new use listed under the <i>Migrating with the associated part</i> column can also occur.</p>

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 15. Level 77 items in records

Migrating with the associated part	Migrating without the associated part
<p>When migrating any working storage record that contains level 77 items, the migration tool does the following things:</p> <ul style="list-style-type: none"> • 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 Level77 suffix. You can specify the Level77 suffix by setting the VAGen Migration Syntax Preference. • 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. 	<p>When migrating any working storage record that contains level 77 items, the migration tool does the same things mentioned in the <i>Migrating with the associated part</i> column.</p>
<p>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.</p>	<p>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 <i>not</i> include a record declaration for a level 77 record.</p>
<p>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.</p>	<p>When migrating a function, if the working storage record is not available, the migration tool does <i>not</i> change the qualification of item names.</p>
<p>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.</p> <p>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.</p> <p>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.</p>	<p>Potential Problem 1: A problem arises if the primary working storage record contained level 77s and the program used the level 77s. Validation for the program fails due to missing item definitions.</p> <p>Solution: Add the level 77 record to the program.</p> <p>Potential Problem 2: A problem arises for a function if the qualified data item is really a VAGen level 77 item.</p> <p>Solution: Modify the function to provide the correct qualifier for the data item.</p> <p>Potential Problem 3: The same problem listed under the <i>Migrating with the associated part</i> column can also occur. You can use the same solution.</p> <p>Considerations: The same considerations for new use listed under the <i>Migrating with the associated part</i> also apply.</p>

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 SQL tables and keys for RecordA. If RecordA specifies a key item, that key is merged with the keys from RecordB when determining the default selection condition. For DL/I segment records, the field names in RecordB are also the names of the fields in the DL/I PSB.

EGL: A record can embed another record to obtain the record structure. Only the record structure is included. Each SQL record must explicitly state the entire set of SQL tables and keys it references. For DL/I segment records, if RecordB is also a DL/I segment, then RecordB can provide the **dliFieldName** property to specify the name of the field in the DL/I PSB. Alternatively, if RecordB is not a DL/I segment, then RecordA can provide the **dliFieldName** property as an override to the **embed** keyword.

Associated part needed for migration: If RecordA is an SQL record or a DL/I segment, you need the record specified as the altspec record (RecordB).

Table 16. Alternate specification records

Migrating with the associated part	Migrating without the associated part
<p>If RecordA is an SQL record that specifies an alternate specification of RecordB and RecordB is available, the migration tool does the following things for RecordA:</p> <ul style="list-style-type: none"> Creates the list of table names from the list of tables specified in RecordB. Creates the list of keys by merging the following items: <ul style="list-style-type: none"> The items, if any, in RecordB that specified key=yes. The item, if any, in RecordA that is specified as the key item. <p>The order of the keys is the order in which the items appear in the structure of RecordB.</p> Includes the embed keyword pointing to RecordB. Migrates any !itemColumnName variables in the default selection conditions of RecordA to the corresponding SQL column names from RecordB. 	<p>If RecordA is an SQL record that specifies an alternate specification of RecordB, and RecordB is not available, the migration tool does the following things for RecordA:</p> <ul style="list-style-type: none"> 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 keyword 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.
<p>If RecordA is a DL/I segment record that specifies an alternate specification of RecordB and RecordB is available and is not a DL/I segment record, the migration tool does the following things for RecordA:</p> <ul style="list-style-type: none"> Includes the embed keyword pointing to RecordB. Includes an override statement for each field from RecordB that must be renamed. The override statement sets the dliFieldName property to the original VAGen field name so that the DL/I field name is available in EGL. 	<p>If RecordA is a DL/I segment record that specifies an alternate specification of RecordB and RecordB is not available, the migration tool does the following things for RecordA:</p> <ul style="list-style-type: none"> Includes the embed keyword pointing to RecordB. Issues an error message that the tool cannot determine whether any override statements are required to preserve the dliFieldName information.

Table 16. Alternate specification records (continued)

Migrating with the associated part	Migrating without the associated part
<p>Potential Problem 1: A problem only arises for SQL if the definition for RecordB differs, generally in different subsystems.</p> <p>Solution 1: Duplicate the definition of RecordA so that each subsystem has its own definition of RecordA.</p> <p>Potential Problem 2: A problem also arises for DL/I if the definition for RecordB differs, generally in different subsystems.</p> <p>Solution 2: The solution is the same as that described in Solution 1.</p>	<p>Problem 1: EGL validation for SQL record RecordA results in messages in the Problems view.</p> <p>Solution 1: 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 names from RecordB.</p> <p>Potential Problem 2: A problem arises for DL/I segment RecordA if RecordB contains fields that must be renamed due to reserved words or because they start with the # or @ symbol. In this case, if the field is used in a default SSA and the renamed field name is longer than 8 characters or contains a character such as underscore that is invalid for DL/I, then EGL validation displays an error message in the Problems view.</p> <p>Solution 2: Edit RecordA and include the override statements to specify the DL/I field name.</p> <p>Potential Problem 3: A problem also arises for DL/I segment RecordA if the renamed field is used in a default SSA and is not a valid DL/I name. In this case, there is a runtime error when you run the program.</p> <p>Solution 3: Edit RecordA and include the override statements to specify the DL/I field name.</p>

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** property for record definitions. Setting this property to YES enables you to specify that the DataItem parts used for type definitions are based on the name space of the program part.

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 17. Different definitions with the same record name

Migrating with the associated part	Migrating without the associated part
<p>The migration tool does not set the containerContextDependent property to YES for record definitions. If you need this capability, you must set the property for each record that requires it.</p> <p>Note: See “containerContextDependent Property” on page 40 for details on limitations on using this property.</p>	<p>The migration tool does the same thing as mentioned in the <i>Migrating with the associated part</i> column.</p>

Reserved words and UI record names

VisualAge Generator: VisualAge Generator does not have reserved words. VisualAge Generator permits the # and @ symbols as the first character of a UI record name or a field in a UI record.

EGL: EGL has reserved words. In addition, EGL does not permit the # or @ symbol as the first character of a record name. A UI record name cannot be a reserved word and the name cannot start with the # or the @ symbol. A field in a UI record cannot be a reserved word and the name cannot start with the # or the @ symbol. Name resolution conflicts can also occur if the UI record or field name is on the migration tool extended reserved word list.

Associated part needed for migration: Not applicable.

Table 18. Reserved word and UI record names

Migrating with the associated part	Migrating without the associated part
<p>When migrating a UI record, if the record name is on the migration tool extended reserved word list or starts with the # or @ symbol, the migration tool does the following things:</p> <ul style="list-style-type: none">• Renames the UI record by including the Renaming prefix at the beginning of the record name. This is identical to the renaming that the migration tool does for other VAGen records. You can specify the Renaming prefix by setting the VAGen Migration Syntax Preference.• Includes the alias property and sets it to the original VAGen name for the UI record.• Changes the .egl file name for the UI record to match the renamed UI record during Stage 3 of migration.• Issues a warning message.	<p>The migration tool does the same thing as mentioned in the <i>Migrating with the associated part</i> column.</p>
<p>When migrating a field in a UI record, if the field name is on the migration tool extended reserved word list or starts with the # or @ symbol, the migration tool does the following things:</p> <ul style="list-style-type: none">• Renames the field by including the Renaming prefix at the beginning of the field name. This is identical to the renaming that the migration tool does for fields in other VAGen records.• Includes the alias property for the field and sets it to the original VAGen name for the field.	<p>The migration tool does the same thing as mentioned in the <i>Migrating with the associated part</i> column.</p>
<p>The migration tool treats a UI record like any other record for the purposes of renaming. The tool uses the EGL part name for all references to a UI record. This includes references in the following places:</p> <ul style="list-style-type: none">• Program part:<ul style="list-style-type: none">– First UI record– Record declaration• UI record part - First UI record in the Link parameters• Function part - any use of the record in a statement	<p>The migration tool does the same thing as mentioned in the <i>Migrating with the associated part</i> column.</p>

Table 18. Reserved word and UI record names (continued)

Migrating with the associated part	Migrating without the associated part
Potential Problem: None. The alias property provides the same JSP name as in VAGen.	Potential Problem: None. The alias property provides the same JSP name as in VAGen.

Handling ambiguous situations for tables

Reserved words and table names

VisualAge Generator: VisualAge Generator does not have reserved words. The # or @ symbols are not valid in VAGen table names.

EGL: EGL has reserved words. In addition, EGL does not permit the # or @ symbol as the first character of a part name. A DataTable name cannot be a reserved word. Name resolution conflicts can also occur if the DataTable name is on the migration tool extended reserved word list.

Associated part needed for migration: Not applicable.

Table 19. 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 is on the migration tool extended 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 <i>Migrating with the associated part</i> column.
<p>Potential Problem: A problem only arises if a DataTable name is on the migration tool extended reserved word list. EGL validation displays an error message in the Problems view.</p> <p>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, UI records, and data item definitions. If you change the name in EGL, you must change the name of the DataTable and all references to it. This includes references in the following places:</p> <ul style="list-style-type: none"> • Program use declaration statements • Logic statements in programs and functions • Data item validatorDataTable properties • Form field validatorDataTable properties • VGUI record validatorDataTable properties <p>If you want to keep the original table name as the name for the generated DataTable, set the alias property to the original DataTable name. If you do not specify the alias property, be sure to change any non-EGL references to the DataTable name, including CICS program definitions.</p>	The same problem listed under the <i>Migrating with the associated part</i> 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 # or @ symbol are not valid in VAGen map group names.

EGL: EGL has reserved words. In addition, EGL does not permit the # or @ symbol as the first character of a part name. A FormGroup name cannot be a reserved word. Name resolution conflicts can also occur if the FormGroup name is on the migration tool extended reserved word list.

Associated part needed for migration: Not applicable.

Table 20. 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 is on the migration tool extended 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 <i>Migrating with the associated part</i> column.
<p>Potential Problem: A problem only arises if the FormGroup name is on the migration tool extended reserved word list. EGL validation displays an error message in the Problems view.</p> <p>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.</p>	The same problem listed under the <i>Migrating with the associated part</i> 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 21. Map group and FormGroup requirements

Migrating with the associated part	Migrating without the associated part
<p>If a map group does not exist, the migration tool does the following things:</p> <ul style="list-style-type: none"> • 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. 	<p>The migration tool does the same things mentioned in the <i>Migrating with the associated part</i> column. However, if you do not have the map group and all its maps in the same migration set, there might be problems.</p>
<p>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.</p> <p>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.</p>	<p>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 does not include all the forms.</p> <p>Possible Solution 1: Be sure the migration set includes all maps with the same map group name.</p> <p>Possible Solution 2: Add the missing forms to the EGL file and nest them within the FormGroup definition.</p>

Floating areas and starting positions

VisualAge Generator: VisualAge Generator permits having different floating area sizes and starting positions for different device types that have the same device size. If no floating area is specified for a device, the entire device size is considered to be the floating area.

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 22. Floating areas and starting positions

Migrating with the associated part	Migrating without the associated part
<p>The migration tool does the following things:</p> <ul style="list-style-type: none"> • 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. If two or more VAGen devices convert to identical EGL device size and margin specifications, the migration tool only includes one entry for EGL. 	<p>The migration tool does the same things mentioned in the <i>Migrating with the associated part</i> column.</p>

Table 22. Floating areas and starting positions (continued)

Migrating with the associated part	Migrating without the associated part
<p>Potential Problem 1: A problem arises if two or more devices with the same device size specify different floating area sizes or starting positions in VisualAge Generator. EGL validation displays an error message in the Problems view.</p> <p>Possible Solution 1: 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.</p>	<p>The same problem listed under the <i>Migrating with the associated part</i> column can occur. You can use the same solution.</p>
<p>Potential Problem 2: A problem also arises if you used created a floating form with multiple devices with the same size and explicitly specified a floating area size for some devices, but used the default (full screen) floating area for other devices of the same size. In this case, when the migration tool converts the map group, it only converts the explicitly specified floating areas. The migration tool does not issue a warning message. One of the following things might occur:</p> <ul style="list-style-type: none"> • If the floating form is only intended for the device that used the default (full screen) floating area, the floating form might be too large to fit in the floating area of the same-size device that explicitly specified a floating area. In this case, EGL validation issues an error message in the Problems view. • If the floating form fits in the floating area of the device that explicitly specified a floating area, then, at runtime, if you use the physical device that expected the default (full screen) floating area, the floating form is displayed in the wrong place based on the explicitly specified floating area. <p>Possible Solution 2: If there is an EGL error message on the Problems view, set the margins for the explicitly specified floating area to 0 (so the entire screen is the floating area). If there is no EGL error message on the Problems view, then testing is the only way to determine that the problem has occurred.</p>	

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 # and @ symbols are not valid in VAGen map group names, but the # and @ symbols are 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. 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 # or @ 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. For COBOL generation, the form name or the alias is limited to 8 characters. 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 23. Map names and help map names

Migrating with the associated parts	Migrating without the associated parts
<p>Based on the first program to migrate either the main map group or the help map group, the migration tool does the following things:</p> <ul style="list-style-type: none"> • Performs any renaming for map names due to the migration tool extended reserved words list or the # or @ symbol being used as the first character of the map name portion of the name. Maps in both the main map group and help map group for the program are renamed as necessary. • Checks the names of <i>all</i> maps in the help map group for the program for duplicate names with the main map group. • Compares the program name to the names of all maps in the main map group and help map group for the program. <p>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.</p> <p>If a map in the help map group has the same name as any map in the main map group for the program, the migration tool does the following things:</p> <ul style="list-style-type: none"> • If the help map contains only constants, the migration tool does the following things: <ul style="list-style-type: none"> – 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 the help map contains variables, the migration tool does the following things: <ul style="list-style-type: none"> – Issues an error message. – Does not rename the map. <p>This is because the map could be used by some other program that specifies the help map group as the main map group for that program.</p> <p>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 is done for conflicting map names in the help map group and main map group.</p> <p>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.</p>	<p>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 things:</p> <ul style="list-style-type: none"> • Performs any renaming for map names due to the migration tool extended reserved word list or the # or @ symbol. • Does <i>not</i> check for additional renaming of help maps.

Table 23. Map names and help map names (continued)

Migrating with the associated parts	Migrating without the associated parts
<p>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.</p> <p>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.</p> <p>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.</p> <p>Possible Solution: Same as possible solution for Problem 1.</p> <p>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.</p> <p>Possible Solution: Rename all the necessary forms in the help FormGroup by adding your help map suffix to the name. Include the alias property to provide the original help map name for use during generation. Change all corresponding text forms in all FormGroups to specify the new help form name.</p>	<p>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.</p> <p>Possible Solutions: The same solutions as shown for <i>Migrating with the associated part</i> apply.</p>

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 field length 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 24. Numeric variable fields

Migrating with the associated part	Migrating without the associated part
<p>When migrating a numeric field on the map, the migration tool sets the length and fieldLen in the following way:</p> <ul style="list-style-type: none"> • 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 in the following way: <ul style="list-style-type: none"> – 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. 	<p>The migration tool does the same thing as mentioned in the <i>Migrating with the associated part</i> column.</p>
<p>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.</p> <p>Solution: Change the form definition so that the fieldLen is large enough to contain the largest possible number that can occur at run time and all the formatting characters that you specify.</p>	<p>The same problem listed under the <i>Migrating with the associated part</i> column can occur. You can use the same solution.</p>

Map variable 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 **validatorDataTable** and a **validatorFunction**. A form field can also have both a **validatorDataTableMsgKey** and a **validatorFunctionMsgKey**.

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

Table 25. Variable map fields and edit routines

Migrating with the associated part	Migrating without the associated part
<p>The first time the map is migrated, the migration tool converts the edit routine based on the first criterion in the following list that applies:</p> <ul style="list-style-type: none"> • If the editRoutineName is EZEC10 or EZEC11, the migration tool sets the validatorFunction property to the EGL-equivalent system library function. The migration tool also sets the validatorFunctionMsgKey to the edit message, if any. • If the editRoutineName is a function, then the migration tool sets the validatorFunction property. The migration tool omits the validatorFunctionMsgKey because it is not used in VisualAge Generator. • If the editRoutineName is a table, then the migration tool sets the validatorDataTable property. The migration tool also sets the validatorDataTableMsgKey to the edit message, if any. 	<p>If a function or table with the same name as the editRoutineName is not available, the migration tool converts the edit routine based on the first criterion in the following list that applies:</p> <ul style="list-style-type: none"> • If the editRoutineName is EZEC10 or EZEC11, the migration tool sets the validatorFunction property to the EGL equivalent system library function name. The migration tool also sets the validatorFunctionMsgKey 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 validatorFunction property. The migration tool omits the validatorFunctionMsgKey because it is not used in VisualAge Generator. • If an edit message is specified, the migration tool sets the validatorDataTable and validatorDataTableMsgKey. • If an edit message is not specified, the migration tool sets the validatorFunction property and issues an error message.
<p>Potential Problem: A problem only arises if a 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.</p> <p>Possible Solution: Review programs that share a FormGroup. If the situation arises, create a separate FormGroup to use the validatorDataTable.</p> <p>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.</p>	<p>Potential Problem: A problem only arises if the migration tool guesses incorrectly. Any program that uses this form might expect a DataTable when the migration tool specified a function.</p> <p>Possible Solution: Review the uses of maps that have error messages.</p>

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 user from typing non-numeric data in a variable field.

EGL: EGL only supports the **isDecimalDigit** property 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 26. Map fields and the numeric hardware attribute

Migrating with the associated part	Migrating without the associated part
<p>The migration tool does the following things:</p> <ul style="list-style-type: none"> For any character variable on a map that specified the numeric hardware attribute, the tool sets the isDecimalDigit property to YES. 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. 	<p>The migration tool does the same thing as mentioned in the <i>Migrating with the associated part</i> column.</p>
<p>Potential Problem: There is a slight change in runtime behavior. Users can now type non-numeric data into numeric fields. EGL issues a runtime error message if they do. In addition, the users cannot enter a sign or a decimal point in the character fields that have the isDecimalDigit property set to YES.</p> <p>Possible Solution: Consider notifying your users that this is an expected difference when changing from VAGen-generated code to EGL-generated code. If the character field needs to contain a sign or a decimal point, change the primitive type to numeric and remove the isDecimalDigit property.</p>	<p>The same problem listed under the <i>Migrating with the associated part</i> column can occur. You can use the same solution.</p>

Map arrays and attributes

VisualAge Generator: VisualAge Generator permits using different attributes for the elements of an array. including the following attributes:

- input required
- require fill on input
- numeric hardware attribute
- light pen detect

However, these attributes are generally set to the same value for all elements of the array

EGL: In EGL, only the following properties can be overridden for an array item:

- the field presentation properties:
 - **color**
 - **highlight**
 - **intensity**
 - **protect**
 - **modified**
 - **outline**
- **cursor**,
- **position**
- **value**

Associated part needed for migration: Not applicable.

Table 27. Map arrays and attribute fields

Migrating with the associated part	Migrating without the associated part
<p>The migration tool uses the following properties for the first element of the array (array index 1) to set the EGL equivalent properties:</p> <ul style="list-style-type: none"> • input required • require fill on input • numeric hardware attribute • light pen detect <p>EGL uses the properties for the first element of the array for <i>all</i> the elements of the array.</p>	<p>The migration tool does the same thing as mentioned in the <i>Migrating with the associated part</i> column.</p>
<p>Potential Problem: A problem only arises if you used different attributes for the elements of the array.</p> <p>Possible Solution: Change the properties for the first element of the array to the least restrictive values and add logic to a function specified in the validatorFunction property to verify that each element of the array meets the necessary criteria. Also notify your users of any differences in the appearance of the form at runtime.</p>	<p>The same problem listed under the <i>Migrating with the associated part</i> column can occur. You can use the same solution.</p>

Unnamed map variable fields

VisualAge Generator: VisualAge Generator permits 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 28. Unnamed map variable fields

Migrating with the associated part	Migrating without the associated part
<p>For any unnamed variable fields on the map, the migration tool checks to see if any of the following values are specified:</p> <ul style="list-style-type: none"> • Initial value • Protect = yes • Cursor = yes • Outlining other than "No outlining" • Highlighting other than "No highlighting" <p>If any of the listed values are specified, the migration tool creates a constant field with the corresponding EGL properties and issues a warning message.</p>	<p>The migration tool does the same thing as mentioned in the <i>Migrating with the associated part</i> column.</p>
<p>If none of the properties (or only default values) are specified for the unnamed variable field, the migration tool does the following things:</p> <ul style="list-style-type: none"> • Does not create a constant field on the form. • Issues a warning message. 	<p>The migration tool does the same thing as described in the <i>Migrating with the associated part</i> column.</p>

Table 28. Unnamed map variable fields (continued)

Migrating with the associated part	Migrating without the associated part
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 setting the **protect** property to either **protect** or **skipProtect**. For print forms, EGL does not support the **protect** property.

Associated part needed for migration: Not applicable.

Table 29. 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 things: <ul style="list-style-type: none"> • If the form is a text form, the migration tool sets the EGL protect property to skipProtect 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 <i>Migrating with the associated part</i> column.
Potential Problem: None.	Potential Problem: None.

Fields at row=0, column=0

VisualAge Generator: VisualAge Generator Version 4.5 tolerates fields positioned at row=0, column=0 from older releases of Cross System Product or VisualAge Generator. However, VisualAge Generator Version 4.5 does not provide a way to create fields at this position. You cannot 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 30. Fields at row=0, column=0

Migrating with the associated part	Migrating without the associated part
<p>When migrating a form, if a field is positioned at row=0, column=0, the migration tool does the following things:</p> <ul style="list-style-type: none"> • If the field is a constant field and the first character of the value is blank, the migration tool does the following things: <ul style="list-style-type: none"> – 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 things: <ul style="list-style-type: none"> – 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. 	<p>The migration tool does the same thing as mentioned in the <i>Migrating with the associated part</i> column.</p>
<p>Potential Problem 1: If the field cannot be changed and is at position=[0,0], EGL validation displays an error message in the Problems view.</p> <p>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.</p> <p>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.</p> <p>Solution 2: Review the migration warning messages and be sure to test any forms where the migration tool adjusted the position of a field.</p>	<p>Potential Problem: The same problems listed under the <i>Migrating with the associated part</i> column can occur. You can use the same solutions.</p>

Handling ambiguous situations for programs

Program names and reserved words

VisualAge Generator: VisualAge Generator does not have reserved words. The # and @ symbols are not valid in VAGen program names.

EGL: EGL has reserved words. In addition, EGL does not permit the # or @ symbol as the first character of a part name. A program name cannot be a reserved word. Name resolution conflicts can also occur if the program name is on the migration tool extended reserved word list.

Associated part needed for migration: Not applicable.

Table 31. 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 is on the migration tool extended 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 <i>Migrating with the associated part</i> column.
<p>Potential Problem: A problem only arises if a program name is on the migration tool extended reserved word list. EGL validation displays an error message in the Problems view.</p> <p>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 and references in linkage options parts. Also change the names of any bind control or link edit parts that correspond to this program. 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 and transaction definitions.</p>	The same problem listed under the <i>Migrating with the associated part</i> column can occur. You can use the same solution.

Implicit data items in programs

VisualAge Generator: VisualAge Generator permits 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). However, the use of implicit items is not generally considered to be a good practice.

EGL: EGL does not permit implicit data items.

Associated part needed for migration: Not applicable.

Table 32. 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 <i>Migrating with the associated part</i> column.

Table 32. Implicit data items in programs (continued)

Migrating with the associated part	Migrating without the associated part
<p>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 an implicit item, EGL validation displays an error message in the Problems view.</p> <p>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. For information about a white paper that can help you create the implicit items before you run Stage 1 of migration, see "References" on page 16.</p>	<p>The same problem listed under the <i>Migrating with the associated part</i> column can occur. You can use the same solution.</p>

Associated program parts

VisualAge Generator: The associated parts for a program can be in multiple projects and packages for VisualAge Java or in multiple configuration maps and applications for VisualAge Smalltalk.

EGL: The associated parts for a program 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 38 for additional information on **import** statements.

Table 33. Associated program parts

Migrating with the associated part	Migrating without the associated part
<p>The migration tool does the following things:</p> <ul style="list-style-type: none"> 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 things: <ul style="list-style-type: none"> Includes a declaration for the primary working storage record for the program. 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 declarations for UI records that are used as the First UI Record, in a CONVERSE, or an XFER statement. Includes declarations for DL/I segment records that the program references in I/O options, either directly or because they are in the hierarchical path to the I/O object. Includes use declaration statements for any tables in the VAGen Tables and Additional Records list. Includes a use declaration statement for a message table if the table is explicitly referenced in a statement in the program. Includes a use declaration statement for each map within the main map group that the program references in a CONVERSE, DISPLAY, or CLOSE I/O option, in an XFER with a map statement, as a called parameter, or as the First Map of the program. Includes a use declaration statement for the help map group of the program. Includes a variable declaration for the PSB for the program, if the VAGen program specified a PSB. 	<p>The migration tool makes use of all program associates that are available.</p>

Table 33. Associated program parts (continued)

Migrating with the associated part	Migrating without the associated part
<p>Potential problem: None.</p>	<p>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 examples:</p> <ul style="list-style-type: none"> • 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. • Missing UI record declarations. • Missing DL/I segment record declarations for segments referenced in SSAs. • Missing use declaration statement for a message table if the table is explicitly referenced in a statement in the program. • Missing use declaration statement for maps within the map group. <p>Except for the missing redefines property, errors in the Problems view can help you identify the problem(s).</p> <p>Note: The migration tool does <i>not</i> detect all missing parts.</p> <p>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 associated parts for the program are migrated together.</p> <p>Possible Solution 1B: Locate the missing parts in EGL and correct the EGL program.</p> <p>Potential Problem 2: For missing level 77 items, see “Level 77 items in records” on page 71.</p> <p>Potential Problem 3: For missing redefined records, see “Redefined records” on page 70.</p>

Program with EZEDLPCB in called parameter list

VisualAge Generator: VisualAge Generator uses EZEDLPCB[*n*] to indicate that a program is to receive a PCB as a parameter. *n* must be a numeric literal. The value of *n* must be 0 (for the I/O PCB) or a number that corresponds to one of the PCBs defined in the PSB part for the program.

EGL: EGL uses a variable name with a type definition of xxxx_PCBRecord to indicate that a program is to receive a PCB as a parameter. xxxx must be IO, ALT, DB, or GSAM based on the type of the PCB. EGL also requires the **pcbParms** property to provide the mapping of the PCB variable name to its corresponding position in the PSBRecord for the program.

Associated part needed for migration: The PSB part.

Table 34. Program with EZEDLPCB in called parameter list

Migrating with the associated part	Migrating without the associated part
<p>When migrating the program, if the program specifies EZEDLPCB[<i>n</i>] as a parameter and the PSB part is available, the migration tool does the following things:</p> <ul style="list-style-type: none"> Includes the variable <i>pcb_n</i> as a parameter and specifies the following type definitions based on the corresponding PCB in the PSB: <ul style="list-style-type: none"> The migration tool sets the type definition of EZEDLPCB[0] to IO_PCBRecord. If <i>n</i> corresponds to a PCB in the PSB part, the tool sets the type definition based on the PCB type in the PSB. If <i>n</i> is greater than the number of PCBs in the PSB part, the tool issues a message and sets the type definition to EZE_UNKNOWN_PCB_TYPE. Lists all the <i>pcb_n</i> variables in their corresponding place in the pcbParms property. The I/O PCB is <i>pcb0</i> and, if specified as a parameter, is listed in the first position of the pcbParms property. The remaining <i>pcb_n</i> variables are listed at position <i>n</i>+1 in the pcbParms list. 	<p>When migrating the program, if the program specifies EZEDLPCB[<i>n</i>] as a parameter and the PSB part is not available, the migration issues a message and does the following things:</p> <ul style="list-style-type: none"> Includes the variable <i>pcb_n</i> as a parameter and specifies the following type definitions based on the PCB number: <ul style="list-style-type: none"> The migration tool sets the type definition of EZEDLPCB[0] to IO_PCBRecord. The migration tool issues a message and sets the type definition of all other <i>pcb_n</i> variables to EZE_UNKNOWN_PCB_TYPE. Issues a message and sets the pcbParms property to EZE_UNKNOWN_PCB_MAPPING.
<p>Potential Problem: A problem only arises if the PSB part contains fewer PCBs than the highest value of <i>n</i> for an EZEDLPCB[<i>n</i>] parameter.</p> <p>Solution: The program is invalid in VisualAge Generator. Review your program logic to determine whether to change the parameter list or the PSBRecord for the program.</p>	<p>Potential Problem 1: The correct PCB type definitions and pcbParms property must be provided.</p> <p>Solution: Locate the PSBRecord for the program. Edit the program and correct the PCB type definitions. Also provide the correct mapping of the <i>pcb_n</i> variables for the pcbParms property.</p>

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 35. Intermediate variables required for migration

Migrating with the associated part	Migrating without the associated part
<p>When migrating any program, the migration tool always includes the following declarations:</p> <ul style="list-style-type: none"> • <code>custPrefixEZEREPLY</code> • <code>custPrefixEZE_ITEMLEN</code> • <code>custPrefixEZE_WAIT_TIME</code> <p>If the Do not initialize old EZESYS values migration preference is not selected, the migration tool also does the following things:</p> <ul style="list-style-type: none"> • Includes a declaration for <code>custPrefixEZESYS</code>. • Includes an initialization statement to set the value of <code>custPrefixEZESYS</code> to the old VAGen EZESYS value. <p><code>custPrefix</code> is the same prefix that is used for changing part names that conflict with the migration tool extended reserved word list. Use the VAGen Migration Preferences to set the value for <code>custPrefix</code>.</p>	<p>The migration tool does the same things mentioned in the <i>Migrating with the associated part</i> column.</p>
<p>The four variables are used for migrating the following code:</p> <ul style="list-style-type: none"> • VAGen service routines if the (REPLY option is not specified. In this situation, the current value of handleSysLibraryErrors 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. • References to EZESYS in statements other than IF, WHILE, and TEST where the old VAGen value is required. 	<p>The migration tool does the same things mentioned in the <i>Migrating with the associated part</i> column.</p>
<p>Potential Problems: A problem only arises if you select the VAGen Migration Preference Do not initialize old EZESYS values during migration and you use EZESYS in statements other than IF, WHILE, or TEST. In this situation the migration tool uses <code>custPrefixEZESYS</code> in the statement, but programs do not have a declaration and initialization statement for <code>custPrefixEZESYS</code>. EGL validation displays an error message in the Problems view.</p> <p>Potential Solution 1: Change your EGL logic to use the new values for sysVar.systemType .</p> <p>Potential Solution 2: Add a declaration and an initialization statement for <code>custPrefixEZESYS</code> to any program that needs to use the old VAGen value for EZESYS.</p>	<p>The same problem listed under <i>Migrating with the associated part column</i> can occur. You can use the same solutions.</p>

Handling ambiguous situations for functions, including I/O statements

DISPLAY I/O option 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 main map group for the program.

Table 36. Display I/O option 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 EGL code: <ul style="list-style-type: none">• display <i>textForm</i> if the map is a display map• print <i>printForm</i> if the map is a printer map	If a map with this name is not available, the migration tool does the following things: <ul style="list-style-type: none">• Converts to display <i>form</i>• Issues a warning message that the map type could not be determined

Table 36. Display I/O option for maps (continued)

Migrating with the associated part	Migrating without the associated part
<p>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.</p> <p>Solution 1: Use VisualAge Generator compatibility mode. Edit the function and change the print statement to a display statement.</p> <p>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.</p> <p>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 example:</p> <pre>if (sysVar.systemType is zoscics) DISPLAY_FUNCTION(); else PRINT_FUNCTION(); end</pre> <p>where DISPLAY_FUNCTION and PRINT_FUNCTION use the display and print statements, respectively.</p> <p>Possible Solution 2B: If the function migrated to a display statement, as shown in the following code:</p> <pre>before-logic display TextForm; after-logic</pre> <p>Consider changing the statements to use functions, as shown in the following code:</p> <pre>before-logic-function(); display TextForm; after-logic-function();</pre> <p>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 the print statement, but still use the common before-logic-function and after-logic-function.</p> <p>Disadvantage: This has the potential to ripple back into functions that use the original DISPLAY function.</p>	<p>Potential Problem: The same potential problems and possible solutions as listed in the <i>Migrating with the associated part</i> column apply.</p>

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 options:

- 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 37. File and database I/O error routines

Migrating with the associated part	Migrating without the associated part
<p>Based on the first migration of this function, if there is a program available, the migration tool does the following things:</p> <ul style="list-style-type: none"> • Changes an I/O error routine that specifies a program main function to the following form: <pre>try I/O-Statement; onException exit stack <i>functionName</i>; end</pre> • Changes an I/O error routine that specifies a non-main function to the following form: <pre>try I/O-Statement; onException <i>functionName()</i>; end</pre> 	<p>If there is no program available, the migration tool does the following things:</p> <ul style="list-style-type: none"> • Assumes that the function named in an I/O error routine is a non-main function and changes it to the following form: <pre>try I/O-Statement; onException <i>functionName()</i>; end</pre> • Does not issue a warning message due to the high volume of messages that could be issued and the likelihood that messages would be ignored or hide other serious error messages.
<p>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.</p> <p>Note: EGL validation does not display an error message in the Problems view. Generation does not detect an error. However, the program does <i>not</i> run as it would in VisualAge Generator. Instead of popping the stack as in VisualAge Generator, EGL adds the main function to the stack.</p> <p>Possible Solution: If this situation arises, create a new version of this I/O function with the proper syntax for transferring to a main function.</p> <p>Disadvantage: This technique has the potential to ripple back into other functions that invoke the I/O function.</p>	<p>Potential Problem: A problem arises if this function is used in a program where the I/O error routine is a main function.</p> <p>Note: EGL validation does not display an error message in the Problems view. Generation does not detect an error. However, the program does <i>not</i> run as it would in VisualAge Generator. Instead of popping the stack as in VisualAge Generator, EGL adds the main function to the stack.</p> <p>Possible Solution: The same solution listed for <i>Migrating with the associated part</i> applies.</p>

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 the Execution time statement build option. 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 38. SQL I/O statements

Migrating with the associated part	Migrating without the associated part
<p>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 the Execution time statement build option. If the SQL statement in the function was modified, the migration tool does the following things:</p> <ul style="list-style-type: none"> • 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 alternate specification record for the I/O object. • Converts any !itemColumnName 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 254 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. <p>Note:</p> <ul style="list-style-type: none"> • See “SQL I/O and missing required SQL clauses” on page 98 for details on problems related to missing SQL clauses. • See “SQL I/O and !itemColumnName” on page 103 for details on problems related to using !itemColumnNames. 	<p>If the SQL record or its alternate specification record are not available, the migration tool has only the SQL statement modifications and the 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 things:</p> <ul style="list-style-type: none"> • 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 <i>clauseName</i> is the External Source Format key word for the missing SQL clause (for example, SELECT or VALUES). • 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 254 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. <p>Note:</p> <ul style="list-style-type: none"> • See “SQL I/O and missing required SQL clauses” on page 98 for details on problems related to missing SQL clauses. • See “SQL I/O and !itemColumnName” on page 103 for details on problems related to using !itemColumnNames.

Table 38. SQL I/O statements (continued)

Migrating with the associated part	Migrating without the associated part
<p>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.</p> <p>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.</p> <p>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.</p> <p>Disadvantage: This has the potential to ripple back into functions that use this I/O function.</p> <p>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 the EGL main function for each program. Alternatively, make the change to table name host variables in VisualAge Generator and migrate the program, record and function again.</p> <p>Disadvantage: There are potential performance implications because this changes from static to dynamic SQL.</p> <p>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. EGL validation displays an error message in the Problems view.</p> <p>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 254 for the list of SQL reserved words and an example of the required syntax.</p>	<p>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.</p> <p>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_SQL. Determine the proper tables clause based on the record definition. See "SQL I/O and missing required SQL clauses" on page 98 for information about recreating the SQL clause in EGL. See "SQL I/O and !itemColumnName" on page 103 for information about correcting any !itemColumnName variables.</p> <p>Other potential problems: The same potential problems and solutions as shown for <i>Migrating with the associated part</i> apply.</p>

SQL I/O and missing required SQL clauses

VisualAge Generator: VisualAge Generator Version 4.5 stores 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 Version 4.5, then some of the required SQL clauses might be missing. In addition, in VisualAge Generator Version 4.5, default SQL can be used with the Execution time statement build option. At generation time, this results in an SQL PREPARE statement, followed by an OPEN, and then (for INQUIRY or UPDATE) followed by

a FETCH. VisualAge Generator automatically creates the information required for the SQL PREPARE statement. The following table shows which clauses, in addition to the tables clause, that the migration tool might create for the various I/O options:

Table 39. SQL clauses that the migration tool might create

I/O option	Missing SQL clause with or without Execution time statement build	Default SQL with Execution time statement build
ADD	INSERTCOLNAME, VALUES	not applicable
CLOSE	not applicable	not applicable
DELETE	not applicable	not applicable
INQUIRY	SELECT, INTO	SELECT, INTO, WHERE
REPLACE	not applicable	not applicable
SCAN	not applicable	not applicable
SETINQ	SELECT, INTO	SELECT, INTO, WHERE, ORDERBY
SETUPD	SELECT, INTO, FORUPDATEOF	SELECT, INTO, WHERE, FORUPDATEOF
SQLEXEC	not applicable	with a record and model=UPDATE UPDATE, SET, WHERE with a record and model=DELETE DELETE, WHERE
UPDATE	SELECT, INTO, FORUPDATEOF	SELECT, INTO, WHERE, FORUPDATEOF

EGL: If any SQL clause is modified, all SQL clauses for the SQL statement must be specified. In addition, in EGL a **prepare** statement must specify the entire SQL statement that is required for the SQL PREPARE.

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

Table 40. 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. In addition, if default SQL is used with the Execution time statement build option, the migration tool creates all relevant clauses for that I/O option. 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. In addition, if default SQL is used with the Execution time statement build option, the migration tool creates all relevant clauses for that I/O option. 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.

Table 40. SQL I/O and missing SQL clauses (continued)

Migrating with the associated part	Migrating without the associated part
Missing tables clause: The migration tool creates the tables clause by listing all the SQL tables and table labels from the record. The migration tool includes both SQL table names and table name host variables in the same order that they appear in the VAGen record definition.	Missing tables clause: The migration tool sets the SQL table name to EZE_UNKNOWN_SQLTABLE, sets the table label to T1 and issues an error message.
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_INT0 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 FOR UPDATE OF 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.
<p>Missing SET clause: For modified SQL, there is only one clause (the SET clause) other than the tables clause. If the SET clause is missing from a REPLACE I/O option, the I/O option uses default SQL. In this situation, the migration tool does not create a SET clause.</p> <p>If Execution time statement build is specified with default SQL and a record is specified and the model option is set to UPDATE, the migration tool creates the SET clause by listing the SQL column names and setting each column to the value of the corresponding item from the record. The migration tool lists the SQL column and item name pairs in the same order that the items appear in the record. The migration tool omits the SQL column and item name pair for any item that is included in the EGL keyItems property or any item that is identified as read only.</p>	<p>Missing SET clause: For modified SQL, the migration tool does the same thing as mentioned in the <i>Migrating with the associated part</i> column.</p> <p>If Execution time statement build is specified with default SQL and a record is specified and the model option is set to UPDATE, the migration tool sets the SET clause to EZE_UNKNOWN_SQL_SET and issues an error message.</p>

Table 40. SQL I/O and missing SQL clauses (continued)

Migrating with the associated part	Migrating without the associated part
<p>Missing WHERE clause: For modified SQL, the WHERE clause is not required. In this situation, the migration tool does not create a WHERE clause.</p> <p>If Execution time statement build is specified with default SQL, the migration tool creates a WHERE clause based on the I/O option in the following way:</p> <ul style="list-style-type: none"> For a SETINQ or SETUPD, the migration tool creates a WHERE clause in the following way: <ul style="list-style-type: none"> If there is a default select condition and no key items or more than one key item, the migration tool creates the WHERE clause from the default select condition. If there is no default select condition, but there is one key item, the migration tool creates the WHERE clause using the key item. If there is a default select condition and one key item, the migration tool creates the WHERE clause using both the default select condition and the key item. In all other cases, the migration tool omits the WHERE clause For an INQUIRY, UPDATE, or SQLEXEC with the model option set to UPDATE or DELETE, the migration tool creates a WHERE clause in the following way: <ul style="list-style-type: none"> If there is a default select condition and no key items, the migration tool creates the WHERE clause from the default select condition. If there is no default select condition, but there are one or more key items, the migration tool creates the WHERE clause using all key items. If there is a default select condition and one or more key items, the migration tool creates the WHERE clause using both the default select condition and all the key items. In all other cases, the migration tool omits the WHERE clause. 	<p>Missing WHERE clause: For modified SQL, the migration tool does the same thing as mentioned in the <i>Migrating with the associated part</i> column.</p> <p>If Execution time statement build is specified with default SQL and the I/O option is SETINQ, SETUPD, INQUIRY, UPDATE, or SQLEXEC with the model option set to UPDATE or DELETE, the migration tool sets the WHERE clause to EZE_UNKNOWN_SQL_WHERE and issues an error message.</p>
<p>Missing ORDER BY clause: For modified SQL, the ORDER BY clause is not required. In this situation, the migration tool does not create a ORDER BY clause.</p> <p>If Execution time statement build is specified with default SQL, the migration tool creates an ORDER BY clause based on the I/O option in the following way:</p> <ul style="list-style-type: none"> For a SETINQ, the migration tool creates an ORDER BY clause by listing the field position that corresponds to each item in the EGL keyItems property and then including the ASC option. The first field in the record is considered position 1. For an INQUIRY, UPDATE, SETUPD, or SQLEXEC, the migration tool does not create an ORDER BY clause. 	<p>Missing ORDER BY clause: For modified SQL, the migration tool does the same thing as mentioned in the <i>Migrating with the associated part</i> column.</p> <p>If Execution time statement build is specified with default SQL and the I/O option is SETINQ, the migration tool sets the SET clause to EZE_UNKNOWN_SQL_ORDERBY and issues an error message.</p>

Table 40. SQL I/O and missing SQL clauses (continued)

Migrating with the associated part	Migrating without the associated part
<p>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.</p> <p>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.</p> <p>Disadvantage: This has the potential to ripple back into functions that use this I/O function.</p>	<p>Potential Problem 1: A problem arises for any modified SQL statement or any SQL statement that uses Execution time statement build.</p> <p>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 <i>Migrating with the associated part</i> column.</p> <p>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.</p> <p>Potential Problem 2: The same potential problem and solution as shown for <i>Migrating with the associated part</i> apply.</p>

SQL I/O and Execution time statement build

VisualAge Generator: For an SQL INQUIRY, UPDATE, SETINQ, or SETUPD function when the Execution time statement build option is specified, VisualAge Generator creates an SQL PREPARE statement, followed by an OPEN, and then (for INQUIRY or UPDATE) followed by a FETCH. If a soft error occurs on the SQL PREPARE statement, processing continues with the SQL OPEN or GET. In addition, for an SQLEXEC function when Execution time statement build option is specified, VisualAge Generator creates an SQL EXECUTE IMMEDIATE statement.

EGL: An EGL **prepare** statement follows the normal I/O process. If a soft error occurs, control passes to the **onException** block if one exists. This means that if a soft error occurs on an SQL PREPARE statement, processing does not automatically continue with the corresponding SQL OPEN, GET or EXECUTE statement. In addition, EGL does not support the SQL EXECUTE IMMEDIATE statement. The EGL **prepare** statement, followed by an **execute** statement, provides the closest equivalent.

Associated part needed for migration: Not applicable.

Table 41. SQL I/O and !itemColumnName

Migrating with the associated part	Migrating without the associated part
<p>For an SQL INQUIRY, UPDATE, SETINQ, or SETUPD function with the Execution time statement build option and an I/O error routine, the migration tool converts the I/O option to the following code to provide handling for a soft error on the EGL prepare statement:</p> <pre> try prepareStatement; end if (recordName not HardIOError) try openOrGetStatement; onException IOErrorRoutineEquivalent; end else IOErrorRoutineEquivalent; end </pre> <p>If the function does not have an I/O error routine, the migration tool converts the I/O option to the following code and issues a warning message:</p> <pre> prepareStatement; openOrGetStatement; </pre>	<p>The migration tool does the same thing as mentioned in the <i>Migrating with the associated part</i> column.</p>
<p>For an SQLEXEC function with the Execution time statement build option, the migration tool converts the I/O option to an EGL prepare statement followed by an EGL execute statement. The migration tool issues a warning message.</p>	<p>The migration tool does the same thing as mentioned in the <i>Migrating with the associated part</i> column.</p>
<p>Potential Problem 1: A problem arises if there is an INQUIRY, UPDATE, SETINQ, or SETUPD function that specifies the Execution time statement build option and which does not specify an I/O error routine. The migration tool issues a warning message.</p> <p>Possible Solution 1: If you need processing to continue after a hard error on the EGL prepare statement, edit the function and add the logic shown previously.</p> <p>Potential Problem 2: A problem might arise if there is an SQLEXEC function that specifies the Execution time statement build option because the performance of the prepare statement followed by the execute statement in EGL might not be the same as the SQL EXECUTE IMMEDIATE statement in VisualAge Generator. The migration tool issues a warning message.</p> <p>Possible Solution 2: None. Generate and test the program to check that the performance is acceptable.</p>	<p>Potential Problem: The same potential problems and possible solutions as listed in the <i>Migrating with the associated part</i> column.</p>

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 42. 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 things: <ul style="list-style-type: none">• Uses !itemColumnNames for any column name variables.• Issues an error message that the function needs to be reviewed.
<p>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.</p> <p>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.</p> <p>Disadvantage: This has the potential to ripple back into functions that use this I/O function.</p>	<p>Potential Problem 1: A problem arises for any modified SQL statement or any SQL statement that uses the Execution time statement build option.</p> <p>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.</p> <p>Potential Problem 2: The same potential problem and solution as shown for <i>Migrating with the associated part</i> apply.</p>

SQL I/O with multiple UPDATE or SETUPD functions

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 forUpdate** or **open forUpdate** 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 43. SQL I/O with multiple UPDATE or SETUPD functions

Migrating with the associated part	Migrating without the associated part
<p>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.</p> <p>For SQL, the migration tool does the following things:</p> <ul style="list-style-type: none"> • 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 resultSetID is created using the corresponding UPDATE or SETUPD function name and a customer-specified suffix. <p>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.</p>	<p>When migrating an UPDATE function, if the record is not available, the migration tool does the following things:</p> <ul style="list-style-type: none"> • 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 Declare cursor with hold. • If the migration tool can determine that this UPDATE function 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. <p>When migrating a SETUPD function, the migration tool always includes the resultSetID because SETUPD is only valid for SQL.</p> <p>When migrating a REPLACE function, the migration tool includes the resultSetID if the function specifies a corresponding UPDATE or SETUPD function name.</p>
<p>Potential Problem: None.</p>	<p>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 forUpdate statements. In this case, each replace statement includes a resultSetID, but the get statement that was migrated for the VAGen UPDATE function does not include the resultSetID. Generation for the program fails.</p> <p>Solution: Modify the function to include the resultSetID for the get statement.</p>

DL/I I/O and comparison value items

VisualAge Generator: For DL/I I/O, if the comparison value item is not qualified, VisualAge Generator gives precedence to the record that corresponds to the segment specified for the current Segment Search Argument (SSA). If the comparison value item is not found in that record, the name resolution rules varied over time. In general, VisualAge Generator looks next in working storage records and then in other DL/I segment records such as the I/O object. Items in the function local storage or parameter list are ignored. Records in the function local storage or parameter list are considered, but only for items that are uniquely named within the program.

EGL: For DL/I I/O, if the comparison value item is not qualified, EGL follows the normal EGL qualification rules. EGL looks first at items in the function local storage or parameter list; then fields in records in the function local storage, parameter list, or I/O object; and finally all variables in the program.

Associated part needed for migration: DL/I segment record and the record specified as the alternate specification record, if any.

Table 44. DL/I I/O and comparison value items

Migrating with the associated part	Migrating without the associated part
<p>Based on the first migration of this function, if the comparison value item is not qualified, the migration tool looks for the DL/I segment record associated with the current SSA. If the DL/I segment record and its alternate specification record are available, the migration tool checks the record for the comparison value item in the following way:</p> <ul style="list-style-type: none"> • If the item is in the record, the migration tool qualifies the comparison value item with the DL/I segment record name. • If the item is not in the record, the migration tool does the following things: <ul style="list-style-type: none"> – Uses EZE_UNKNOWN_QUALIFIER as the record name. – Issues a message indicating that it cannot determine the qualification for the item. 	<p>If the DL/I segment record or its alternate specification record are not available, the migration tool does the following things:</p> <ul style="list-style-type: none"> • Uses EZE_UNKNOWN_QUALIFIER as the record name. • Issues a message indicating that it cannot determine the qualification for the item.
<p>Potential Problem: A problem only arises if the comparison value item is not qualified and is not in the associated DL/I segment record or its alternate specification record.</p> <p>Possible Solution: Review your program logic to determine the correct qualification to use. You can also review the generated COBOL source code from the last time you generated the program. In VisualAge Generator, at some points in time, the rules for the qualification of the comparison value item varied. Due to these variations, do not regenerate the program using your current release of VisualAge Generator unless you are certain that the release has not changed since the last time you generated the program.</p>	<p>Potential Problem 1: A problem arises for any unqualified comparison value item.</p> <p>Solution: Modify the DL/I I/O function to include the correct qualification for the comparison value item. Be sure to check the DL/I segment record associated with the qualification statement first.</p> <p>Potential Problem 2: The same potential problem and solution as shown for <i>Migrating with the associated part</i> apply.</p>

Handling ambiguous situations for other statements

Implicit data items in statements

VisualAge Generator: VisualAge Generator permits 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). However, the use of implicit data items is not generally considered to be a good practice.

EGL: EGL does not permit implicit items.

Associated part needed for migration: Not applicable.

Table 45. Implicit data items in statements

Migrating with the associated part	Migrating without the associated part
See “Implicit data items in programs” on page 88	See “Implicit data items in programs” on page 88

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 primary working storage record.

Table 46. Level 77 items in statements

Migrating with the associated part	Migrating without the associated part
See “Level 77 items in records” on page 71	See “Level 77 items in records” on page 71

Table references in statements

VisualAge Generator: If a function references a table, the table is not considered to be an associate of the function.

EGL: If a function references a DataTable, the file containing the function must include an **import** statement for the package containing the DataTable.

Associated part needed for migration: Table.

Table 47. Table references in statements

Migrating with the associated part	Migrating without the associated part
If the table is available, the migration tool adds the table as an associate of the function during Stage 2 migration. Stage 3 migration then adds the corresponding import statement to the file containing the function.	If the table is not available, the migration tool does not add the table as an associate of the function. The import statement is not added during Stage 3.
Potential Problem: None.	Potential Problem: A problem arises if the import statement is not present due to the need to import a part in the same package as the table, either for the function or for some other part in the same file as the function. Solution: Add the import statement to the file containing the function.

MOVEA with a single row table as the source

VisualAge Generator: When a table with a single row of contents is used as the source of a MOVEA statement, the source is treated as a scalar and the target array is completely initialized by the scalar source. This is contrary to the VisualAge Generator documentation, which indicates that the table should always be treated as an array, which in turn would cause only the first element of the target array to be initialized.

EGL: A **move** with the **for** modifier is always treated as a move of one array to another. When a table with a single row of contents is used as the source of a **move** with the **for** modifier, only the first element of the target array is initialized.

Associated part needed for migration: Table.

Table 48. MOVEA with a single row table as the source

Migrating with the associated part	Migrating without the associated part
When migrating a table, if the table contents only has a single row, the migration tool issues a warning message.	When migrating a table, the migration tool does the same thing as described in the <i>Migrating with the associated part</i> column.

Table 48. MOVEA with a single row table as the source (continued)

Migrating with the associated part	Migrating without the associated part
<p>When migrating a statement, if the source for the MOVEA statement is qualified and the qualifier is available, the migration tool checks to determine if the qualifier is a table with only a single row of contents. If so, the migration tool issues an error message.</p> <p>If the source for the MOVEA is not qualified or the qualifier is not a table, the migration tool does not issue a message.</p>	<p>When migrating a statement, if the source for the MOVEA statement is qualified, the qualifier name is 7 or fewer characters, and the qualifier is not available, the migration tool issues a warning message.</p>
<p>Potential Problem 1: A problem arises for tables with a single row of contents. The migration tool issues an error message.</p> <p>Possible Solution 1: If you need the entire target array to be initialized, modify the program logic to use a loop to initialize the elements of the target array from the single row of the source table.</p> <p>Potential Problem 2: A problem also arises if the source is not qualified and resolves to a field in a table that only has a single row of contents. In this case, the program does <i>not</i> run the same as in VisualAge Generator.</p> <p>Possible Solution 2: The solution is the same as for Potential Problem 1.</p>	<p>Potential Problem: The same potential problems and possible solutions as listed in the <i>Migrating with the associated part</i> column.</p>

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 forms. The **move byName** statement is required for a "move corresponding" of records and forms. You can use the **move** statement with the **withV60Compat** modifier to cause a move corresponding if the source or target is a record or form, or an assignment statement if the source or target is a field.

Associated part needed for migration: Not applicable.

Table 49. Assignment statements

Migrating with the associated part	Migrating without the associated part
<p>To preserve as much common code as possible, the migration tool does the following things if both the source and target of an assignment or move statement are unqualified, unsubscripted names:</p> <ul style="list-style-type: none"> • Checks the function 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 in the following way: <ul style="list-style-type: none"> – To an assignment statement if the source or target is an item. – To a move byName 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 with the withV60Compat modifier. 	<p>This is handled the same as mentioned in the <i>Migrating with the associated part</i> column.</p>
<p>Potential Problem: None. EGL debug and generation convert the move statement with the withV60Compat modifier to a VAGen MOVE statement. This is an item to item move or a move byName (move corresponding), depending on the actual source and target of the move. Any program can use the function without modifying it.</p>	<p>Potential Problem: None. The same situation mentioned in the <i>Migrating with the associated part</i> column applies.</p>

FIND statement

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

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

Associated part needed for migration: The VAGen table.

Table 50. FIND statement

Migrating with the associated part	Migrating without the associated part
<p>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.</p>	<p>If the search column is not explicitly specified and the table is not available, the migration tool does the following things:</p> <ul style="list-style-type: none"> • Sets the search-column to EZE_UNKNOWN_SEARCH_COLUMN • Issues an error message stating that the function must be modified to use the proper column name.

Table 50. FIND statement (continued)

Migrating with the associated part	Migrating without the associated part
<p>Potential Problem: A problem only arises if two DataTables, probably in different subsystems, have the same DataTable name, but different search column names.</p> <p>Solution: For the second subsystem, add a field as a substructure for the first column in the DataTable. The name of this new field 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.</p>	<p>Potential Problem 1: The search column name must be provided. EGL validation displays an error message in the Problems view.</p> <p>Solution: Edit the function and specify the correct column name for the DataTable.</p> <p>Potential Problem 2: The same potential problem and solution as shown for <i>Migrating with the associated part</i> apply.</p>

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 VAGen table. The return column defaults to the second column.

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

Associated part needed for migration: The VAGen table.

Table 51. RETR statement

Migrating with the associated part	Migrating without the associated part
<p>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 information:</p> <ul style="list-style-type: none"> 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. 	<p>If the search column or return column is not explicitly specified and the table is not available, the migration tool does the following things:</p> <ul style="list-style-type: none"> Sets the search column to EZE_UNKNOWN_SEARCH_COLUMN Sets the return column to EZE_UNKNOWN_RETURN_COLUMN Issues an error message stating that the function must be modified to use the proper column names.
<p>Potential Problem: A problem only arises if two DataTables, probably in different subsystems, have the same DataTable name, but different search or return column names.</p> <p>Solution: For the second subsystem, add a field as a substructure for the first column in the DataTable. The name of this new field should be the same as the search column in the first subsystem. Substructure the second column of the DataTable 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.</p>	<p>Potential Problem 1: The search and return column names must be provided. EGL validation displays an error message in the Problems view for each missing column.</p> <p>Solution: Edit the function and specify the correct column names for the DataTable.</p> <p>Potential Problem 2: The same potential problem and solution as shown for <i>Migrating with the associated part</i> apply.</p>

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:

- **converseLib.clearScreen()** for text (display) forms
- **converseLib.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 main map group for the program.

Table 52. SET map PAGE statement

Migrating with the associated part	Migrating without the associated part
<p>Based on the first migration of this function, if the map is available, the migration tool converts SET <i>map</i> PAGE to one of the following functions:</p> <ul style="list-style-type: none"> • converseLib.clearScreen() for a text form • converseLib.pageEject() for a print form <p>The migration tool also includes a comment with the original map name.</p>	<p>If the map is not available, the migration tool does the following things:</p> <ul style="list-style-type: none"> • Converts SET <i>map</i> PAGE to converseLib.EZE_SETPAGE(). • Includes a comment with the original map name. • Issues an error message that it was unable to determine the map type.
<p>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. EGL validation does not display an error message in the Problems view. Generation does not fail for the program.</p> <p>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 example:</p> <pre>if (sysVar.systemType is ZOSBATCH) converseLib.pageEject(); else converseLib.clearScreen(); end</pre> <p>Similar logic can be used based on transaction code, user ID, and so on, depending on the specific details of your system.</p>	<p>Potential Problem 1: If the function containing the statement is used in a program, EGL validation displays an error message in the Problems view. If the function is not used in a program, there is no message in the Problems view.</p> <p>Solution: Edit the function and change EZE_SETPAGE() to either converseLib.clearScreen() or converseLib.pageEject(), depending on the map type.</p> <p>Potential Problem 2: The same potential problem and solution as shown for <i>Migrating with the associated part</i> apply.</p>

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 53. SET mapItem attributes

Migrating with the associated part	Migrating without the associated part
<p>When migrating a printer map, the migration tool omits attributes that are not supported by EGL for print forms.</p> <p>When migrating a function, the migration tool migrates the SET statement without regard to whether the map is a display map or printer map.</p>	<p>The migration tool does the same thing as mentioned in the <i>Migrating with the associated part</i> column.</p>
<p>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. EGL validation displays an error message in the Problems view.</p> <p>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.</p> <p>Disadvantage: This has the potential to ripple back into functions that use the function with the set statement.</p>	<p>Potential Problem: The same potential problem and solution as listed in the <i>Migrating with the associated part</i> column apply.</p>

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 54. Checking for IN literal or scalar

Migrating with the associated part	Migrating without the associated part
<p>For an IF or WHILE statement that checks a data item IN a literal, the migration tool does the following things to match the VAGen behavior:</p> <ul style="list-style-type: none"> • 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. <p>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).</p>	<p>The migration tool does the same thing as mentioned in the <i>Migrating with the associated part</i> column.</p>

Table 54. Checking for IN literal or scalar (continued)

Migrating with the associated part	Migrating without the associated part
<p>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, EGL validation displays an error message in the Problems view.</p> <p>Solution: Modify the function to initialize <code>sysVar.arrayIndex</code> to 0 before the if or while statement and to set <code>sysVar.arrayIndex</code> to 1 immediately after the if or while statement. Also change the if or while statement to compare using <code>==</code> rather than <code>in</code>.</p>	<p>Potential Problem: The same potential problem and solution as listed in the <i>Migrating with the associated part column</i> apply.</p>

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**. Form fields 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 55. Checking SQL and map items for NULL

Migrating with the associated part	Migrating without the associated part
<p>Based on the first migration of this function, if the item is qualified, the migration tool does the following things:</p> <ul style="list-style-type: none"> 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. 	<p>The migration tool tries to determine the type of the item in the following way:</p> <ul style="list-style-type: none"> If the item is qualified and the qualifier is not available, the migration tool does the following things: <ul style="list-style-type: none"> Checks if the qualifier is also the I/O object of the function. 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 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 one of the following values: <ul style="list-style-type: none"> 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 things: <ul style="list-style-type: none"> Converts to EZE_NULL. Issues an error message indicating that this statement should be reviewed.

Table 55. Checking SQL and map items for NULL (continued)

Migrating with the associated part	Migrating without the associated part
<p>Based on the first migration of this function, if the item is not qualified, the migration tool does the following things:</p> <ul style="list-style-type: none"> Checks the function 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 associated parts 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. 	<p>If the item is not qualified, the migration tool checks the function parameter list to see if the item is specified there as either an SQLITEM or a MAPITEM.</p> <p>If the migration tool can determine that the item is in an SQL record or on a map, the tool migrates to one of the following values:</p> <ul style="list-style-type: none"> null for an SQL record blanks for a map item <p>If the migration tool cannot determine that the item is in an SQL record or on a map, then the tool does the following things:</p> <ul style="list-style-type: none"> Converts to EZE_NULL. Issues an error message indicating that this statement should be reviewed.
<p>Potential Problem: None.</p>	<p>Potential Problem 1: A problem arises if the migration tool uses EZE_NULL. EGL validation displays an error message in the Problems view.</p> <p>Solution: Edit the function and change EZE_NULL to null for an SQL item or blanks for a form variable field.</p>

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 in the following circumstances:

- any soft error
- any hard error if EZEFECC = 1
- for DL/I I/O, any hard error if EZEDLERR or EZEFECC = 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 in any of the following circumstances:

- any soft error
- any hard error if **vgVar.handleHardIOErrors** is set to 1
- for DL/I I/O, any hard error if **dliVar.handleHardDLIErrors** or **vgVar.handleHardIOErrors** is set to 1

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

Table 56. I/O error values UNQ and DUP

Migrating with the associated part	Migrating without the associated part
<p>Based on the first migration of this function, if the record is available, the migration tool does the following things:</p> <ul style="list-style-type: none"> • 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. 	<p>If the record is not available, the migration tool tries to determine the type of the record in the following way:</p> <ul style="list-style-type: none"> • If the statement specifies the same record as the I/O object for the function, 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, Declare 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 cases, the migration tool cannot determine the record type: <ul style="list-style-type: none"> – 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. <p>In the previous situations, and in other situations when the migration tool cannot determine the record type, the migration tool does the following things:</p> <ul style="list-style-type: none"> – Converts UNQ to unique. – Converts DUP to EZE_DUPLICATE and issues an error message.

Table 56. I/O error values UNQ and DUP (continued)

Migrating with the associated part	Migrating without the associated part
<p>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, and the function is used with an SQL record and checks for duplicate, then EGL validation displays an error message. If the SQL record is available when the function is migrated, then the additional information conveyed by the duplicate check is not available for the non-SQL record.</p> <p>Possible Solution: Copy the function and use the original function for SQL and the new function for non-SQL.</p> <p>Disadvantage: This has the potential to ripple back into functions that use the original function that checked for UNQ or DUP.</p> <p>Potential Problem for SQL: None. DUP and UNQ were always set the same way and unique continues to be a hard error.</p> <p>Potential Problem 1 for non-SQL: A problem arises if you do <i>not</i> set vgVar,handleHardIOErrors (EZEFEFEC) to 1 for the program. In this case, because unique is now a hard error, the onException statement does not get control and the program ends.</p> <p>Solution: Make sure your programs set vgVar,handleHardIOErrors to 1.</p> <p>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 tests true in EGL in some cases, even though it did not test true in the past on VisualAge Generator. Validation and generation do not detect an error. However, the program might <i>not</i> run as it did in VisualAge Generator.</p> <p>Possible Solution: You might need to reorder the testing of the I/O error values in your program logic.</p>	<p>Potential Problem 1: EZE_DUPLICATE is not valid in EGL.</p> <p>Solution: Edit the function and change EZE_DUPLICATE to duplicate or unique based on the record type.</p> <p>Other Potential Problems: The same potential problems and solutions as shown for <i>Migrating with the associated part</i> apply.</p>

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 in the following circumstances:

- any soft error
- any hard error if EZEFEFEC = 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 in the following circumstances:

- any soft error

- any hard error if `vgVar.handleHardIOErrors` is set 1

Associated part needed for migration: Not applicable.

Table 57. 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 <i>Migrating with the associated part</i> column.
<p>Potential Problem 1: A problem arises if you do <i>not</i> set <code>vgVar.handleHardIOErrors</code> (EZEFECE) = 1 for the program. In this case, because deadlock is a hard error, the <code>onException</code> statement does not get control and the program ends.</p> <p>Solution: Make sure your programs set <code>vgVar.handleHardIOErrors</code> to 1.</p> <p>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 tests true in EGL in some cases where it did not test true in VisualAge Generator. Validation and generation do not detect an error. However, the program might <i>not</i> run as it did in VisualAge Generator.</p> <p>Possible Solution: You might need to reorder the testing of the I/O error values in your program logic.</p>	The same potential problems as in the <i>Migrating with the associated part</i> column can occur. You can use the same solutions.

Handling ambiguous situations for EZE words

For some EZE word replacements, the migration tool must declare an extra item variable. The migration tool adds these new item variables to the program. This permits the variables to be used by any function in the program.

EZELTERM

VisualAge Generator: EZELTERM is the conversation ID in a Web Transaction program and the terminal ID in all other program types.

EGL: `sysVar.conversationID` is the conversation ID in a VGWebTransaction program. `sysVar.terminalID` is the terminal ID in all other program types. `sysVar.conversationID` and `sysVar.terminalID` are treated as synonyms so either provides the correct information based on the program type.

Associated part needed for migration: The program.

Table 58. EZELTERM

Migrating with the associated part	Migrating without the associated part
<p>Based on the first migration of this function, if the program is available, the migration tool converts EZELTERM based on the program type in the following way:</p> <ul style="list-style-type: none"> • If the program is a Web Transaction program, the migration tool uses <code>sysVar.conversationID</code> • Otherwise, the migration tool uses <code>sysVar.terminalID</code> 	If the program is not available, the migration tool always converts EZELTERM to <code>sysVar.terminalID</code> .

Table 58. EZELTERM (continued)

Migrating with the associated part	Migrating without the associated part
Potential Problem: None. <code>sysVar.conversationID</code> and <code>sysVar.terminalID</code> are treated as synonyms.	Potential Problem: None. <code>sysVar.conversationID</code> and <code>sysVar.terminalID</code> are treated as synonyms.

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 `sysVar.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 `vgLib.getVGSysType` provides the old VAGen values.

Associated part needed for migration: Not applicable.

Table 59. EZESYS

Migrating with the associated part	Migrating without the associated part
<p>When migrating any program, if the Do not initialize old EZESYS values migration preference is not selected, the migration tool does the following things:</p> <ul style="list-style-type: none"> Includes a declaration for <code>custPrefixEZESYS</code>. Includes an initialization statement to set the value of <code>custPrefixEZESYS</code> to the old VAGen EZESYS value. <p>If the preference is selected, the migration tool does not include the declaration or initialization statement.</p> <p><code>custPrefix</code> is the same prefix that is used for changing part names that conflict with reserved words. Use the VAGen Migration Preferences to set the value of <code>custPrefix</code>.</p>	<p>The migration tool does the same thing as mentioned in the <i>Migrating with the associated part</i> column.</p>

Table 59. EZESYS (continued)

Migrating with the associated part	Migrating without the associated part
<p>Based on the first migration of the function, the migration tool does the following things:</p> <ul style="list-style-type: none"> • If EZESYS is used in an IF, WHILE, or TEST statement, the migration tool converts EZESYS to <code>sysVar.systemType</code> <p>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 MVS BATCH to the EGL equivalent ZOS BATCH. The migration tool migrates OS2 and NTCICS to the same value as in VisualAge Generator. See Table 123 on page 332 for specifics of which values are converted.</p> <ul style="list-style-type: none"> • If EZESYS is used in any other statement, the migration tool does the following things: <ul style="list-style-type: none"> – Issues a warning message that this use results in the old VAGen EZESYS values – Uses <code>custPrefixEZESYS</code> <p>to replace EZESYS in the statement.</p>	<p>The migration tool does the same thing as mentioned in the <i>Migrating with the associated part</i> column.</p>
<p>Potential Problem 1: A problem arises if you select the Do not initialize old EZESYS values migration preference and you use EZESYS in statements other than IF, WHILE, or TEST. In this situation the migration tool uses <code>custPrefixEZESYS</code> in the statement, but programs do not have a declaration and initialization statement for <code>custPrefixEZESYS</code>. EGL validation displays an error message in the Problems view.</p> <p>Potential Solution 1A: Change your EGL logic to use the new values for <code>sysVar.systemType</code> .</p> <p>Potential Solution 1B: Add a declaration and an initialization statement for <code>custPrefixEZESYS</code> to any program that needs to use the old VAGen value for EZESYS.</p> <p>Potential Problem 2: A problem arises if you want to use the new EGL values in statements other than if and while.</p> <p>Possible Solution 2: Modify the function and change the logic to use <code>sysVar.systemType</code> instead of <code>custPrefixEZESYS</code></p> <p>Be sure to change the old VAGen values to the new EGL values in any Data Tables, files, or databases that you use for comparisons.</p>	<p>The same potential problems mentioned in the <i>Migrating with the associated part</i> column apply. You can use the same solutions.</p>

EZEWAIT

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

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

Associated part needed for migration: Not applicable.

Table 60. EZEWAIT

Migrating with the associated part	Migrating without the associated part
When migrating any program, the migration tool always includes a declaration for <code>custPrefixEZE_WAIT_TIME</code> .	The migration tool does the same thing as mentioned in the <i>Migrating with the associated part</i> 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 <code>custPrefixEZE_WAIT_TIME</code> .	The migration tool does the same thing as mentioned in the <i>Migrating with the associated part</i> column.
Potential Problem: None. However, if you use the function in a new program, be sure to include a declaration for <code>custPrefixEZE_WAIT_TIME</code> in the program.	The same potential problem mentioned in the <i>Migrating with the associated part</i> 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, follow these steps:

1. Upgrade to VisualAge Generator 4.5 with Fix Pack 5. Also review Appendix F, “APARs required for VisualAge Generator,” on page 457 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. Shut down VisualAge for Java.
4. Run the self-extracting file VAGenMigJava.exe, which is in the following directory:

`\installationDirectory\bin`

Note: If you installed and kept a previous version of the 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.

5. 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 **Unzip**.

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

- \ide\vgmigration\checkStage1.bat
- \ide\vgmigration\checkStage1.sql
- \ide\vgmigration\createdatabase.sql
- \ide\vgmigration\createindex.sql
- \ide\vgmigration\createtables.sql
- \ide\vgmigration\deletemigsets.bat
- \ide\vgmigration\MigPreferences.xml
- \ide\vgmigration\runStats.bat
- \ide\vgmigration\SetupDatabase.bat
- \ide\vgmigration\SetupIndex.bat
- \ide\vgmigration\SetupTables.bat
- \ide\vgmigration\VGMigReserved.txt
- \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, using the following steps:

1. Start VisualAge Generator on Java.
2. Add the **IBM VisualAge Generator EGL Migration** feature by following these steps:
 - 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 is loaded.
 - d. Click the **Projects** tab in the Workbench. You should see the **IBM VisualAge Generator EGL Migration** project in your workspace.

Note: If you have problems adding the migration feature and are using a remote repository, try either of the following techniques:

- Copy the following directory:

`VJavaInstallDirectory\ide\features\com-ibm-vgj-mig`

to the features directory on the machine with the remote repository. Then try the **Add Feature** step as described previously. This is the preferred technique if several developers need to load the Stage 1 migration tool.

- Make sure you have the **IBM VisualAge Generator Utilities** project at V4.5 FP5 loaded in your workspace. Then follow these steps:
 1. From the Workbench window, click **File -> Import**.
 2. Click **Repository** and then click **Next**.
 3. Click **Local repository** and point to the .dat file in the following directory:

`VJavaInstallDirectory\ide\features\com-ibm-vgj-mig`
 4. Click **Projects** and then click **Details**. Select the only project/version that is in this .dat file and then click **OK**. Select the **Add most recent project edition to workspace** checkbox. Click **Finish**.

The **IBM VisualAge Generator EGL Migration** project should be added to your workspace.

Creating the migration database

See “Creating the DB2 migration database” on page 459 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 might also want to copy the MigPreferences.xml file to a directory outside the VisualAge for Java installation directory and make your modifications in the copy. This avoids accidentally overwriting your modifications if you install a new version of the migration tool.

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, follow these steps:

1. Start VisualAge Generator for Java.
2. In the Workbench window, click the **Projects** tab.
3. Navigate to the **IBM VisualAge Generator EGL Migration** project. Expand the migration project and then expand the **com.ibm.vgj.mig** package.
4. Within the package, select the **PreferencesUI** class.
5. Right-click the **PreferencesUI** class and then click **Properties**.
6. Click the **Program** tab.
7. On the Program page, set 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.

8. Click **OK** to save the properties.
9. Right-click the **PreferencesUI** class and then click **Run -> Run main**. (Or you can click 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:

- If you do not currently use Project List Parts (PLPs), see “Migration plans and high-level PLP projects” on page 139.
- 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 page
- Mapping page
- Renaming page
- Execution page

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** section 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 138 for information on setting the *-o* (override) option for the **VAGenToEGLMigration** class, which is the actual Stage 1 migration tool.

Plan directory

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 in the following way:

- If you include the *-o* option in the properties for the **VAGenToEGLMigration** class, the Stage 1 migration tool creates new plan files 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 files for you based on your repository filters and high-level PLP projects. If you need assistance creating a PLP project, see “Creating a high-level PLP project” on page 140.

- 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 uses the existing plan files, 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 141 for details about creating your own migration plan files.

Repository Filters

This 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 in the following way:

- 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 in the following way:
 - 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, click 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 processes the project version in the following way:
 - 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, but no migration set is specifically created for this project version.

Specify the **Repository Filters** information in the following way:

- **Projects** filter. The migration tool matches the project names in your repository to the **Projects** filter that you specify. You can specify multiple **Projects** filters. To add or remove filters, use the **Add** and **Remove** buttons. To update a filter, overtype in the table. The filters are not case sensitive. You can use wildcards in the following way:
 - 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 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** buttons. To update a filter, overwrite in the table. The filters are not case sensitive. You can use wildcards in the following way:
 - 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 any 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 names of some of the EGL projects, packages, and files that are created during migration.

File Names

This 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 packages. See "Placing parts in EGL files" on page 42 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

This 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*. 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 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*. The migration tool concatenates the suffix to the project name after any **Renaming Rules** are applied.

Common Identifiers

This 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.

Projects

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 is either placed in a program file or in the file specified by your **Common Parts** preference. The part is not 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** 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.

Packages

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 is either placed in a program file or in the file specified by your **Common Parts** preference. The part is not 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** 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** buttons. To update a renaming rule, overwrite the contents of the cells in the table. You can double-click 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 following values are available:

front The rule applies if the from string appears at the beginning of a project, package, or version name.

back The rule applies if the from string appears at the end of a project, package, or version name.

any The rule applies if the from string appears anywhere within a project, package, or version name.

token 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 following values are available:

project

The renaming rule only applies to VAGen project names.

package

The renaming rule only applies to VAGen package names.

both The renaming rule applies to both VAGen project names and VAGen package names.

version

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 in the following example:

- Step 1 -- Clear both **Generate report** and **Update database**. This enables you to review the migration plan files that are created, and to ensure that your **Repository Filters** are set correctly to process the project versions that you want. If you are not satisfied with the project versions that are selected, you can refine your **Repository Filters** and run this step again.
- Step 2 -- Select **Update database**, with or without **Generate report**.

Note:

- Updating the database 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.
- You must update the database before the report can be run.
- Generating the report can also take some time. You might prefer to run some simple queries to see the EGL file names rather than generating the report. For sample queries that produce the EGL file names, see “Reviewing the EGL file names” on page 464.
- If you generate the report, the report files are overwritten. 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 causes 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 be in one of the following formats:

- `jdbc:DB2:databaseName` if you are using a local database or a locally catalogued remote database.

Note: *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

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

The user ID needed to connect to the migration database. If you do not specify the user ID, 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

The password needed to connect to the migration database. If you do 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 details:

Trace level

Enables you to specify the level of information that you want to write to the log and debug files. Use the drop-down list to specify one of the following values:

Fatal 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 process the .pln file again.

Warning

Warning messages, as well as fatal error messages, are logged.

Informational

Informational messages, as well as warning and fatal error messages, are logged.

Debug

Debug information, as well as informational, warning, and fatal error messages, are logged. This is the only trace level that causes the migration tool to write information to the debug file. This is the default value.

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 file:

```
<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 = "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>*</fromString>
        <toString>_</toString>
        <stringContext>any</stringContext>
        <mappingContext>version</mappingContext>
    </renameRule>
    <renameRule order = "508">
        <fromString>&lt;</fromString>
        <toString>_</toString>
        <stringContext>any</stringContext>
        <mappingContext>version</mappingContext>
    </renameRule>
    <renameRule order = "509">
        <fromString>&gt;</fromString>
        <toString>_</toString>
        <stringContext>any</stringContext>
        <mappingContext>version</mappingContext>
    </renameRule>
    <renameRule order = "510">
        <fromString>&quot;</fromString>
        <toString>_</toString>

```

```

        <stringContext>any</stringContext>
        <mappingContext>version</mappingContext>
    </renameRule>
    <renameRule order = "511">
        <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, you might need to perform the following tasks:

- Customize the Stage 1 migration tool
- Specify your character set
- Take steps to improve the performance of the Stage 1 migration tool
- Save your workspace

Customizing the Stage 1 migration tool

The Stage 1 migration tool on Java has the following built-in customizations that you can enable if they are appropriate to your environment:

- Move all data item parts to a new EGL project called ItemsProject. The items are placed in packages that correspond to their original package name, with the suffix .items (for example, my.original.pkg.items). This customization is particularly useful if you plan to select the Stage 2 migration preference **Convert shared data items to primitive item definitions**. By enabling this customization in Stage 1, you can move all the data items parts to a single project during migration and then remove that project from your EGL workspace.
- Split the common parts files by part type and, optionally, by part name. This customization can reduce the size of the common parts files and make it easier to locate specific parts.

In addition to the built-in customizations, there is a white paper that describes how to consolidate projects and packages during Stage 1 on Java. For information on obtaining this white paper, see “References” on page 16. Depending on your VAGen project and package naming conventions, the white paper technique can be easier to use than the **Renaming rules** preference described in “Renaming page” on page 129.

Enabling the Stage 1 built-in customizations

To enable the customizations, follow these steps:

1. Expand the **IBM VisualAge Generator EGL Migration** project. Expand the **com.ibm.vgj.mig.db** package. Expand the **MigrationSetUtility** class.
2. Edit the **customizedPlacementScriptSet()** method.
3. To enable the customization that moves all the data item parts to a new EGL project,
 - a. Change the line:


```
boolean consolidate = false;
```

to

```
boolean consolidate = true;
```

By default, the Stage 1 tool sets the EGL file name to "Items", concatenated with the first character of the data item name.
 - b. If you want to change the number of characters from the data item name that are concatenated to the file name, change the value of **suffixLength** in the **customizedPlacementScriptSet()** method.
 - c. If you want to change the name of the EGL Project or the suffix for the packages, modify the **consolidateDataItemsQuery()** method.
4. To enable the customization that splits the common parts files by part type and part name:
 - a. Change the line:


```
boolean redistribute = false;
```

to

```
boolean redistribute = true;
```
 - b. The following lines specify the number of characters from the part name that you want to append to the name of the common parts files:


```
int suffixLength = 1;
suffixLength = 2;
```

By default, the value of **suffixLength** is set to use the first character from a data item part name and the first 2 characters from record and function names. However, you can change the value for **suffixLength**. For example:

 - If you set this number to 0, the common parts files are named **CommonRecords.egl**, **CommonFunctions.egl**, and **CommonItems.egl**.
 - If you set this number to a digit, *n*, the first *n* characters of the part name are used as a suffix for the file name. If you set *n* to 1, then the names are **CommonRecordsA**, **CommonRecordsB**, and so on.
 - c. If you want the names of the files to be different, change the **redistributeCommonRecords(int)**, **redistributeCommonFunctions(int)**, or **redistributeCommonItems(int)** methods.
5. Save your changes and then version and release the class. You might also want to version the package and project so that you have a record of your changes.

Specifying your character set information

The Stage 1 migration tool specifies a character set based on your VAGen national language code. For single-byte languages this is "iso-8859-1"; other values are used for double-byte languages. If you need to use a different character set, you can change this value by following these steps:

1. Expand the **IBM VisualAge Generator EGL Migration** project. Expand the **com.ibm.vgj.mig** package. Expand the **Preferences** class.

2. Edit the **determineXMLEncoding()** method.
3. Change the value that is specified for `xmlCharset` to the character set that you need. For example, you can change the last 4 lines of the method:

```
if (vgNLS.equals("PTB") )
    xmlCharset = "iso-8859-1";
return xmlCharset;
}
```

Add the line shown in bold:

```
if (vgNLS.equals("PTB") )
    xmlCharset = "iso-8859-1";
xmlCharset = "my required character set";    // new line with value you need
return xmlCharset;
}
```

4. Save your changes and then version and release the class. You might also want to version the package and project so that you have a record of your changes.

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, follow these steps:

1. Shut down VisualAge Generator.
2. See “Saving your workspace” on page 138 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/FixPack5/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 follow these steps 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, restore your workspace using the following steps:

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 **com.ibm.vgj.mig** package.
3. Within the package, select the **VAGenToEGLMigration** class.
4. Right-click the **VAGenToEGLMigration** class and then click **Properties**.
5. Click the **Program** tab.
6. On the Program page, set the **Command line arguments** field to point the MigPreferences.xml file you want to use:

Table 61. Valid command line options for VAGenToEGLMigration class

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

7. If this is the first time you are running the Stage 1 tool, follow these steps:
 - a. In the same Properties window, click the **Class Path** tab.

- b. On the Class Path page, select the **Extra directories path** check box and then click the **Edit** button for the Extra directories.
 - c. Click **Add Jar/Zip**.
 - 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 **Compute Now** and then click **Yes** at the prompt.
8. Click **OK** to save the properties.
 9. Right-click the **VAGenToEGLMigration** and then click **Run -> Run main**. (Or you can click the running man icon on 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.

Depending on the number of parts and the complexity of the associations between parts, some steps of the Stage 1 migration tool can take a long time. The following steps in the Console window can be particularly time-consuming (over an hour) without noticeable activity:

- Executing partPlacementQuery 3
- Executing partPlacementQuery 4
- Executing setPartFilePathQuery 3
- Starting Migration Report Generation

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. If you select **Update database** again and a migration set with the same name already exists in the database, the Stage 1 migration tool automatically deletes the old information about the migration set from the database and then adds the new information for the migration set. Therefore, there is no need for you to clean up a migration set from the database. However, it can be much faster to use the techniques described in “Resetting the migration database for Stage 1” on page 460.

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 EGL development environment. See Chapter 6, “Stage 2—Conversion to EGL syntax,” on page 171 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 **Repository Filter** 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 use one of the following techniques:

- Create a high-level PLP project to use for migration using the tool provided by VisualAge Generator. This high-level PLP project needs to specify the list of project versions that you want to migrate as a group. Using the tool provided by VisualAge Generator ensures that the project and version names are spelled correctly and have the correct case. After you create the high-level PLP project, you can use the Stage 1 migration tool to create the migration plan.
- 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 141 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 follow these steps in VisualAge Generator:

1. From the Workbench window, click the **Projects** tab.
2. Create a new Java project to contain the Project List Part. Be sure to give the project a different name than that of any of your existing projects. For example, create a project called MySubsystem1.
3. Right-click the new project and then click **Manage -> Configure VAGen Required Projects**.
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 by following these steps:
 - a. Delete the high-level PLP project and any other VAGen projects from your workspace.

- b. Click **Selected -> Add -> Project**.
- c. From the Add Project window, follow these steps:
 - 1) Click **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, click **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. Note that the Stage 1 migration tool stops if there are any duplicate parts in the migration set.
- f. You might also want to validate your programs and tables to ensure that they are valid in VisualAge Generator and that you are not missing any parts.

You can chain PLP projects. For example, you might create a PLP project that lists the project versions for all your common projects. Then, for each subsystem, you could 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 the high-level PLP project for every subsystem.

When you are ready to run the Stage 1 migration tool, follow these steps:

1. 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.
2. 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` 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>
    .
    .
  </migrationSet>
```

```

        <project name="projectNameZ" version="projectNameZVersion1"></project>
    </migrationSet>
</migrationDefinition>

```

The following discussion applies to the previous example:

- *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 in the following ways:
 - 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 to 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 list a *projectName* only 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.

To ensure the migration set is valid, test each migration set in your migration plan by following these steps:

1. Delete all your VAGen projects from your workspace.
2. Manually add the project versions specified in the migration set to your workspace. Be sure to add the version of each project that you specified in the migration set. Note that the Stage 1 migration tool stops if a specified project version or package version is not available in the repository.
3. From the VAGen Parts Browser, click **Tools -> Show Duplicate Parts**. There should not be any parts on the list. If there are, you need to change your migration set definition so that there are no duplicates. Note that the Stage 1 migration tool stops if there are any duplicate parts in the migration set.

4. You might also want to validate your programs and tables to ensure that they are valid in VisualAge Generator and that you are not missing any parts.
5. The Stage 1 migration tool requires that the projects be versioned. Run a Management Query to determine whether there are any open or scratch editions of projects or packages. To run a Management Query, follow these steps:
 - a. From the Workbench window, click **Workspace -> Management Query**.
 - b. From the Management Query window, follow these steps:
 - 1) In the **Program element** section, select **Projects** and **Packages**. Be sure that **Types** is cleared.
 - 2) In the **Status** section, click **Scratched**.
 - 3) In the **Scope** section, click **Workspace**.
 - 4) In the **Owners** section, click **Any User**.
 - 5) Click the **Start Query** button (the last button on the tool bar). There should not be any scratch editions. If there are scratch editions, create open editions of the projects and packages. If you are the owner, you can create the open editions from the Status pane.
 - 6) Change the **Program element** section to select **Projects**, **Packages**, and **Types**.
 - 7) In the **Status** section, click **Open Edition**.
 - 8) Click the **Start Query** button.
 - 9) There should not be any open editions other than for the **IBM VisualAge Generator EGL Migration** project. If there are open editions, version the projects, packages, or types. If you are the owner, you can version them from the Status pane.

When you are ready to run the Stage 1 migration tool, follow these steps:

1. When you set your Stage 1 preferences, on the Build plans page, 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**.
2. 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, follow these steps:

1. Upgrade to VisualAge Generator 4.5 with FixPack 4 and FixPack 5. Also review Appendix F, “APARs required for VisualAge Generator,” on page 457 for additional VisualAge Generator APARs that might be necessary for your specific situation.

Note: An early version of Fix Pack 5 for VAGen on Smalltalk was not cumulative. Check the readme file to ensure that the version of Fix Pack 5 you are using is cumulative. If necessary, download the Fix Pack again or contact IBM Support.

2. On your system, determine where VisualAge Smalltalk is installed.
3. Shut down VisualAge Smalltalk.
4. Run the self-extracting VAGenMigST.exe file. The file is in the following directory:

`\installationDirectory\bin`

Note: If you installed and kept a previous version of the 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.

5. When the GUI prompt appears, navigate to the drive and directory where VisualAge Smalltalk is installed. Then click **Unzip**.

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

- feature\vgMigST.ctl
- image\Messages.properties
- image\MigPreferences.xml
- image\VGMigReserved.txt
- import\vgMigST.dat
- checkStage1.bat
- checkStage1.sql
- createdatabase.sql
- createindex.sql
- createtables.sql
- deletemigsets.bat
- runStats.bat
- SetupDatabase.bat

- SetupIndex.bat
- SetupTables.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 following these steps:
 - a. From the System Transcript, click **Tools -> Load/Unload Features**.
 - b. On the Selection Required window, follow these steps:
 - 1) Ensure the **Show other features** checkbox is selected.
 - 2) In the **Available features** pane, select **Other: VAGen EGL Migration - versionName**.
 - 3) Click the >> button to move **Other: VAGen EGL Migration - versionName** to the **Loaded features** pane.
 - 4) Click **OK**. The VAGen EGL Migration feature is 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 are prompted to save your image. Click **Yes** so you do not have to load the feature again.

Note:

1. 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.somecompany.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.
2. If you are using a remote library manager, you must run the self-extracting executable on the remote server and extract based on the [Feature Installation] heading and the values specified for installationPath and importDirectory in your abt.ini file.

Creating the migration database

See “Creating the DB2 migration database” on page 459 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. You might also want to copy the MigPreferences.xml file to a directory outside the VisualAge for Smalltalk installation directory and make your modifications in the copy. This avoids accidentally overwriting your modifications if you install a new version of the migration tool.

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. You can start the Stage 1 GUI editor in either of two ways:

- From the System Transcript, click **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 **Open**.
- From the System Transcript, click **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 **Save** or **Save As** (specifying a new file name), and then close the editor.

Note:

- If you do not use currently use configuration maps, see “Migration plans and high-level configuration maps” on page 165.
- 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 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 page
- Mapping page
- Renaming page
- Execution page

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

Identifies where the Stage 1 migration tool is to read or write the migration plan file (or files).

Plan Directory

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 **Plan File Name** to view existing plan files in your plan directory. If you need to see details within a plan file, click **View Plans** and expand the plan file to see the migration sets.

- If you do not specify a **Plan File Name**, the migration tool deletes *all* of 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 of the migration sets.

Repository Filters

This 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 in the following way:

- 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 in the following way:
 - 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. The default **Version Depth** filter is 1.

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 processes the project version in the following way:
 - 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, but no migration set is specifically created for this configuration map version.

Specify the **Repository Filters** information in the following way:

Configuration Maps

The migration tool matches the configuration map names in your library to the **Configuration Maps** filter that you specify. You can specify multiple **Configuration Maps** filters. To add, change, or remove filters, right-click a filter and use the options on the pop-up menu. The filters are not case sensitive. You can use wildcards in the filters in the following way:

- 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

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 a filter and use the options on the pop-up menu. The filters are not case sensitive. You can use wildcards in the filters in the following way:

- 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

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 information:

- EGL file names for common parts and for unused parts.
- Suffixes that are used in building certain 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.

This section 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 each 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 42 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 each 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 new project name is *migrationSetName_ProjectSuffix*. 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 an 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 new package name is *packageName.PackageSuffix*. The migration tool concatenates the suffix after any **Renaming rules** are applied.

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`.

Collapse subapplications

If you select this option, the migration tool converts each VAGen subapplication to an EGL package. If you clear this option, the migration tool converts the subapplication using dot notation. Consider the situation in which there is an application named `MainApp` that contains a subapplication named `SubApp`. If you select **Collapse subapplications**, the migration tool creates two packages—one named `MainApp` and one named `SubApp` so that both packages are at the same level in the EGL folder structure. If you clear **Collapse subapplications**, the migration tool also creates two packages, but they are named `MainApp` and `MainApp.SubApp` so that the two packages appear in a hierarchy in the EGL folder structure. The default is selected.

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 **Use package naming dot notation** and **Convert package names to lowercase**. If both options 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 the field and use the options on the pop-up 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 is placed either in a program file or in the file specified by your **Common Parts** preference. The part is not 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 is placed either in a program file or in the file specified by your **Common Parts** preference. The part is not 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 a rule and use the options on the pop-up 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 following values are available:

front The rule applies if the from string appears at the beginning of a configuration map, application, or version name.

back The rule applies if the from string appears at the end of a configuration map, application, or version name.

any The rule applies if the from string appears anywhere within a configuration map, application, or version name.

token 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 following values are available:

configMap

The renaming rule only applies to VAGen configuration map names.

application

The renaming rule only applies to VAGen application names.

both

The renaming rule applies to both VAGen configuration map names and VAGen application names.

version

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 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

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. This section describes the preferences you can specify on the Execution page in more detail:

Database

This section enables you to specify details about the migration database:

DB

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

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

The user ID needed to connect to the migration database. If you do not specify the user ID, 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

The password needed to connect to the migration database. If you do 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

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 details:

Trace Level

Enables you to specify the level of information that you want to write to the log and debug files. You can select one of the following values:

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 extension. This enables you to process the .pln file again.

WARN

Warning messages and error messages are logged.

INFO Informational, warning, and error messages are logged.

DEBUG

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. This is the default value.

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

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 information:

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.htm 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>
    <collapseSubapplications>true</collapseSubapplications>
    <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>
  </eglMapping>
</preferences>
```

```

</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 62. 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: <ul style="list-style-type: none"> 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, you might need to perform the following tasks:

- Customize the Stage 1 migration tool
- Specify your character set information
- Take steps to improve the performance of the Stage 1 migration tool
- Save your image

Customizing the Stage 1 migration tool

The Stage 1 migration tool on Smalltalk has the following built-in customizations that you can enable if they are appropriate to your environment:

- Move all data item parts to a new EGL project called ItemsProject. The items are placed in packages that correspond to their original Smalltalk application name, with the suffix .items (for example, my.original.app.items). This customization is particularly useful if you plan to select the **Convert shared data items to primitive item definitions** Stage 2 migration preference. By enabling this customization in Stage 1, you can move all the data items parts to a single project during migration and then remove that project from your EGL workspace.
- Split the common parts files by part type and, optionally, by part name. This customization can reduce the size of the common parts files and make it easier to locate specific parts.

In addition to the built-in customizations, there is a white paper that describes how to consolidate EGL projects and packages during Stage 1 on Smalltalk. For information on obtaining this white paper, see “References” on page 16. Depending on your VAGen project and package naming conventions, the white paper technique can be easier to use than the Renaming rules preference described in “Renaming page” on page 154.

Enabling the Stage 1 built-in customizations

To enable the customizations, follow these steps:

1. From the VisualAge Organizer, in the left pane, select the **HptEglMigrationUtilityApp** application. In the center pane, double-click the **HptEglMigrationSetUtility** class.
2. In the Script Editor window, toggle the **class / instance** button so that it is set to **instance**. Toggle the **private / public** button so that it is set to **public**.
3. In the Categories pane, select **PP - Accessors**. In the methods pane, select **customizedPlacementScriptSet**.
4. To enable the customization that moves all the data item parts to a new EGL project, follow these steps:

- a. Change the line:
`consolidate := false.`

to
`consolidate := true.`

By default, the Stage 1 tool sets the EGL file name to "Items", concatenated with the first character of the data item name.

- b. If you want to change the number of characters from the data item name that are concatenated to the file name, change the value of `suffixLength` in the **customizedPlacementScriptSet** method.
 - c. If you want to change the name of the EGL Project name or the suffix for the packages, modify the **consolidateDataItemsQuery** method.
5. To enable the customization that splits the common parts files by part type and part name, follow these steps:

- a. Change the line:
`redistribute := false.`

to
`redistribute := true.`

- b. The following lines specify the number of characters from the part name that you want to append to the name of the common parts files:

```
suffixLength := 2.  
...  
suffixLength := 1.
```

By default, the value of `suffixLength` is set to use the first character from a data item part name and the first 2 characters from record and function names. However, you can change the value for `suffixLength`. For example:

- If you set this number to 0, the common parts files are named **CommonRecords.egl**, **CommonFunctions.egl**, and **CommonItems.egl**.
 - If you set this number to a digit, *n*, the first *n* characters of the part name is used as a suffix for the file name. If you set *n* to 1, then the names are **CommonRecordsA**, **CommonRecordsB** and so on.
- c. If you want the names of the files to be different, change the **redistributeCommonRecordsUsing**, **redistributeCommonFunctionsUsing**, or **redistributeCommonItemsUsing** methods.
6. Save your changes, then version and release the class. You might also want to version the application so that you have a record of your changes.

Specifying your character set information

The Stage 1 migration tool specifies a character set based on your VAGen national language code. For single-byte languages this is "iso-8859-1"; other values are used for double-byte languages. If you need to use a different character set, you can change this value by following these steps:

1. From the VisualAge Organizer, click **Applications -> View -> Show All Applications**.
2. In the left pane, select the application **HptEglMigrationUtilityApp**. In the center pane, double-click the class **EsImPreferences**.
3. In the Script Editor window, toggle the **instance/class** button so that it is set to **instance**. Toggle the **public/private** button so that it is set to **public**.
4. In the Categories pane, select **Reports**. In the methods pane, select **determineXmlEncoding**.
5. Change the value that is specified for `xmlCharset` to the character set that you need. For example, you can change the last 3 lines of the method:

```
(vgNLS = 'PTB')
  ifTrue: [ xmlCharset := 'iso-8859-1' ].
  ^xmlCharset.
```

Add the line in bold:

```
(vgNLS = 'PTB')
  ifTrue: [ xmlCharset := 'iso-8859-1' ].
xmlCharset := 'iso-8859-1'.           "new line with the value you need"
  ^xmlCharset
```

Save the method.

6. Version and release the class. You also might want to version the application so that you have a record of your changes.

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, follow these steps:

1. From the System Transcript, click **Tools -> Open VAGen Tools Workspace**.
2. Under the "Image Management" section, swipe through:
System abtScrubImage.
3. Then right-click and click **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 148 for information on how to load the feature. Alternatively, to add the EGL Migration Tools option back onto the System Transcript tool bar, follow these steps:
 - a. Type the following into the System Transcript window:
`HptEglMigrationGuiApp loaded`
 - b. Swipe through the line you typed, then right-click and click **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 follow these steps 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, restore your workspace using the following steps:

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:
 - If you modified the preferences by starting the Preferences Editor, start the EGL Migration Driver View from the System Transcript by clicking **EGL Migration Tools -> Migration Driver**.
 - 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. Click **Browse** to point to a different preferences file. Click **Edit** 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 in the following way:

Overwrite PLN

Controls the migration plan file or files in the following way:

- If you select **Overwrite PLN**, the Stage 1 migration tool creates new plan files based on 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 the file with the same name from the specified **Plan Directory** before creating a new .pln file.

Select the **Overwrite PLN** option if you want the Stage 1 migration tool to create the migration plan files for you based on your repository filters and high-level configuration maps. If you need assistance creating a configuration map to use for migration, see “Creating a high-level configuration map” on page 166.

- If you clear **Overwrite PLN**, the Stage 1 migration tool does not create any new migration plan files. Instead, the Stage 1 migration tool uses the existing plan files 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.

Clear the **Overwrite PLN** 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. For details about creating your own migration plan files, see “Creating a migration plan file manually” on page 167.

Generate Report

Controls whether the migration tool creates the report specified in the **Verification** section of the preferences file. If you clear 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 are assigned to EGL projects, packages, and files during migration.

Update Database

Controls whether the migration tool updates the migration database with the migration plan information. If you clear this option, the migration database is not updated. If you select this option, the migration database you specified in your preferences is updated with information from the migration plan, including the External Source Format for every VAGen part in the 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.

Depending on the number of parts and the complexity of the associations between parts, some steps of the Stage 1 migration tool can take a long time. The following steps in the System Transcript window can be particularly time-consuming (over an hour) without noticeable activity:

- Executing partPlacementQuery 3
- Executing partPlacementQuery 8
- Executing eglPathBuilderQuery 8
- Starting Migration Report Generation

You might run the Stage 1 migration tool in several steps as follows:

1. Clear both **Generate Report** and **Update Database**. This enables you to review the migration plan files that are created to ensure that your Repository Filters are set correctly to process the configuration map versions that you want. If you are not satisfied with the configuration map versions that are selected, you can refine your Repository Filters and run this step again until you are satisfied with the configuration map versions that the migration tool will process.
2. Select **Update Database**, with or without **Generate Report**.

Note:

- Updating the database 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.
- You must update the database before the report can be run.
- Generating the report can also take some time. You might prefer to run some simple queries to see the EGL file names rather than generating the report. For sample queries that produce the EGL file names, see “Reviewing the EGL file names” on page 464.
- If you generate the report, the report files are overwritten. 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 causes the links to be lost so the files are no longer viewable.
- The **Repair Message Table Associates** option is only used if you need to repair a migration database that was created with a previous version of the Stage 1 migration tool. For details, see “Repairing a migration database created using a previous version of the Stage 1 tool” on page 165.

When the Stage 1 tool finishes, if you 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. If you select **Update Database** again and a migration set with the same name already exists in the database, the Stage 1 migration tool automatically deletes the old information about the migration set from the database and then adds the new information for the migration set. Therefore, there is no need for you to clean up a migration set from the database. However, it can be much faster to use the techniques described in “Resetting the migration database for Stage 1” on page 460.

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 EGL

development environment. See Chapter 6, “Stage 2—Conversion to EGL syntax,” on page 171 for information about continuing your migration process.

Repairing a migration database created using a previous version of the Stage 1 tool

Note: This section only applies if you have a migration database that was created with the Stage 1 tool prior to the tool version of 20071022_1400.

If you use message tables and have a migration database that was created with a version of the Stage 1 tool prior to version 20071022_1400, you might want to update your migration database to include the message tables as associates of the program. Using the repair tool before running the Stage 2 and 3 tools enables those tools to set the EGL Build Path and **import** statements to include the message tables as associates of the program parts. To update an old Stage 1 database, follow these steps:

1. Set your migration preferences as described in “Setting Stage 1 preferences” on page 148. For the purposes of the repair tool, you only need to set the **Database** and **Service** information on the Execution page.
2. From the System Transcript, click **EGL Migration Tools -> Migration Driver**.
3. Ensure the **File Name** for the Migration Preference File points to the file in which you stored your preferences.
4. Select **Repair Message Table Associates**.
5. Click **OK**. The repair tool runs and updates the program associates for any program that specifies a message table prefix to include all message tables that match the prefix.

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 166 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 use one of the following techniques 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 required configuration map versions, or a combination of application versions and required 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 are considered as a group when migrating. See “Creating a high-level configuration map” on page 166 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, follow these steps in VisualAge Generator:

1. From the VisualAge Organizer, click **Options** and be sure that **Full Menus** is selected.
2. From the VisualAge Organizer, click **Tools -> Configuration Maps**.
3. From the Configuration Maps Browser, click **Names -> Create**.
4. In the Information Required window, enter the name of the configuration map and then click **OK**. A new edition of the configuration map is automatically created and selected.
5. Click **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.
6. Version the configuration map by clicking **Editions -> Version**.
7. Select the configuration map version and load it into your image by clicking **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 by following these steps:

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

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

Using configuration maps with the Stage 1 tool

When you are ready to run the Stage 1 migration tool, follow these steps:

1. 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.
2. 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>
```

The following discussion applies to the previous example:

- *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 in the following ways:
 - 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 contains 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 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, follow these steps:

1. When you set your Stage 1 preferences, on the Build plans page, set the **Plan Directory** 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**.
2. When you instruct the Stage 1 tool which outputs to produce, be sure to clear **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 migration steps

The remaining steps of the migration are the same whether you are migrating 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 EGL. 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 DB2 performance information

After you run Stage 1 and before you run Stage 2, you should use the DB2 **runStats** command to evaluate and set performance information for the DB2 tables. To set the performance information, follow these steps:

1. From a DB2 Command Window, navigate to the directory where runStats.bat is located.
 - For Java, this is *VisualAge-for-Java-install-directory\ide\vgmigration*
 - For Smalltalk, this is *VisualAge-Smalltalk-install-directory*
2. If you changed the default migration database name (VGMIG) or the default schema name (MIGSCHEMA), change the runStats.bat file to use your database name and schema name.
3. Run runStats.bat.

You might also want to backup the migration database. This enables you to restore the database to the end of Stage 1 if you want to try different migration preferences or options for Stage 2. For details, see “Backing up and restoring the migration database” on page 466.

Setting your workbench preferences

Before you start to migrate you should set your workbench preferences, as described in the following sections:

- Start up parameters
- Required EGL preferences
- Suggested preferences
- VAGen Migration Preferences
- Other suggested settings

Start up parameters

To improve the performance of the EGL development environment, you might need to increase several start up parameters in the initialization file. The parameters are the same regardless of the product that you use. The initialization file is always in the product installation directory, but the name of the file varies with the product that you are using. For example, if you are using Rational Business Developer, the initialization file is named eclipse.ini. To set the start up parameters, follow these steps:

1. Using a text editor, edit the initialization file.
2. Change the following parameters:

-Xms256m
-Xmx1024m

Setting -Xms increases the amount of memory that is used when starting the product. Set this to a value that is less than or equal to your setting for -Xmx.

Setting -Xmx increases the available memory. Generally, set this to a value that is less than your real memory. For example, if your real memory is 2000K, then set -Xmx to 1024. This helps avoid the use of virtual memory.

3. Save the initialization file.
4. Start the EGL development environment. For example, if you are using Rational Business Developer, start that product.

Required EGL preferences

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

1. From the Workbench window, click **Window -> Preferences**.
2. Expand the **General** preference by selecting the + beside it and then select **Capabilities**.
3. Click **Advanced**.
4. In the Advanced Capabilities Settings window, expand **EGL Developer**.
5. Select **EGL VAGen Migration**. Depending on the types of applications you plan to migrate, you might also need to select the following capabilities:
 - EGL DLI
 - EGL MQ
 - EGL Text UI
 - EGL VG Web Transactions
6. Click **OK** to close the Advanced Capabilities Settings window.
7. Click **OK** to close the Preferences window.

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, follow these steps:

1. From the Workbench window, click **Window -> Preferences**.
2. Select **EGL**.
3. Select the **VisualAge Generator compatibility** preference.
4. Click **OK**.
5. When you are prompted for a full-rebuild of the workspace, click **OK**.

If you plan to migrate VAGen Web transaction programs, you should set the type of EGL Web project. To set this preference, follow these steps:

1. From the Workbench window, click **Window -> Preferences**.
2. Select **EGL**.
3. In the Default EGL Web Project Facet Choices section, follow these steps:
 - a. Clear **EGL support with JSF** and **EGL support with JSF Component Interfaces**.
 - b. Select **EGL support with Legacy Web Transaction**.
4. In the **Code migration's default web runtime** section, select the Web Application Server that you plan to use. Select **New** and use the wizard to specify any additional options you need for the server. These options are used in Stage 3 of migration to create a valid EGL Web project.

5. Click **OK**.

If you plan to migrate any VAGen programs that use direct MQ API calls, you should select the MQ API support preference by following these steps:

1. From the Workbench window, click **Window -> Preferences**.
2. Select **EGL**.
3. In the Default EGL Project Features Choices section, select **EGL with low-level MQ API support**.
4. Click **OK**.

If you do not have any immediate plans to create reports using EGL, you should clear the report-related preferences by following these steps:

1. From the Workbench window, click **Window -> Preferences**.
2. Select **EGL**.
3. In the Default EGL Project Features Choices section, clear the following options:
 - **EGL with BIRT report support**
 - **EGL with Jasper report support**
4. Click **OK**.

If you do not have any immediate plans to create services when you migrate your VAGen source code, you might want to clear the EGL deployment descriptor preferences in the following way:

1. From the Workbench window, click **Window -> Preferences**.
2. In the Default EGL Project Features Choices section, clear the following options:
 - **Create an EGL service deployment descriptor**
3. Click **OK**.

If your VAGen control parts use non-English special characters, you might want to set the encoding preference by following these steps:

1. From the Workbench window, click **Window -> Preferences**.
2. Select **EGL**.
3. In the Creating .eglbld Files section, use the drop-down list to select a code page for the **Encoding** preference.
4. Click **OK**.

Suggested preferences

To set other preferences, click **Window -> Preferences** from the Workbench window, and then select the appropriate preferences page. The following preferences might be useful when you are resolving EGL validation messages in the Problems view:

- **EGL -> Editor**. Select **Show line numbers**.
- **General -> Editors -> Text Editors**. Select **Show line numbers**.
- **General -> Workspace**. Decide whether to select or clear **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 clear this option, EGL does not rebuild anything when you save a file. The advantage of clearing 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 clear **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 disables **Build automatically**.

- **General -> Workspace -> 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 **Maximum entries per file** based on your backup requirements.
- **General -> Perspectives.** You might want to set either the EGL perspective or the Web perspective as your default perspective. Use the EGL perspective if you do not plan to develop Web applications. Use the Web perspective if you plan to migrate Web transactions or develop Web applications. To set the default perspective, select the perspective you want to use and then click **Make Default**.

VAGen Migration preferences

The preferences described in the following sections control the overall migration process. Unless otherwise noted, these preferences are used both for Stage 2 during Stage 1 - 3 migration and for single file migration. To set these preferences, click **Window -> Preferences -> VAGen Migration** from the Workbench window.

Single File Mode Preferences:

- **Separate parts into EGL files.** This preference is only used during single file migration. If you select this preference, each program, map group, and table is placed in its own file; other parts are placed in the file you specify on the Import VAGen External Source Format File wizard. This adheres to the EGL requirement of one generatable part per file. If you clear this preference, all the parts are placed in the EGL file you specify on the Import VAGen External Source Format File wizard. For specifics of the parts placement algorithm for single file mode, see "Overview of single file migration" on page 27.

Rename User Exit Information:

- **Rename user exit.** The VAGen migration tool provides simple renaming for data items, records, functions, and maps based on adding a prefix to the part name or part reference. Optionally, you can write a user exit routine to provide more complex renaming. For example, you might want to change a hyphen (-) to an underscore (_) during migration. Select the **Rename user exit** preference if you are providing a user exit routine to rename parts. If you select this option, you must provide additional information about your **Rename user exit** routine. See the white paper "Using the Rename User Exit in the VisualAge Generator to EGL Migration Tool" for details of how to create a **Rename user exit**, as well as sample code.
- **JAR file location.** Specify the location on your system of the .jar file that contains your **Rename user exit** routine.
- **Package name.** Specify the name of the package within the .jar file that contains your **Rename user exit** routine.
- **Class name.** Specify the name of the class within the package that contains your renaming logic. This class must contain the method `renameUserExit(String s, Connection c)`.
- **Use a database.** Select this option if your **Rename user exit** uses a database to provide the relationship between the old part name and the new part name.

Minimize VisualAge Generator Compatibility Mode: This group of preferences enables you to minimize the automatic use of migration techniques that require VisualAge Generator Compatibility mode. Use caution when selecting these preferences because they can result in more error messages in the EGL Problems view or changes in runtime behavior.

- **Do not initialize old EZESYS values.** If you select this preference, the migration tool does not include a declaration and initialization statement in each program for an extra variable to contain the old VAGen values for EZESYS. The migration tool still uses the extra variable when it converts statements involving EZESYS other than IF, WHILE, or TEST. If you select this preference and you have statements involving EZESYS other than IF, WHILE, or TEST, EGL validation displays an error message in the Problems view after migration. Consider selecting this preference if you plan to convert all statements to use the new EGL **sysVar.systemType** values or if you know that you did not use EZESYS in statements other than other than IF, WHILE, or TEST. See “EZESYS” on page 118 for additional details.
- **Do not include deleteAfterUse for tables.** If you select this preference, the migration tool always omits the **deleteAfterUse** property for the DataTable **use** declaration statements in each program. The migration tool issues a warning message for those situations in which the **deleteAfterUse** property is omitted. Consider setting this preference if you are migrating directly from VisualAge Generator Version 4.5 Fix Pack 4 or higher and all your production programs are generated with that version and Fix Pack. If you are migrating from Cross System Product or earlier releases of VisualAge Generator and you select this preference, be sure to thoroughly test any program for which the **deleteAfterUse** property is omitted from the **use** declaration for a DataTable.
- **Do not honor evensql=y for items or variables.** If you select this preference, the migration tool always uses odd precision (or 18 if the item is the maximum length) when migrating a PACK data item part or nonshared item in a record. The migration tool issues a warning message for those situations in which the VAGen item specified even precision (evensql=y). Consider setting this preference if you are certain that your SQL tables do not use even precision for columns that you might reference in an SQL WHERE clause or in an EGL **prepare** statement. Alternatively, you can select this preference, migrate, and then review all the item definitions for which the migration tool issued a warning message. Using a precision other than what is specified in the SQL table definition can result in poor performance for database access.
- **Do not set compatibility mode.** If you select this preference, the migration tool always omits the **vagCompatibility** = "YES" build descriptor option whenever the tool converts a generation options part. You should select this preference only if you follow all of these steps:
 1. Select all of the other three preferences in the "Minimize VisualAge Generator Compatibility Mode" section.
 2. Ensure that all of your migrated part names adhere to the EGL part naming conventions when VisualAge Generator compatibility mode is disabled. You can specify a **Rename user exit** and code the exit to rename the VAGen parts during migration to achieve this.
 3. Disable the EGL preference for VisualAge Generator compatibility after you finish migration. **Note:** Regardless of how you set the preferences, the migration tool always turns on VisualAge Generator compatibility mode when refreshing the workspace.

VAGen Migration Database I/O Preferences

The preferences described in the following section control the migration of the VAGen syntax to EGL syntax for SQL and DL/I database I/O. Unless otherwise noted, these preferences are used both for Stage 2 during a Stage 1 - 3 migration and for single file migration. To set these preferences, from the Workbench window, click **Window -> Preferences -> VAGen Migration -> VAGen Migration Database I/O Preferences**.

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 appending 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 is MY-SETUPD_RSI01. You can set the Result Set suffix to any value other than blank. However, be sure to choose something that does 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 appending 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 other than blank. However, be sure to choose something that does not cause conflicts with any of your part names.
- **Omit column name.** In VisualAge Generator, the SQL column name is always specified for a field in an SQL record. In EGL, you can omit the SQL column name if it is the same as the field 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 field name. Omitting the column name can make your EGL source code less cluttered.
- **Omit isSQLNullable property.** In VisualAge Generator, the null indicator variable is always included for every field in the SQL record. To preserve the VisualAge Generator behavior, the migration tool always includes the **isSQLNullable** property for every field 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 the **Omit isSQLNullable property**, the migration tool omits the **isSQLNullable** property from every field in SQL records. Omitting the **isSQLNullable** property can improve runtime performance and reduce the risk of exceeding the DB2 precompiler limits. However, you should only select the **Omit isSQLNullable 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 clear **Omit isSQLNullable property** during migration. After migration, you can remove the **isSQLNullable** property for selected fields 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 field in an SQL record. Read Only must always be yes if there are multiple tables specified for the SQL record. By default, the

migration tool includes the **isReadOnly** property for every field in an SQL record that references multiple SQL tables. However, the EGL **isReadOnly** property defaults to NO if there is only one SQL table and to YES if there are multiple SQL tables specified for the SQL record. If you select the **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.

DL/I preferences:

- **Database PCB suffix.** In VisualAge Generator, the same database name can be used multiple times in a PSB. In EGL, the PSB is a record. The database name becomes a field name and must be unique. The migration tool creates the field name in the PSB from the database name, a number (if necessary for uniqueness), and the Database PCB suffix. This avoids conflicts between the database name and any other names in the program. Consider the situation in which the database name COURSE is used for two different PCBs and COURSE is also the name of a DL/I segment record. If you use the default Database PCB suffix of _dbPCB, then the field name created for the first COURSE PCB is COURSE_dbPCB. The field name created for the second COURSE PCB is COURSE_n_dbPCB, where *n* is the number of the PCB in the VAGen PSB. The name of the DL/I segment is still COURSE. You can set the Database PCB suffix to any value other than blank. However, be sure to choose something that does not cause conflicts with any of your part names.
- **GSAM PCB suffix.** The GSAM PCB suffix is used for the GSAM PCBs similar to the way the Database PCB suffix is used for database PCBs. You can set the GSAM PCB suffix to any value that you want other than blank. However, be sure to choose something that does not cause conflicts with any of your part names.

VAGen Migration Syntax Preferences

The preferences described in the following section control the migration of the VAGen syntax to the EGL syntax. Unless otherwise noted, these preferences are used both for Stage 2 during a Stage 1 – 3 migration and for single file migration. To set these preferences, from the Workbench window, click **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 on the migration tool extended reserved word list or starts with the # or @ symbol. The tool adds the Renaming prefix to the beginning of 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 other than blank, EZE, or a value that starts with the # or @ symbol. Be sure to choose something that does 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 by appending the

Level77 suffix to the original working storage record name. 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 is named `MYRECORD_Level77Items`. You can set the Level77 suffix to any value other than blank. However, be sure to choose something that does 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 FormGroups for a program have unique names. The migration tool renames maps in the help map 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` as `MAP1_helpmap`. You can set the Help Map suffix to any value other than blank. However, be sure to choose something that does not cause conflicts with any of your part names.

Other Conversion Options:

- **Convert shared data items to primitive item definitions.** 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 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. **Warning:** Only the primitive definition is included in the record. Therefore, clear this preference if you use shared items in a UI record because the UI edit customizations for the shared item are not copied to the EGL VGUI record.
- **Change decimal comma to decimal point.** The migration tool automatically converts numeric literals to use the point if your workstation specifies a locale that typically uses a comma to indicate decimal positions. For example, the migration tool automatically converts the comma to a point for French and German locales. However, some customers specify English as their locale and override the normal decimal point setting to be a comma. If you use this technique, you must select the **Change decimal comma to decimal point** preference. This ensures that the migration tool converts the comma to a point in numeric literals. EGL only supports the point as the decimal position indicator for internal storage.
- **Use VAGen logical operators (AND and OR).** If you select this preference, the migration tool uses "and" and "or" as the logical operators in EGL **if** and **while** statements. If you clear this preference, the migration tool uses `&&` and `||` as the logical operators. Set this preference based on the coding standards you plan to use when you write new EGL programs.
- **Enclose CALL and DXFR program names in quotes.** If you select this preference, the migration tool encloses the program name in quotes for every EGL **call** or **transfer to program** statement. If you clear this preference, the migration tool does not enclose the program name in quotes. Regardless of the preference setting, the migration tool does the following things:
 - Never encloses `sysVar.transferName` (EZEAPP) in quotes.
 - Always encloses the program name in quotes if the `isExternal` property is set to yes (NONCSP). This includes the use of `EZCHART`.

The advantages of selecting this preference are:

- Any **call** and **transfer to program** statements that refer to a program that is not in your workspace are not marked as errors, so you can generate the program.
- If there is one (and only one) program with this name in your workspace, you can debug the calling or transferring program.

The disadvantages of selecting this preference are that you do not get an error message so there is no indication that you need to provide a linkage options part to provide the package information if you generate for Java.

- **Enclose XFER program names in quotes.** If you select this preference, the migration tool encloses the program name in quotes for an EGL **transfer to transaction** or **show** statement. If you clear this preference, the migration tool does not enclose the program name in quotes. Regardless of the preference setting, the migration tool provides the following special treatment:
 - Never encloses **sysVar.transferName** (EZEAPP) in quotes.
 - Always encloses the program name in quotes if the **isExternal** property is set to yes (NONCSP).

The advantages of selecting this preference are:

- Any **transfer to transaction** and **show** statements that refer to a program that is not in your workspace are not marked as errors.
- For runtime environments in which the transfer-to name is a program, if there is one (and only one) program with this name in your workspace, you can debug the transferring program. For transactional runtime environments (for example, CICS or IMS/VS) in which the transfer-to name is a transaction name, you can debug as you would in VisualAge Generator by using the program name at debug time and the transaction name at runtime.

The disadvantages of selecting this preference are that you do not get an error message so there is no indication that you need to provide a linkage options part to provide the package information if you generate for Java.

Other suggested settings

You might want to consider the following settings:

- Optionally, close the Welcome view.
- If you are not already using the EGL perspective, switch to it by clicking **Window -> Open Perspective -> Other -> EGL -> OK**. Alternatively, if you plan to migrate VAGen Web transaction programs or develop new EGL JSFHandlers, 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 clicking **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, click **Window -> Show View -> Other**. From the Show View window, expand **General** and then select **Navigator**.

Note: Some activities such as displaying error markers or deleting an EGL package are not fully supported in the Navigator view. Always use the Project Explorer view if you are restructuring your projects or packages.

- In the Problems view, click the Menu button (a downward-facing triangle) in the upper right corner and then click **Configure Contents**. In the Configure Contents window, click the **On selected element only** radio button. This limits the error messages in the Problems view to the messages for the currently selected file. When there are numerous 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.

- In the Problems view, click the Menu button and then click **Preferences**. In the Preferences window, clear the **Use marker limits** option. Clearing this option enables you to see all the messages in the Problems view
- When you have multiple files open for editing, you can configure the Project Explorer or 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 Project Explorer or Navigator view. To do this, click the **Link Open Editors with content in the Navigator** icon on the tool bar of the Project Explorer or Navigator 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 create the Stage 2 migration preferences, follow these steps:

1. Start the EGL development environment. Be sure to set your workbench preferences as explained in the section “Setting your workbench preferences” on page 171.
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 picks it up automatically by following these steps:
 - a. Under **Window->Preferences**, click **Java->Build Path->Classpath Variables**.
 - b. Click **New**.
 - c. For **Name**, enter: **DB2_DRIVER_PATH**
 - d. For **Path**, click **File** 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. Optionally, create a simple project that can contain your Stage 2 preferences file if you choose to save it. This is required if you want to run Stage 2 in batch mode. It is also useful to have a record of your settings if you run Stage 2 in online mode. Follow these steps to create a simple project:
 - a. Start the New Project wizard by clicking **File->New->Project**. Expand **General**, select **Project**, and then click **Next**.
 - b. Give the project a name, such as VAGENMIG. Click **Finish**.
4. Right-click your project and then click **New->Other**.
5. Expand **VAGen Migration to EGL** and then select **VAGen Migration File**. Click **Next**.
6. 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 63. 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 Choose File to select an existing .vgmig file. Click Load File 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.	<code>path_to_db2java.zip\db2java.zip</code>
Database driver	This is the name of your DB2 driver.	This value must always be <code>COM.ibm.db2.jdbc.app.DB2Driver</code> . 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: <ul style="list-style-type: none"> <code>jdbc:DB2:databaseName</code> <i>databaseName</i> 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 of the log file for the Stage 2 messages.	Enter a valid location (drive and directory) in the file system.
Log file name	This is the name of the log file for the Stage 2 messages.	Enter a valid file name.

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

Table 64. 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 JavaSource folders.	If you plan to generate COBOL only, click COBOL . If you might generate Java, click Java .
Migrate remaining VAGen parts	This determines whether or not parts that are not referenced by any generatable part in the migration set are converted to EGL. Note: For the purposes of the Migrate remaining VAGen parts preference, UI records are treated like other records. This preference does not consider UI records to be generatable parts.	Select the checkbox to convert unreferenced parts to EGL. Generally, you should select Migrate remaining parts . If you clear Migrate remaining parts , control parts and any other parts that are unused within the migration set are not migrated to EGL source. For more information on the impact of selecting this option, see “Overwriting and merging files” on page 48.

Table 64. Preferences to enter on second page of wizard (continued)

Preference	Meaning	Value
Import into workspace	<p>This determines whether or not Stage 3 (importing EGL into files in the current workspace) is automatically started after Stage 2 is complete.</p> <p>Note: If you select this checkbox, you must click one of the radio buttons under this checkbox to specify the version to import (latest or oldest; see the next rows in this table), because only one version of a project can be in the workspace at a time.</p>	<p>Select this checkbox to import EGL files directly after the conversion of parts to EGL. Clear this checkbox to import files later, during Stage 3.</p> <p>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.</p> <p>For more information on the impact of selecting this option, see “Overwriting and merging files” on page 48.</p>
Latest version or Oldest version	This option specifies which version of the desired migration sets should be imported.	If the Import into workspace option is selected, you must click one of these 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 are overwritten.	<p>This option can only be selected if the Import into workspace option is selected. The Override existing files option 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 the Override existing files option, the Stage 3 migration tool replaces the existing file and includes <i>only</i> those parts that are in the current migration set. If you clear the Override existing files option, the Stage 3 migration tool merges any new parts into the existing file. The new parts are placed alphabetically by part type.</p> <p>For more information on the impact of selecting this option, see “Overwriting and merging files” on page 48.</p>

Table 64. 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.	<p>If you plan to migrate multiple versions of a migration set, then follow these steps:</p> <ol style="list-style-type: none"> 1. Select this checkbox so that each version can be written to a different subdirectory. 2. Specify the Folder under which the subdirectories for the versions are placed. 3. Clear the Migrate now option. 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. 4. Select the Save current configuration to file option. 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 the Save migrated files to temporary directory option, 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 the Migrate now option to run Stage 2 in online mode or the Save current configuration to file option 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. Clear 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	<p>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:</p> <ul style="list-style-type: none"> • In online mode, right-click the saved .vgmig file and click Start Migration. • In batch mode, use the -importFile option to specify the saved .vgmig file. For details, see “Running Stage 2 in batch mode” on page 185. 	<p>You must select either the Migrate now option to run Stage 2 in online mode or the Save current configuration to file option 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.</p> <p>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.</p>
Path	This specifies the project into which the file should be saved.	<code>\projectName</code> , where <i>projectName</i> is the name of the project that you want to contain your saved file.

Table 64. Preferences to enter on second page of wizard (continued)

Preference	Meaning	Value
File name	This specifies the name of the file to which preferences are saved.	<i>fileName</i> , where <i>fileName</i> is the desired name for your file <i>without</i> 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 options that you specify 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 **Finish**.
- If you select **Migrate now**, Stage 2 migration runs after you click **Finish**.
- If you also select either **Import into workspace** or **Save migrated files to temporary directory**, Stage 3 starts automatically after Stage 2 completes.

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

```
databaseDriverLocation=d:\SQLLIB\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 EGL development environment.

Running Stage 2 from the user interface

The wizard described in “Setting up the Stage 2 VAGen migration file” on page 180 provides the option to save your preferences in a .vgmig file. If you select the **Migrate now** option in the wizard, then Stage 2 migration starts when you click **Finish** in the wizard. If you cleared **Migrate now** in the wizard, follow these steps when you are ready to run Stage 2 migration using your saved .vgmig file:

1. From the Project Explorer or Navigator view, expand the project containing the Stage 2 preferences file by clicking the + symbol next to the project name.
2. Right-click the .vgmig preferences file and then click **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 starts automatically 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. The migration tool creates one "to do" list file for each migration set version that it processes. The "to do" list file names are in the form: *TODO_migrationSetName_migrationSetVersion*.
- If you selected the **Import into workspace** option, 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 the **Save migrated files to temporary directory** option, 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 the projects for each migration set version in a separate subdirectory under the temporary directory that you specified. The subdirectory names are in the form: *migrationSetName_migrationSetVersion*. 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. If you want to use batch mode, you need to consider the following restrictions:

- Regardless of your EGL preference settings, the .egldd file is not created.
- You can direct the EGL projects, packages, and files to either the current workspace or a temporary directory. If you specify both a current workspace and a temporary directory, the migration tool ignores the current workspace.
- If you want to write to temporary directories, you must create a directory and workspace structure similar to the following scheme:

```
temporaryDirectoryName << temporaryDirectoryName can be any name
stub                   << "stub" is the required workspace name
  projectName         << projectName can be any name
    xxxx.vgmig        << xxxx can be any name
```

Set up the stub workspace in the following way:

- Specify your EGL and VAGen migration preferences as described in “Setting your workbench preferences” on page 171.

- Set the **tempDirectory** option in the *xxxx.vgmig* file to point to the *temporaryDirectoryName* in which the stub workspace is located. During Stage 3, the migration tool creates one workspace for each migration set version under *temporaryDirectoryName*. The migration tool copies *everything* (capabilities, preferences, projects, and so on) from the stub workspace to each of the new workspaces.
- Set the **-data** option in the batch command file to point to *temporaryDirectoryName\stub*.

To run in batch mode, follow these steps:

1. Follow the steps described in “Setting up the Stage 2 VAGen migration file” on page 180, with the following exceptions:
 - 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 clear **Migrate now**. Clearing 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 *.bat* file should have the following contents:

```
set INSTALL_PATH=InstallDirectory
set SHARED_INSTALL_PATH=SharedInstallDirectory
set JDK_PATH=jdk\jre\bin
set PLUGIN_PATH=plugins
set MIG_JAR=com.ibm.etools.egl.vagenmigration_version.jar
set STARTUP_JAR=org.eclipse.equinox.launcher_version.jar
set path=%INSTALL_PATH%\%JDK_PATH%;%path%
set classpath=%SHARED_INSTALL_PATH%\%PLUGIN_PATH%\%MIG_JAR%;
               %INSTALL_PATH%\%PLUGIN_PATH%\%STARTUP_JAR%

cd InstallDirectory
java com.ibm.etools.egl.internal.vagenmigration.batch.VGMIG
    -importFile Path\vgmigFileName.vgmig
    -data Path\workspace
    >Path\LogName.log
```

Note:

- The following statements must be written so that the statement is all on one line:
 - For Windows environments, the **set 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 only reflects the *last* of the *.vgmig* files to be migrated.
- *InstallDirectory* is the drive and directory in which you installed the EGL developer product. You must include the *InstallDirectory* for the set *INSTALL_PATH* statement and the *cd* (change directory) statement.
- *SharedInstallDirectory* is the drive and directory in which you installed the shared resources for the EGL developer product. You must include the *SharedInstallDirectory* for the set *SHARED_INSTALL_PATH* statement.

Note: If you installed and kept a previous version of the EGL developer product before installing the product that you are using now, the installation directory or shared installation directory of interest may be the directory that was used in the earlier install.

- *version* is the plug-in version number. To determine the version number, check the following plug-in directories:
 - The MIG_JAR plug-in is in *SharedInstallDirectory*\plugins. The version number might be similar to 7.5.0.RFB_20080811_1638.
 - The STARTUP_JAR plug-in is in *InstallDirectory*\plugins. The version number might be similar to 1.0.100.v20080509-1800.

In general, you should use the highest version number you see for the .jar file in corresponding plug-in directory.

- *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 have set your EGL and VAGen migration preferences. For details, see “Setting your workbench preferences” on page 171. In addition, the migration tool uses the *path\workspace* in the following ways:
 - If your .vgmig file specifies that the Stage 3 output is to be written to the current directory, *path\workspace* is the drive, directory and workspace name where the migration tool places the EGL files. In this case, the workspace name can be any name, for example, d:\mypath\myworkspace.
 - If your .vgmig file specifies that the Stage 3 output is to be written to temporary directories, *path* is the drive and directory under which the migration tool places one workspace for each migration set version specified by the .vgmig file. The workspace name must be stub.
- *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 migration 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:\myTempDirectory\stub\myProject\myMigrationInfo.vgmig
  -data d:\myTempDirectory\stub\
  >d:\migrationLogs\myMigrationInformationPiped.log
```

3. Shut down the EGL development environment.
4. Open a Command Prompt window, navigate to the directory containing your .bat file, and run your .bat file.

Note: You can safely ignore the following message:

PolicyClassLoader could not find policy
com.ibm.jxesupport.JxeClassLoaderPolicy.

5. When the process completes, your EGL project, source folders, packages, and files are stored in the directories you specified for them. 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 EGL development environment.
7. If you selected the **Import into workspace** option, 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 EGL. 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

There are two ways to run the Stage 3 tool:

- Automatically start Stage 3 at the end of Stage 2 by selecting either the **Import into Workspace** or the **Save migrated files to temporary directory** option when you set your options in the Stage 2 wizard (as described in “Setting up the Stage 2 VAGen migration file” on page 180). In general, this is the easiest way to run Stage 3 because it starts automatically.
- Run Stage 3 as a separate step. From the EGL development environment, in the Workbench window, follow these steps:
 1. Click **File->Import**.
 2. Expand **Other** and then select **VAGen Migration from Database**. 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 65. 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 Choose File to select an existing .vgmig file. Click Load File 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.	<code>path_to_db2java.zip\db2java.zip</code>
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: <ul style="list-style-type: none">• <code>jdbc:DB2:databaseName</code> <i>databaseName</i> 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.)

Table 65. Preferences to enter on first page of wizard (continued)

Preference	Meaning	Value
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 of the log file for the Stage 3 messages.	Enter a valid location (drive and directory) in the file system.
Log file name	This is the name of the log file for the Stage 3 messages.	Enter a valid file name.

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

Table 66. 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 JavaSource folders.	If you plan to generate COBOL only, click COBOL . If you might generate Java, click Java .
Latest version or Oldest version	This option specifies which version of the desired migration sets should be imported into the workspace.	Click 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 are overwritten.	<p>The Override existing files option 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 <i>only</i> those parts that are in the current migration set. If you clear 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.</p> <p>For more information on the impact of selecting this option, see “Overwriting and merging files” on page 48.</p>

Table 66. 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.	<p>If you plan to migrate multiple versions of a migration set, then follow these steps:</p> <ol style="list-style-type: none"> 1. Select this checkbox so that each version can be written to a different subdirectory. 2. Specify the Folder under which the subdirectories for the versions are placed. 3. Clear the Migrate now option. 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. 4. Select the Save current configuration to file option. 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 the Save migrated files to temporary directory option 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 your 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 the Migrate now option to run Stage 3 in online mode, or the Save current configuration to file option 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. Clear 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	<p>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:</p> <ul style="list-style-type: none"> • In online mode, right-click the saved .vgmig file and click Start Migration. • In batch mode, use the -importFile option to specify the saved .vgmig file. For details, see “Running Stage 2 in batch mode” on page 185. 	<p>You must select either the Migrate now option to run Stage 3 in online mode, or the Save current configuration to file option 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.</p> <p>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.</p>
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.

Table 66. Preferences to enter on second page of wizard (continued)

Preference	Meaning	Value
File name	This specifies the name of the file to which preferences are saved.	<i>fileName</i> , where <i>fileName</i> is the desired name for your file <i>without</i> a file extension. The extension <i>.vgmig</i> 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. If no migration sets are listed, check that you ran Stage 2 of migration.

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 EGL build path for the project 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 189 for details on setting up the .vgmig file to run just Stage 3. See “Running Stage 2 in batch mode” on page 185 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*.

There are two techniques you can use to bring the projects into a workspace:

- **Technique 1** is convenient if you only have a few projects in the subdirectory; however, it does not support EGL Web projects (VAGen Web Transactions or UI records). In this technique, you can point an existing workspace to each of the projects. This technique does not require that you 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, click **Window -> Preferences -> General -> Workspace** and clear **Build automatically**. This avoids multiple rebuilds while you are bringing in each of the projects.
 2. From the workbench view, click **File -> Import**. Expand **General** and then select **Existing Projects into Workspace**. Click **Next**.
 3. Click **Select root directory**. Click **Browse** to point to the subdirectory for a migration set version in the form: *migrationSetName_migrationSetVersion*.
 4. Select the projects that you want to include in your current workspace.
 5. Optionally, select **Copy projects into workspace**.

6. Click **Finish** to import the projects into your workspace.
- **Technique 2** is convenient if there are a number of projects in the subdirectory; it is required if you use EGL Web projects. In this technique, you bring up a workspace for the subdirectory by following these steps:
 1. Start the EGL development environment.
 2. When you are prompted for the workspace name, point to the subdirectory containing a migration set version and then click **OK**.
 3. Modify the following workbench preferences:
 - Your EGL capabilities and preferences as described in “Required EGL preferences” on page 172.
 - Any other preferences that you normally set for a new workspace.
 4. Build the workspace using one of the following techniques:
 - Click **Project -> Build Automatically**.
 - Click **Project -> Clean**. On the **Clean** window, click **Clean all projects**. Then click **OK**.
 5. If you have errors in the Problems view indicating that projects could not be built, use the Project Explorer view to locate any closed projects. Closed projects do not have the plus (+) symbol to the left of the project name. Open the closed projects by clicking **Open Project** from the pop-up menu. The projects should now be visible on the Project Explorer 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 EGL. 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, follow these steps:

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 pop-up menu, click **Import/Export -> VAGen Export** (or **VAGen Export with Associates**).
4. Type a name for the External Source Format file in the box and click **Save**. (If you type the name of an existing file, the tool asks if you would like to add parts to the file or overwrite it. Choose the answer that is appropriate for you.)

To prepare for single file mode, follow these steps:

1. Start the EGL development environment and point to your workspace. (For example, d:\workspaces\myworkspace.)
2. Set your EGL capabilities and preferences as described in “Required EGL preferences” on page 172.
3. Set your migration preferences. See “VAGen Migration preferences” on page 174 for information on how to do this.
4. From the Workbench window, click **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, table, and UI record is placed in its own file. This adheres to the EGL requirement of one generatable part per file. If you clear **Separate parts into EGL files**, all the parts except UI records are placed in the same EGL file. See “Overview of single file migration” on page 27 for specifics of the parts placement algorithm for single file mode.
5. Create a new EGL project. (For example, MyProject.) Use a **General Project** if you do not plan to migrate VAGen Web transactions, and a **Web Project** if you plan to migrate VAGen Web transactions or develop new EGL JSFHandlers.
6. Under the EGLSource directory for the EGL project, create a new EGL package. (For example, my.pkg)
7. See the section “Running single file migration using the user interface” on page 195 for details on running in online mode or “Running single file migration using batch mode” on page 197 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 EGL, follow these steps:

1. From the Project Explorer view, select the EGL package in which to put the resulting EGL file.
2. Right-click the package and then click **Import**.
3. From the Import window, expand **Other**. Select **VAGen External Source Format File** and click **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 you want to create from your External Source Format file. By default, the EGL file name is the same as the External Source Format file, but with the .egl file extension. See “Overview of single file migration” on page 27 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 package you specified, you are prompted to append to or overwrite the file. Based on your response to the overwrite prompt, the migration tool places the parts into the EGL files in the following way:
 - If you specify that you do not want to overwrite the existing targetFile, then any data items, functions, PSBs, and non-VGUI records in the second import are added to the targetFile. All the common parts in the second import result in duplicate parts within the targetFile.
 - If you specify that you want to overwrite the existing targetFile, then any data items, functions, PSBs, and non-VGUI records in the second import completely replace the targetFile. This results in the loss of any parts included in the first import, but not included in the second import.
 - If you selected the **Separate parts into EGL files** migration preference, the migration tool overwrites the files created for Programs, FormGroups, and DataTables. If you cleared the preference, then these parts are placed in the targetFile and added or overwritten based on your response to the overwrite prompt.
 - The migration tool always overwrites the files for VGUI records and .eglbld files.
10. When migration completes, the following output should appear:
 - One or more EGL files should be listed in the project, EGLSource folder, and package that you specified. See “Overview of single file migration” on page 27 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.

- The migration tool displays any error messages in a pop-up window. If you did not specify a log file location, click **Save to File** to save the messages in a file. Be sure to close the pop-up window.
11. If you have the **Build automatically** option selected, validation runs automatically. Otherwise, right-click the project and then click **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, follow these steps:

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

```
set INSTALL_PATH=InstallDirectory
set SHARED_INSTALL_PATH=SharedInstallDirectory
set JDK_PATH=jdk\jre\bin
set PLUGIN_PATH=plugins
set MIG_JAR=com.ibm.etools.egl.vagenmigration_version.jar
set STARTUP_JAR=org.eclipse.equinox.launcher_version.jar
set path=%INSTALL_PATH%\%JDK_PATH%;
set classpath=%SHARED_INSTALL_PATH%\%PLUGIN_PATH%\%MIG_JAR%;
                                     %INSTALL_PATH%\%PLUGIN_PATH%\%STARTUP_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
```

Note:

- The following statements must be written so that the statement is all on one line:
 - For Windows environments, the **set 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 EGL developer product. You must include the *InstallDirectory* for the set INSTALL_PATH statement and the cd (change directory) statement.
- *SharedInstallDirectory* is the drive and directory in which you installed the shared resources for the EGL developer product. You must include the *SharedInstallDirectory* for the set SHARED_INSTALL_PATH statement.

Note: If you installed and kept a previous version of the EGL developer product before installing the product that you are using now, the installation directory or shared installation directory of interest may be the directory that was used in the earlier install.

- *version* is the plug-in version number. To determine the version number, check the following plug-in directories:

- The MIG_JAR plug-in is in *SharedInstallDirectory*\plugins. The version number might be similar to 7.5.0.RFB_20080811_1638.
- The STARTUP_JAR plug-in is in *InstallDirectory*\plugins. The version number might be similar to 1.0.100.v20080509-1800.

In general, you should use the highest version number you see for the .jar file in corresponding plug-in directory.

- *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 27 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, directory, and workspace name where you want to place the EGL files (for example, d:\workspaces\myworkspace) If you do not specify the -data option, anything you specified in the VAGen Migration Preferences is ignored and the migration tool uses the default VAGen Migration Preferences. If you want to specify VAGen Migration 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 drive, directory, 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 optional, but is the only way to obtain messages from the migration tool in batch mode. 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:\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 EGL development environment.
3. Open a Command Prompt window, navigate to the directory containing your .bat file, and run your .bat file.

Note: You can safely ignore the following message:

PolicyClassLoader could not find policy
com.ibm.jxesupport.JxeClassLoaderPolicy.

4. When the process completes, your EGL files and log files are 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 EGL development environment.
6. Right-click the project into which you imported External Source Format files and then click **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 EGL. If you have the **Build automatically** option selected, this also 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 perform. This includes the following tasks:

- Set the Build Order preference.
- Export your preferences.
- Save a baseline for the EGL projects and packages in your source code repository.
- Preliminary tasks for completing single file migration.
- Review your EGL source code.
- Review your EGL build descriptor parts.
- Review your EGL linkage options parts.
- Review your EGL resource associations parts.
- Establish a bind control part to use as a template.
- Establish program-specific a bind control parts.
- Review your link edit commands.
- Review your VGWebTransactions.
- Prepare for debugging.
- Install the EGL server product.
- Convert VAGen preparation templates and procedures to EGL build scripts.
- Convert VAGen runtime templates.
- Convert the VAGen reserved words file.
- Generate and test with COBOL generation.
- Generate and test with Java generation.
- Review your standards.
- Plan for dual maintenance of your source code.
- Consider whether to eliminate the use of VisualAge Generator compatibility mode.

Setting the Build Order preference

When EGL builds the workspace, it builds the projects based on the **Build Order** preference. The default for this preference might not provide the best performance. In general you want the projects that are referenced the most frequently (such as common projects) to be built first. This ensures that when files in other projects import packages, the location of the common parts is already known. To see or manually change the current build order, follow these steps:

1. From the Workbench window, click **Window -> Preferences**.
2. Expand **General** and then **Workspace**. Select **Build Order**.
3. You can manually change the build order by following these steps:
 - a. Clear the **Use default build order** option.
 - b. Select projects in the **Project build order** list and use the **Up** and **Down** buttons to modify the build order.
 - c. After making all the changes, click **Apply** and then click **OK**.

4. You can also change the **Max iterations when building with cycles** to reduce or increase the number of times that the build tool cycles through the projects when attempting to resolve validation messages.

For EGL projects, there is a tool to help you set the build order. The Stage 3 migration tool automatically invokes this tool to set the build order before it starts the build. You might also need to run the build order tool if you add other projects to the workspace. To run the build order tool, follow these steps:

1. From the Workbench window, click **Project -> Optimize EGL Project Build Order**. The tool clears the **Use default build order** option and updates the **Project build order** list so that projects that are the most frequently referenced appear first in the build list.
2. Review the build order as described previously to determine if there are additional changes that you need based on your understanding of the project references.

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 suggested in “Required EGL preferences” on page 172, “Suggested preferences” on page 173, “VAGen Migration preferences” on page 174, and “Setting the Build Order preference” on page 203. 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, follow these steps:

1. From the Workbench window, click **File -> Export**.
2. On the Export page, expand **General**. Select **Preferences**. Click **Next**.
3. On the Export Preferences page, click **Export all**. Click **Browse** to specify a directory and **File name** in which to save your preferences.
4. Click **Finish**.

Other developers can import your preferences into a workspace by following these steps:

1. From the Workbench window, click **File -> Import**.
2. On the Import page, expand **General**. Select **Preferences**. Click **Next**.
3. On the Import Preferences page, click **Import all**. Click **Browse** to point to the file that contains the preferences.
4. Click **Finish**.

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 tasks for completing single file migration

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

- Nest any forms within their corresponding FormGroup. This is required if you migrate two FormGroups to the same package and the two FormGroups 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, right-click the current project in the Project Explorer view, then click **Properties**. Select **EGL Build Path**. On the EGL Build Path page, click the Projects tab to 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 tasks for both Stage 1 — 3 and single file migration

Reviewing your EGL source code

You need to perform the following tasks 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 DataTable definition. See Appendix C, "Messages from the migration tools," on page 395 for help in resolving messages in the migration log or the "TODO" list log. See Appendix D, "Messages in the Problems view," on page 427 for help with finding and resolving specific strings that the migration tool uses when it creates intentionally invalid EGL syntax.
- If you have Program, DataTable, or FormGroup 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. Specifying an

alias helps you avoid having to change any external references to the Program, DataTable, or FormGroup. If you change the name of a program, be sure to also change the name of any program-specific bind control or link edit parts that have the same name as the program.

- Review and resolve any additional errors in the Problems view. See Appendix E, “IWN.xxx messages in the Problems view,” on page 435 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 40.

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 435 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 tasks 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 develop programs for a national language in which the decimal point is the comma symbol, consider the following to determine whether to set the EGL **decimalSymbol** and **separatorSymbol** build descriptor options:
 - For programs that you generate to COBOL and that do not use print forms, the **decimalSymbol** and **separatorSymbol** build descriptor options take their default values from the language-dependent options module that is used at runtime. You can override the default values by explicitly setting the build descriptor options.
 - For programs that you generate to COBOL and that use print forms, the default value for the **decimalSymbol** option is a period, and the default value for the **separatorSymbol** option is a comma. If these values are not appropriate to your location, you must explicitly set these build descriptor options.
 - If you generate Java, you might want to set the **decimalSymbol** build descriptor option to improve runtime performance. If you do not set the **decimalSymbol** build descriptor option, you can set the **vgj.nls.number.decimal** property in the properties file used at runtime.
- 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 help for more information about this option.
- Refer to the online help 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 options 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 named *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.
- If you generate Web transaction parts for the COBOL environments, refer to the online help for information about the **secondaryTargetBuildDescriptor** that is used for generating the Java parts associated with VGUI records. The migration tool automatically splits any generation options part into two build descriptor parts if the generation option part contains any of the following options: */javadestdir*, */javadesthost*, */javadestpassword*, */javadestuserid*, or */javasystem*. One of the build descriptor parts is named *xxxxx*, and the other is named *xxxxx_TARGET2*, 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 are used when generating the EGL VGWebTransaction program for the primary (COBOL) runtime environment. The part named *xxxxx_TARGET2* contains the EGL replacement for all the VAGen generation options that are used when generating the EGL VGUI record for the secondary (Java) runtime environment. The migration tool places the EGL equivalent for the following generation options in the secondary build descriptor part: */javadestdir*, */javadesthost*, */javadestpassword*, */javadestuserid*, and */javasystem*. The migration tool places the EGL equivalent for the following options in both the primary and secondary build descriptor parts: */genout*, */genresourcebundle*, */msgtableprefix*, */resourcebundlelocale*, and */targnls*. The migration tool includes the **secondaryTargetBuildDescriptor** option in the primary build descriptor part and sets the value for the option to the name of the secondary build descriptor part.
- If you generate Web transaction parts and use message tables, you might need to change the **msgTablePrefix** build descriptor option. The message table is specified by the program that uses a VGUI record. If the message table and the VGUI record are in different packages, you must modify the secondary build descriptor part and include the package name (for example, *msgTablePrefix = "packageName.prefixID"*).
- The migration tool does not create a default build descriptor for you when it creates the EGL non-Web projects. This enables you to specify one of your migrated build descriptor parts as the default build descriptor. 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. Use one of the following techniques to set a default build descriptor.
 - To set a preference for a particular file, package, EGL source folder, or project, right-click the resource (file, package, folder, or project), then click **Properties**. Select **EGL Default Build Descriptors** 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 is used whenever you generate anything in this resource. The **Debug build descriptor** is the default that is used when you use the debugging tool.

- To set a workbench preference for a build descriptor part, click **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.
- The migration tool creates a default build descriptor for you when it creates the first EGL Web project in a migration set. This ensures that the VGUI records can be generated into JSPs when the workspace is refreshed at the end of Stage 3. You can change the default build descriptor for the project, the EGL source folder, or any packages or files the project contains.
- If your control parts (build descriptor, linkage options, resource association, bind control, and link edit 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.
 - The transfer step requires a code page conversion. You should test that all the special characters that you use are transferred correctly. For example, the ¯ (not sign symbol) converts to different code points when using the default VAGen and default EGL conversion tables for U. S. English. In VAGen, the default value for the /contable generation option is ELACNENU. The equivalent EGL build descriptor options are serverCodeSet="IBM-037" and clientCodeSet="IBM-1252". However, the EGL defaults for these build descriptor options are serverCodeSet="IBM-037" and clientCodeSet="IBM-850". Therefore, you might need to set the **serverCodeSet** and **clientCodeSet** build descriptor options if you have special characters or use a language other than U. S. English.
 - EGL uses a build server to handle the preparation steps. For the EGL z/OS and iSeries build servers, you must specify the **destPort** build descriptor option that is used to transfer the outputs of generation. Contact the person who installed and configured the build server to determine the port number on which the remote build server is listening for build requests.
- If your COBOL compiler supports only a maximum of 18 digits for numeric fields, set the **maxNumericDigits** build descriptor option to NO.
- If your program uses SQL, and any SQL statement uses an SQL built-in function such as sum or other math operations, you should add the symbolic parameter **ADDSPACESAROUNDSQLHYPHENS** and set the value to YES. This symbolic parameter ensures that EGL inserts a blank between the column names and the operators and matches the default behavior for VisualAge Generator. If you do not use this symbolic parameter, you might see one or both of the following errors:

- SQLCODE -206 or SQL0206N
- SQLCODE -113 or SQL0113N
- If you do not use Web transactions and do not plan to create EGL page handlers, you might want to set the EGL build descriptor option **genResourceBundle** to NO in your build descriptor parts that generate for COBOL runtime environments. This prevents the Java generation of your DataTables.
- If you generate COBOL for the z/OS environment, you might also need to perform the following tasks:
 - Establish a bind control part to use as a template. (See “Establishing a bind control part to use as a template” on page 212.)
 - Establish a program-specific bind control part. (See “Establishing a program-specific bind control part” on page 214.)
 - Review link edit commands. (See “Reviewing link edit commands” on page 214.)
 - If you generate COBOL for the VSE environment, you might also need to review link edit commands. (See “Reviewing link edit commands” on page 214.)

Reviewing Java generation build descriptor options

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

- 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. The **genProject** option is required in the following cases:
 - If you generate for HP-UX or SOLARIS
 - If you generate VGWebTransactions or VGUI records or their associated parts such as DataTables. In this case, be sure that the **genProject** option specifies an EGL Web project.
- There are some EGL build descriptor options that have somewhat different behavior from their corresponding VAGen generation option. Refer to the online help 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** and **wrapperCompatibility**, which are 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 help for information about the following build descriptor options to determine whether you need to set them for your environment:
 - **dateMask**
 - **sessionBeanID**
 - **sqlJDBCClass**
 - **sqlValidationConnectionURL**
 - **tempDirectory** (for VGUI records only)

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 help for guidance on creating a debug build descriptor part.

Reviewing your EGL linkage options 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 435 for help in resolving common messages in the Problems view that are a result of the migration process. Also refer to the online help for details about the linkage options that are supported for your environment. You need to perform the following tasks 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 the **callLink** element, consider the following points:
 - Not all of the linktypes 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, DCESECURE, and APPCIMS 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 use remote calls to CICS, you need to specify additional properties based on the **remoteComType** values:
 - If you change to using **remoteComType** = "CICSECI", you must add the **ctgPort** and **ctgLocation** properties.
 - If you change to using **remoteComType** = "CICSSSL", you must add the **ctgKeyStore** and **ctgKeyStorePassword** properties. In addition, if you have not already included the **ctgPort** and **ctgLocation** properties in your VAGen linkage table, you must include them for the EGL **remoteComType** = "CICSSSL".
 - If you change to using **remoteComType** = "CICSJ2C", you must add the **pgmName**, **conversionTable**, **remotePgmType**, **luwControl**, **remoteBind**, **location**, and **parmForm** properties.

Regardless of the communications protocol that you choose, you must set up and configure a CICS Transaction Gateway infrastructure. Direct calls to CICS using the CICS Client product are no longer supported.

- If you change to using **remoteComType** = "IMSTCP" as a replacement for APPCIMS, refer to the online help for assistance in setting the additional properties that are necessary. Also review the online help for existing properties because the values that must be specified have different meanings for IMSTCP.
- **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 options part. However, you must modify the value.
- You might need to add **callLink** entries. EGL requires **callLink** elements for the following situations:
 - If a generated Java program calls a native C++ or a VAGen generated program, it is always a remote call even if the programs are running on the same workstation. A **callLink** entry is required.

- If you used the VAGen generation option /system=JAVAWRAPPER for a called program, you must create an EGL **callLink** entry with the **javaWrapper** property set to YES. If you do not have the entry, EGL does not generate the Java wrapper.
- If you generate Java and a called program name conflicts with an EGL reserved word, you must create an EGL **callLink** entry and set the **alias** property to the actual name of the called program.
- If you generate for CICS and the called program requires special linkage, such as a PL/I program. For example, if you call a PL/I program, the VAGen default linkage is CICSLINK and COMMPTR regardless of whether the call is to a VAGen or non-VAGen program. In EGL, the default linkage for calls to EGL programs is DYNAMIC and COMMPTR and the default linkage for calls to non-EGL programs is CICSLINK and COMMPTR. In this case, if your **call** statement does not specify **isExternal** = yes, then you must create a **callLink** entry and set **linkType** = "CICSLINK", **parmForm** = "COMMPTR", and **pgmType** = "EXTERNALLYDEFINED" so that the correct linkage is used for the PL/I program.
- If you shared a linkage options part in VAGen for ITF and COBOL generation, you might have errors in the EGL Problems view for **remoteCall** options that are not supported during COBOL generation. In this case, split your EGL linkage options part into two separate linkage options parts. For example, you might use the following technique:
 - Copy your original linkage options part to use for COBOL generation. Remove the **remoteCall** entries that are used only during debug. Be sure to keep any **remoteCall** entries that are used at runtime to call programs in another CICS region.
 - Make another copy of your original linkage options part to use for debug. Remove the **localCall** entries that are used only during COBOL generation. Be sure to keep any **localCall** entries that might be used for calling a native Java or C++ program during debug.
- For a **fileLink** element, **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 options part. However, you must modify the value.
- For an **asynchLink** element (VAGen crtlink), **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 options part. However, you must modify the value.
- The EGL **transferToProgram** element 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 **transferToProgram** entries. Refer to the online help for information about this new linkage entry.

Reviewing your EGL resource associations parts

The migration tool converts VAGen resource associations 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 435 for help in resolving common messages in the Problems view that are a result of the migration process. Also refer to the online help for details about the resource association options that are supported for your environment. You need to perform the following tasks 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 associations 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 properties you must set for the resource association entry.
- Review the online help for the **formFeedOnClose** and **text** properties 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 runtime resource association file for the workstation environment. Therefore, these options are not set by the migration tool because the tool only processes resource associations parts.

Establishing a bind control part to use as a template

Note: This section applies to you only if you generate COBOL for the z/OS environments.

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 EFK2MBDx.tpl where *x* is a letter. Bind control parts are only required if you need to do a special bind for a particular program. Table 67 shows the VAGen bind control templates based on the runtime environment and database access.

Table 67. VAGen bind command templates

Environment and Database Access	VisualAge Generator BIND Template
MVS CICS - with DB2	EFK2MBDA
MVS Batch - with DL/I and DB2	EFK2MBDA
MVS Batch - with DB2 only	EFK2MBDD
IMS/VS- with DL/I with DB2 work database	EFK2MBDC
IMS/VS- with DL/I and DB2 with DB2 work database	EFK2MBDB
IMS/VS- with DL/I and DB2 without DB2 work database	EFK2MBDA
IMS BMP - with DL/I and DB2	EFK2MBDA

EGL does not use externally defined bind control templates. Instead EGL uses an internal template or 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 214 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. You might have a VAGen bind control template that looks like the following example:

```
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 three additional lines and a change to the EZEMBR symbolic parameter. The corresponding EGL bind control part looks like the following example, with the additional and changed lines highlighted in bold:

```
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 runtime 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 three 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 EGL build server to determine what the additional three 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 384 for changes to the names of the symbolic parameters. Also see the online help 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. To minimize the number of build descriptor parts you need to modify, consider adding the **bind** build descriptor option to one of your existing, common build descriptor parts.

Note: Be sure to compare the VAGen bind control templates for any of your target environments. If the templates are different, you might be able to add additional symbolic parameters to support the differences. Alternatively, you

might need to set different **bind** build descriptor options on a program basis to point to different EGL bind control parts that are needed as templates for different target environments.

Establishing a program-specific bind control part

Note: This section applies to you only if you generate COBOL for the z/OS environments.

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 212. 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 three 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 uses the bind control part that is the template because that is what your normal **bind** build descriptor option specifies.

Reviewing link edit commands

Note: This section applies to you only if you generate COBOL for the z/OS or VSE environments.

VisualAge Generator provides default link edit commands based on the target environment and database access. However, in some cases, you might have a program that requires specific link edit 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 link edit 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 link edit 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 link edit part. In EGL, if you do not specify the **linkEdit** build descriptor option, then EGL looks for a link edit part with the same name as the program. If EGL does not find a link edit part with the same name as the program, then EGL creates default link edit 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 link edit part with a different name from the program. You might need to do this if you generate the same program for several COBOL environments.

If you have VAGen program-specific link edit 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 finds the program-specific part first, before it attempts to create a default link edit command.

Reviewing your VGWebTransactions

You should consider the following points when reviewing your migrated VGWebTransaction programs:

- Before you debug your VGWebTransaction programs, you must generate all the VGWebTransaction programs and VGUIRecords.
- When you deploy or use an EGL-generated bean, you must perform the following tasks:
 - Regenerate all the VGUIRecords that you plan to include in the new WAR file.
 - Regenerate all of the corresponding VGWebTransaction programs.
 - Migrate and generate any programs that the VGWebTransaction program calls or transfers to using the **call**, **transfer**, or **show** statements (VAGen CALL, DXFR, or XFER statements).

You do not need to migrate and generate programs that are referenced in a program link or hyper link.

- If the EGL package name differs from the VAGen package name, you must update your JSPs and properties files. The package name might have changed for any of the following reasons:
 - You used the Stage 1 migration tool renaming rules to rename packages because they conflict with EGL reserved words.
 - You used the Stage 1 migration tool renaming rules or modified the Stage 1 tool to consolidate packages in EGL.
 - You used the VAGen /packagename generation option to specify the runtime package name, which can be different from the package name of the VAGen source code. In EGL, the runtime package name is normally the same as the package name of the EGL source.

For information about a white paper that can help you update the package name in your modified JSPs for use with EGL, see “References” on page 16.

- To deploy your EGL Web transactions, refer to the online help for assistance. Be sure to perform the following tasks:
 - Review and modify the default gw.properties file in the src folder for the project. Be sure to set the **hptEntryPage** and the **hptEntryApp** values. You might be able to copy this information from the corresponding gw.properties file in your VisualAge Generator system. You might need to set additional

options depending on the modifications you made to the VisualAge Generator gw.properties file. The **hptDisableRMIIDManager** option was added in a VisualAge Generator Fix Pack. If this option is new to you, review the EGL online help for assistance in setting the value.

- Review and modify the default csogw.properties file in the src folder for the project. You might need to update the **serverLinkage** entries to change the **javaProperty** information to reflect any changes to your package names. Be sure to include the information to specify which applications are to be found on which server. You might be able to copy this information from the corresponding csogw.properties file in your VisualAge Generator system. You might need to set additional options depending on the modifications you made to the VisualAge Generator csogw.properties file.
- Review and modify the default Vagen1EntryPage.jsp in the WebContent folder for the project. Be sure to update the OPTION information for **hptAppId** to include the names of your VGWebTransaction programs and the associated text that you want to display for each program in the list. You might be able to copy this information from the corresponding JSP file in your VisualAge Generator system.
- Generate the project.
- Create an Enterprise Application Resource (EAR) project by following these steps:
 1. From the Workbench window, in the Project Explorer view, click **New -> Other -> J2EE -> Enterprise Application Project**.
 2. Enter the **Name** of the EAR project.
 3. Click **Next**.
 4. Select the projects to include in the EAR Project.
 5. Click **Finish**.
- Define a Web Application Server.
- Add the EAR Project to the server.
- Run the application by right-clicking the EGLWebStartup.jsp in the WebContent folder for the project and then clicking **Run -> Run on Server**.

Preparing for debugging

To prepare for debugging, you should perform the following tasks:

- From the Workbench window, click **Window -> Preferences -> EGL -> Debug**. Refer to the online help to determine which, if any, of these preferences you need to set for your environment. For example:
 - If you used the VAGen Test preference to **Run in EBCDIC** mode, you should set the EGL Debug preference **Character Encoding** to the EBCDIC code page for your host system.
 - You might also want to select **Stop at first line of a program** and **Enable hotswapping**.
- If you used the VAGen Test preference **Programs in Library to Bypass**, you might need to provide information in the **EGL -> Debug -> Debug Behavior Mapping** preference page. In VisualAge Generator, the Test Facility looks at the linkage table part first and then checks whether the information was overridden by the Test preferences. In EGL, the **EGL -> Debug -> Debug Behavior Mapping** preference specifies the situations in which the Debugger looks at the linkage options part. Therefore, on the **Called Program** page, you need to list all the non-EGL (or generated EGL) programs that you want to call on the host. Use **generated** as the mapping mode. If the program is not listed on the **Called**

Program page, then the Debugger displays a pop-up window when the EGL source cannot be found in the workspace. For additional details about the **Debug Behavior Mapping** preference, refer to the EGL online help.

- 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.
- The VAGen Date Formats preference enables you to specify the date format used in Test Facility for the VAGen EZE formatted date words (EZEDAYLC or EZEDTELC) and the Date edit masks for fields defined on maps or in UI records. In EGL, you must provide the equivalent date format information in the debug build descriptor part. To avoid problems due to differences in the default date formats, specify this information for EGL even if you did not have to for VisualAge Generator. Use the following steps to set the date format:
 1. Edit your debug build descriptor part.
 2. Click **Show Java Run-time Properties** (the icon is on the upper right side of the tool bar within the editor).
 3. In the Date Masks section of the editor, click **Add**.
 4. In the **Locale** field, click once to get a drop down list to set your locale (for example, ENU).
 5. Click in the **Long Gregorian Mask** field to get a drop down list to set the Gregorian format used for your locale (for example, yyyy-MM-dd).
 6. Click in the **Long Julian Mask** field to get a drop down list to set the Julian format used for your locale (for example, yyyy-DDD).
 7. Similarly, set the **Short Gregorian Mask** and the **Short Julian Mask** fields (for example yy-MM-dd and yy-DDD respectively).
 8. Repeat steps 3 through 7 to add information for any other locales that you use.
 9. Click **Show General Build Descriptor Options** to return to the main build descriptor editor.
 10. Save and close your build descriptor file.

Note: For z/OS and VSE, the values you specify for a locale should match the values that your installation specifies for the corresponding language-dependent options module in Rational COBOL Runtime for zSeries or Rational COBOL Runtime for z/VSE.

- If you are calling generated EGL or non-EGL programs on a remote CICS system from the debugger, you need to set up and configure a CICS Transaction Gateway infrastructure. Direct calls to CICS using the CICS Client product are no longer supported.
- If you plan to use the EGL debugger for programs that use DL/I database I/O, or to call programs on a remote IMS system, you need to configure a number of files. Refer to the *EGL Programmer's Guide* for assistance.
- If you are using remote VSAM files on z/OS, you need to install and use the Distributed File Manager (DFM). Refer to the *EGL Generation Guide* for details of how to install DFM and specify the resource association entries for debug.

Installing the EGL server product for zSeries

Note: This section applies to you only if you generate COBOL for the z/OS environments.

For z/OS, the EGL server must be installed in a separate SMP/E zone and have different target libraries from the VisualAge Generator Server for MVS, VSE, and VM (VAGen server product).

If you placed any of the VAGen server product load modules in the LPA, you must replace them with the EGL server product load modules before migration is complete. When you make this change be sure to avoid having a combination of modules from VAGen server product and the EGL server product because this can cause unpredictable results. Therefore, if you removed the VAGen server product load modules from the VAGen SELALMD load library when you originally placed the load modules in the LPA, follow these steps to keep the sets of load modules consistent:

1. Put the VAGen modules back into the VAGen SELALMD load library.
2. Remove the VAGen load modules from the LPA.
3. Add the EGL server product load modules to the LPA.
4. Remove the EGL server product load modules from the EGL SELALMD load library

Even though EGL uses a build server for the preparation process, the EGL server product includes the same preparation JCL procedures as the VAGen server product. If you plan to continue generating VAGen programs, but use the EGL runtime server, you must tailor the EGL preparation JCL procedures (or retaylor your existing VAGen preparation JCL procedures) to point to the EGL runtime server libraries.

The EGL server product is also compatible with VAGen-generated programs. Therefore, you can migrate to the EGL server product before you migrate your source code to EGL. For example, your IMS or CICS regions might evolve over time in the following way:

- A production region with the VAGen server load library and an application load library with VAGen-generated programs. A test region that is identical to the production region.
- A production region with the VAGen server load library and an application load library with VAGen-generated programs. A test region with the EGL server load library and an application load library with VAGen-generated programs so that you can ensure that existing VAGen programs run the same as before. Optionally, a second test region with the EGL server load library and an application load library with EGL-generated programs so that you can test programs during your pilot migration.
- A production region with the EGL server load library and an application load library with VAGen-generated programs. A test region with the EGL server load library and an application load library with VAGen-generated programs so that you can continue developing and maintaining programs in VisualAge Generator during your pilot migration. A second test region with EGL server load library and an application load library with EGL-generated programs so that you can test EGL programs during your pilot migration.
- A production region with the EGL server load library with a mixture of VAGen-generated and EGL-generated programs. A test region with the EGL server load library with a mixture of VAGen-generated and EGL-generated programs. The test region has more EGL-generated programs than the production region. If you migrate all your source code to EGL at the same time, you can skip this configuration of regions.
- A production region with the EGL server load library with EGL-generated programs. A test region that is identical to the production region.

Installing the EGL server product for VSE

Note: This section applies to you only if you generate COBOL for the VSE environments.

For VSE, you must install IBM Rational COBOL Runtime for z/VSE. Refer to the Program Directory for details.

Converting VAGen preparation templates and procedures to EGL build scripts

Note: This section applies to you only if you generate COBOL.

In VisualAge Generator, for COBOL generation, preparation templates are used to control the preparation process. The template that is used depends on the type of part that is generated, the runtime environment, and, for program parts, the type of database access. The /TEMPLATES generation option points to the drive and directory that contains the preparation templates. The templates vary by environment in the following way:

- For the MVS runtime environments, preparation templates are used to generate the JCL necessary to do the DB2 precompile, CICS translate, COBOL compile, link edit, and bind. The preparation templates invoke JCL procedures to provide the actual steps in the preparation process.
- For the VSE runtime environments, preparation templates are used to generate the JCL necessary to do the DB2 precompile, CICS translate, COBOL compile, and link edit. The preparation templates invoke JCL procedures to provide the actual steps in the preparation process.
- For the OS/400 runtime environment, preparation templates are used to generate the control language (CL) to do the compile and bind.

In EGL, the preparation process is handled in the following way:

- For zSeries and iSeries, preparation is handled by an EGL build server. The EGL build scripts merge the information previously contained in the VAGen preparation templates and preparation procedures.
- For VSE, the preparation process is still handled by preparation templates and procedures. For details, refer to the documentation for the Generation for VSE feature.

For zSeries, review your VAGen preparation templates and procedures to determine whether you customized them. If so, the best way to modify and debug an EGL build script is to follow these steps:

1. Set the **prep** build descriptor option to NO.
2. Generate a program for your runtime environment. Review the preparation file that is created to determine the name of the build script that is being used for this type of program, database access, and runtime environment.
3. Manually upload the outputs of generation to the data sets you plan to use on the z/OS host.
4. Create preparation JCL for the program. You can use your VAGen preparation templates and procedures as a starting point and then modify them to point to the EGL data sets. Also compare the preparation templates and procedures to the build scripts supplied by EGL to determine if there are additional steps, libraries, and so on required for EGL.

5. Test the preparation JCL until you are satisfied that your modifications are correct.
6. Convert the preparation JCL to pseudo-JCL so it can be used as a build script. For details, refer to the *IBM Rational COBOL Runtime Guide for zSeries*.
7. Set the **prep** build descriptor option to YES.
8. Generate the program again for your runtime environment. The outputs of generation should now be uploaded and prepared automatically using the build server.
9. Repeat the process for programs that access each type of database in each runtime environment. Be sure to generate FormGroups (with both text and print forms) and DataTables for all your runtime environments because these use different build scripts.

For zSeries, also see “Preparation templates and procedures” on page 386 for a list of the VAGen templates and procedures and their corresponding EGL build scripts. VisualAge Generator also uses bind control templates for the MVS runtime environments. See “Establishing a bind control part to use as a template” on page 212 for details of how to convert the MVS bind control templates.

For iSeries, refer to the *Rational Business Developer EGL Server Guide for IBM i* for information about how to customize the build script.

Converting VAGen runtime templates

Note: This section applies to you only if you generate COBOL.

In VisualAge Generator, runtime templates are used to generate the following code:

- Sample runtime JCL for the MVS Batch and IMS BMP runtime environments.
- Sample runtime JCL for the VSE Batch environment.
- Sample control language for the iSeries environment.

The /TEMPLATES generation option points to the drive and directory that contains the runtime templates. The template that is used depends on the type of program, the runtime environment, and the type of file or database access. The following types of runtime JCL templates are used for MVS Batch, IMS BMP, and VSE:

- Execution JCL templates that create the main portion of the sample runtime JCL.
- File and database allocation templates that create DD statements that are included in the sample JCL for DL/I databases, serial, indexed, or relative files based on the runtime environment and the file implementation specified by the resource association information.
- File and database allocation placeholder templates that create comments that are included in the sample JCL when the program does something that might require additional DD statements, but the information is not available to generation. For example, if ProgramA calls ProgramB, when you generate ProgramA, there is no way to determine the DD statements required by ProgramB. The file and database allocation placeholder templates include a comment in the sample runtime JCL for ProgramA to indicate that it calls ProgramB.

Similarly, for OS/400, there are several runtime CL templates.

In EGL, there are also runtime templates. The **templateDir** build descriptor option points to the directory where the templates reside. The default directory is the following subdirectory within your shared product installation directory:

```
plugins\com.ibm.etools.egl.generators.cobol_version\MVSTemplates
plugins\com.ibm.etools.egl.generators.cobol_version\VSETemplates
plugins\com.ibm.etools.egl.generators.cobol_version\iSeriesTemplates
```

version is the highest version level of COBOL generation that you have installed.

If you need to tailor the EGL runtime JCL or CL templates, consider the following points:

- Create your own directory outside the product installation directory. This simplifies installing EGL maintenance because you do not have to worry about overlaying your customized templates.
- Consider putting this directory on a shared drive where it can be accessed by all developers. This makes it easier to change a template because you do not have to distribute the new template to all the developer workstations.
- For a list of the VAGen runtime templates and the corresponding EGL runtime templates, see the following information:
 - For zSeries and iSeries, see “Runtime templates” on page 388.
 - For VSE information, refer to the *Rational Business Developer V7.5 Generation for z/VSE feature Reference Manual* (SC19-2539-00)

Use your VAGen runtime templates as a starting point and compare each template to the corresponding EGL template to determine the tailoring you might require. Be sure to change the VAGen symbolic parameters to the corresponding EGL symbolic parameters as shown in “Symbolic parameters” on page 384.

Converting the VAGen reserved words file

Note: This section applies to you only if you generate COBOL.

In VisualAge Generator, the **/RESVWORD** generation option points to the drive, directory, and file name of an optional reserved words file. This file contains all the COBOL, SQL, and CICS reserved words that are not permitted as part or field names in the generated COBOL programs. The default reserved words file is shipped with VisualAge Generator. If you specified the **/RESVWORD** generation option, you probably added additional words to the list. If you never made modifications to the reserved words list, you can skip this section.

In EGL, the standard COBOL, SQL, and CICS reserved words are predefined in the COBOL generator. The EGL reserved words file only contains your additions to the list.

Consider whether you still really need additions to the EGL reserved word list. If so, create a file on the workstation with a list of the additional words you require. Then modify your EGL build descriptor parts to include the **reservedWord** build descriptor option and point this option to your reserved words file. You might want to put the reserved words file on a shared drive so that everyone can access a single copy of the file rather than propagating the file to every developer workstation. This technique simplifies making a change to the reserved words file.

Generating and testing with COBOL generation

To prepare for COBOL generation, perform the following tasks:

- 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.
- Be sure that the EGL runtime server for your environment is installed with all of the latest PTFs.
- For z/OS and iSeries, be sure the EGL build server is installed with all of the latest PTFs. Also, be sure it is configured in your host environment. In VisualAge Generator, customization was done to the preparation process for z/OS and iSeries by changing preparation templates on the workstation. In EGL, this customization is done on the host machine. For more information, see “Converting VAGen preparation templates and procedures to EGL build scripts” on page 219.
- For VSE, customize the preparation JCL templates and procedures.
- Contact the person who installed and configured the EGL 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 default 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 DataTables. When you generate the programs, use the following build descriptor options:
 - **genFormGroup**="YES"
 - **genHelpFormGroup**="YES"
 - **genDataTables**="NO"

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

Note: For details of when you must generate the EGL programs or relink the VAGen programs, see Appendix I, “VisualAge Generator and EGL interoperability,” on page 473.

- VAGen and EGL create the names of COBOL records and items differently. The EGL names result in a more readable COBOL program, but can result in a generated COBOL program that exceeds the SQL statement precompiler limits, depending on your SQL product. The following are examples of precompiler limits:
 - Maximum number of processed lines. All SQL statements must occur in the program prior to this limit. COBOL generation places the SQL statements as early as possible in the Procedure Division. However, a program might encounter this limit if it has many SQL functions or large numbers of data items in records or on forms.
 - Maximum number of unique host variables. Each host variable that allows nulls also has an indicator variable that counts toward the maximum. By default, the VAGen to EGL Migration tool sets the **isSQLNullable** property to YES for each field in the SQL records to preserve the VAGen behavior.

- Maximum number of lines or characters for an SQL statement.

If any of the SQL precompiler limits is exceeded, you need to make one or more of the following changes to the program:

- If some of the columns in your SQL tables are defined as NOT NULL, remove the **isSQLNullable** = yes property from the corresponding field in the EGL SQL record definitions. This reduces the number of unique host variables which in turn reduces the number of characters and lines for an SQL statement and the total number of lines for the program. This technique has the biggest impact for the least amount of work and also has the potential of improving runtime performance.
- Review the use of default SQL statements. If the default statements are retrieving more columns than you actually need, modify the statements to specify only the required columns.
- Shorten the name of the SQL record.
- Split the SQL statements into multiple statements. For example, change one **get** statement into multiple **get** statements and retrieve a subset of the columns in each statement.
- Split the program into multiple programs.
- 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

To prepare for Java generation, perform the following tasks:

- 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 DataTables. When you generate the programs, use the following build descriptor options:
 - **genFormGroup**="YES"
 - **genHelpFormGroup**="YES"
 - **genDataTables**="NO"

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

Note: For details of when you must generate the EGL programs or relink the VAGen programs, see Appendix I, “VisualAge Generator and EGL interoperability,” on page 473.

- If you modified the VAGen product message text, you can make similar modifications to the EGL message text.
- 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 is still readable. The only time an alias name is assigned is if there are duplicate part or variable names after changing the underscore to a hyphen.
- To improve the readability of generated COBOL code, avoid the use of names that cause generation to assign an alias. 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.
 - DataItem names should be 27 or fewer characters. This is slightly less than the VAGen suggested limit of 30 characters for COBOL generation and the maximum of 32 characters for VisualAge Generator.
- Program and DataTable names can now be 8 characters.

Planning for dual maintenance of your source code

You can migrate your VAGen source code to EGL. However, you cannot migrate an EGL source file back to VisualAge Generator. If you are not migrating all of your VAGen source code at the same time and have parts that are needed by both the VAGen and the EGL code, you need to make changes to the common parts in one of the following ways:

- Make the change in both the VAGen and EGL versions of the part.
- Make the change in VAGen and then export an external source format file for the part. Depending on the changes, to ensure that cross-part migration can be done, you might need to include parts used by the changed part, as well as parts that use the changed part in the external source format file. Import the external source file into a new file in EGL. Then compare the changes for the newly-migrated parts to the original EGL parts and move the changes to the correct location within the EGL workspace.

The simplest solution to the dual maintenance problem is to avoid the problem completely in the following way:

- Freeze your VAGen development and maintenance while you are migrating to EGL.
- Generate and test all the VAGen programs that are currently work-in-progress and move them into production. This enables you to migrate just the production version of your source code.

If you cannot migrate everything at the same time, try to limit the changes to a small number of parts and then make the changes in both the VAGen and the EGL versions of the part. Making the changes in both places can be easier than changing the part in VAGen and migrating it to EGL again because you do not have to determine all the related parts that are required for cross-part migration and you do not have to move the newly-migrated parts to the correct location within the EGL workspace.

Eliminating the use of VisualAge Generator compatibility mode

VisualAge Generator compatibility mode supports a number of behaviors that make it easier to preserve the behavior of your VAGen programs. It is not necessary to turn off VisualAge Generator compatibility mode. However, particularly if you use only a few of these compatibility behaviors, you might want to eliminate the use of VisualAge Generator compatibility mode. The following list describes the EGL behavior when VisualAge Generator compatibility mode is turned on and provides information that might be helpful in removing the need to use the behavior.

- EGL permits the use of the hyphen (-) and the national language characters @ and # in part names. If you plan to eliminate the use of VisualAge Generator compatibility mode, you should create a **Rename user exit** to use during Stage 2 migration. Create the **Rename user exit** so that it eliminates the use of the hyphen, @, and #. For example, if you never use the underscore in your VAGen part names, you can create a **Rename user exit** that renames parts by changing the hyphen to an underscore. See “VAGen Migration preferences” on page 174 for details of how to specify a **Rename user exit** for Stage 2 migration. If you turn off VisualAge Generator compatibility mode, EGL validation displays an error message in the Problems view for any part name or variable name that contains a hyphen, @ or #. You can correct the problem by changing the name. For examples of creating an exit routine, see the white paper on using the **Rename user exit** listed in “References” on page 16.
- EGL permits the use of the primitive data types NUMC and PACF. NUMC is similar to NUM except that NUMC uses the C as the positive sign indicator. PACF is similar to DECIMAL (VAGen PACK) except that PACF uses the F as the positive sign indicator. If you turn off VisualAge Generator compatibility mode, EGL validation displays an error message in the Problems view for any DataItem definition or variable declaration that specifies a primitive type of NUMC or PACF. You might be able to correct the problem by changing the primitive type to NUM or DECIMAL respectively and then making any related program changes required for the new positive sign indicator. Depending on how the NUMC or PACF field is being used, you might also need to change the data in databases, files, or DataTables.
- EGL defaults the subscript to 1 for single-dimension, structure-field arrays. Single-dimension, structure-field arrays are the EGL equivalent of a VAGen array or multiply occurring item in a record, map or table. If you turn off VisualAge Generator compatibility mode, EGL validation displays an error message in the Problems view for any statement that specifies a structure-field array that now requires a subscript. To correct the problem, you must modify the statement to explicitly specify a subscript of 1.
- EGL permits the **deleteAfterUse** property on a **use** declaration for a DataTable. The **deleteAfterUse** property is the replacement for the VAGen Keep After Use property. If you turn off VisualAge Generator compatibility mode, EGL validation displays an error message in the Problems view for each program that contains a **use** declaration that specifies **deleteAfterUse**. You can correct the problem by removing the **deleteAfterUse** property, in which case EGL treats the table similarly to a VAGen table that specifies Keep After Use = yes.

If your production programs are already generated using VisualAge Generator Version 4.5 Fix Pack 4 or higher, there is no impact from eliminating the EGL **deleteAfterUse** property. If you are migrating from Cross System Product or an earlier version of VisualAge Generator, you should thoroughly test any program that you change to eliminate the **deleteAfterUse** property.

If you select the **Do not include deleteAfterUse** migration preference for tables, the migration tool automatically omits the **deleteAfterUse** property and issues a warning message for the affected program and table.

- EGL implements a **display printForm** statement in the same way as a **print printForm** statement. You can minimize the use of **display printForm** by including the maps in your migration set. This enables the migration tool to correctly migrate to a **display** statement for text maps and a **print** statement for printer maps. If you turn off VisualAge Generator compatibility mode, EGL validation displays an error message in the Problems view for each **display printForm** statement. You can correct the problem by modifying the statement to be a **print** statement.
- EGL uses the **value** property of a form field only when displaying a field on the screen that has not had a value assigned to it. The **value** property does not set the initial value of the form field in storage. The migration tool includes the **value** property when it migrates the map. If you turn off VisualAge Generator compatibility mode, EGL uses the **value** property to set the initial value of the form field in storage. There might or might not be an error in the Problems view. For example, in VisualAge Generator, a numeric map variable field can have an initial value of "MM/DD/YYYY" to provide a value for the Map Editor Preview mode and to provide output to the user if the program does not move any data to the field before the DISPLAY or CONVERSE I/O option. In this example, if you turn off VisualAge Generator compatibility mode, there might be a message in the Problems view because the value is not compatible with the NUM primitive type; you might also see a runtime error. However, if the initial value is a number such as 5, there is no message in the Problems view, but the program might not behave in the same way it did in VisualAge Generator. If you turn off VisualAge Generator compatibility mode, in addition to correcting the messages in the Problems view, you should also search your EGL forms for the **value** property and then determine the program changes necessary to prevent any change in behavior. This might include removing or changing the **value** property for the form fields.
- EGL supports even precision for DECIMAL fields (VAGen PACK fields) by incrementing the precision by 1 except for host variable references in SQL WHERE clauses and the EGL **prepare** statement. If you turn off VisualAge Generator compatibility mode, EGL validation does not display an error message in the Problems view. However, the program might not run the same as in VisualAge Generator. Specifically, DECIMAL fields that have even precision might be too small for the data contained in the database. In general you need to carefully evaluate each DataItem part or variable to determine whether you can safely turn off VisualAge Generator compatibility mode. In VisualAge Generator, you can search all DataItem and Record parts using the References tool for the text string *evensql = Y* (one blank before and after the = sign). This search can help you determine if you have any items or records that specified even precision. In EGL, you can search for an even precision DECIMAL field by searching for *decimal(2, decimal(4, decimal(18*. If you did not use even precision DECIMAL fields, then you can safely turn off VisualAge Generator compatibility mode. If you did use even precision DECIMAL fields, you need to consider the performance impact on your SQL access of changing to the next higher odd precision. SQL provides better performance for DECIMAL fields if your EGL host variables exactly match the precision of the SQL column definitions.

If you select the **Do not honor evensql=y for items or variables** migration preference, the migration tool automatically uses odd precision (or 18 if the item is the maximum length) and issues a warning message for the affected data item part or nonshared record item.

- EGL permits the use of the **converseVar.segmentedMode** system variable. This is the replacement for VAGen EZESEGM, which enables you to switch between segmented and nonsegmented mode in a CICS main transaction program at runtime. If you turn off VisualAge Generator compatibility mode, EGL validation displays an error message in the Problems view for each statement that uses **converseVar.segmentedMode**. In many cases, because EZESEGM is rarely used, there might not be any errors in the Problems view. If there is an error, removing the use of **converseVar.segmentedMode** might require restructuring the program to avoid the need to switch between segmented and nonsegmented mode.
- EGL permits the use of the **vgLib.getVAGSysType0** system function. The migration tool declares a variable named *customerPrefixEZESYS* and initializes it in each program to the results of **vgLib.getVAGSysType**. *customerPrefix* is the **Renaming prefix** you specify during Stage 2 migration. **vgLib.getVAGSysType** provides the original VAGen EZESYS system values (for example, MVSCICS or OS400) for use when migrating statements other than IF, WHILE, or TEST. If you turn off VisualAge Generator compatibility mode, EGL validation displays an error message in the Problems view for each program for the statement that initializes *customerPrefixEZESYS*. You can correct the problem by changing the program to remove the *customerPrefixEZESYS* declaration and the initialization statement. When you save the program, EGL validation displays an additional error message in the Problems view for each statement that uses *customerPrefixEZESYS*. To correct this error, you can change the statement to use the EGL **sysVar.systemType** system variable, which provides the new EGL system values. Depending on how *customerPrefixEZESYS* is being used, you might also need to change the values in databases, files, or DataTables from the old VAGen system values to the new EGL system values. See “EZESYS” on page 118, as well as “EZESYS state conditions” on page 336.

If you select the **Do not initialize old EZESYS values** migration preference, the migration tool automatically omits the variable declaration and initialization statement from each program. EGL validation displays an error message in the Problems view for any statement that uses *customerPrefixEZESYS*.

- EGL permits the use of the **vgLib.connectionService0** system function. This is the replacement for VAGen EZECONCT system function, which provides a variety of SQL connection services depending on the arguments you specify and your runtime environment. If you turn off VisualAge Generator compatibility mode, EGL validation displays an error message in the Problems view for each statement that uses **vgLib.connectionService0**. You can correct the error by changing to use one of the new EGL specialized system functions (for example, **sysLib.connect0**, **sysLib.disconnect0**, **sysLib.disconnectAll0**, or **sysLib.queryDatabase0**). Which EGL system function you use depends on the arguments you specified for the VAGen EZECONCT system function and your runtime environment. You should also check for any use of **vgVar.sqlIsolationLevel** because this might affect your choice of the EGL system function or the arguments that you need to specify for it.
- EGL permits the use of the **vgVar.sqlIsolationLevel** system variable. This is the replacement for VAGen EZESQISL, which is used to control the SQL isolation level in older releases of Cross System Product and VisualAge Generator for the VSE runtime environments and in VisualAge Generator 4.5 for accessing ODBC databases. If you turn off VisualAge Generator compatibility mode, EGL validation displays an error message in the Problems view for each statement that uses **vgVar.sqlIsolationLevel**. In many cases, because EZESQISL is rarely used, there might not be any error in the Problems view. If there is an error, you might be able to remove **vgVar.sqlIsolationLevel** entirely if it is not being used to control your program logic. Alternatively, if **vgVar.sqlIsolationLevel** is used

to control some of your program logic, you can replace **vgVar.sqlIsolationLevel** with a new variable that you declare in the program. Be sure to check for any use of **vgLib.connectionService()** because the behavior for that system function might depend on the value in **vgVar.sqlIsolationLevel**.

If you do turn off the VisualAge Generator compatibility mode preference, be sure to remove **vagCompatibility = "YES"** from each of your build descriptor parts. If you select the **Do not set compatibility mode** migration preference, the migration tool automatically omits **vagCompatibility = "YES"** from each build descriptor part.

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

See “Determining whether you can migrate to EGL” on page 9 for information about areas in which EGL is not a complete replacement for VisualAge Generator Developer.

See Chapter 3, “Handling ambiguous situations,” on page 65 for details about VAGen language elements or migration strategies that do not allow a precise migration to the EGL language.

See Appendix B, “Relationship of VisualAge Generator and EGL language elements,” on page 257 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 and what the program does. These differences are described in the following sections:

- General differences.
- Differences in SQL support.
- Differences in DL/I support.
- Differences in debug.
- Differences in generated COBOL.
- Differences in generated Java.
- Differences between host and workstation environments.
- 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 differences apply to text programs and print forms:
 - 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 84.

- 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 86.
- If you used different floating area specifications (including using the default specification of the full screen size) for two or more devices that have the same physical size, the location of a floating form might not be the same as in VisualAge Generator. VisualAge Generator tolerated, but did not recommend, using different floating area specifications for devices of the same size. EGL permits only one floating area specification for a physical device size.
- For workstation platforms, including debug, the following key mappings are used for textForms:

Table 68. Key mappings for textForms

3270 Keys	VAGen mappings	EGL mappings on Windows	EGL mappings on Linux and AIX
PF1–PF12	F1–F12	F1–F12	F1–F12
PF13–PF24	Alt+F1–Alt+F12	Shift+F1–Shift+F12	press Ctrl+S and then press F1–F12
PA1–PA3	Ctrl+F1–Ctrl+F3	Ctrl+F1–Ctrl+F3	press Ctrl+A and then press F1–F3

Note:

- + indicates that you must press 2 keys simultaneously.
- For Linux and AIX, the Ctrl+S and Ctrl+A work as a toggle. If you press the combination of keys by mistake, you can press them again to turn it off. Pressing Ctrl+S and then pressing a key other than F1–F12 has no effect. Similarly, pressing Ctrl+A and then pressing a key other than F1–F3 has no effect.
- 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 70 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 114 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 116 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 95 for details.

- Using a field in a DataTable with a single row of contents to initialize a structured field array in a record or an array on a form results in different initialization from VisualAge Generator. In this situation in VisualAge Generator, the source is treated as a scalar and the target array is completely initialized by the scalar source. In EGL, only the first element of the target array is initialized. Errors, including abends, can result due to uninitialized or invalid data in the target array. For details, see “MOVEA with a single row table as the source” on page 107.

Differences in SQL support

The differences in SQL support can affect program behavior in the following situations:

- If you generate COBOL, the differences between DB2 on the host and JDBC used by the debugger might affect the behavior of your program when you debug in EGL.
- If you previously generated C++ and now generate Java, the differences between DB2 or ODBC and JDBC used by Java generation might affect the behavior of your program both when you debug in EGL and when you run the program in the native workstation environment.

The following are differences in SQL behavior:

- ODBC is not supported in EGL. If you use an 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 one resource to commit or rollback without the corresponding commit or rollback for the other resource.
 - EZECONCT (EGL **vgLib.connectionService()**). In VisualAge Generator, the R option for the unit of work argument changes the connection to another database without ending the current connection. This permits 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 help for the **vgLib.connectionService()** for details.
- JDBC always runs dynamic SQL. Generated COBOL (in both VAGen and EGL) and generated C++ (in VAGen) use static SQL except when you use the VAGen Execution time statement build option (EGL **prepare** statement) or a table name host variable. Therefore, there are the following differences:
 - In dynamic mode, single row select can result in more than one row being returned without setting **sysVar.sqlData.sqlCode** to -811.
- JDBC handles code page conversion differently than DB2. If your database is on the host and includes a CHAR, DBCHAR, or MBCHAR SQL column defined as "FOR BIT DATA", DB2 does not do any code page conversion. However, JDBC does convert the data. If you have this situation, change the EGL SQL record definition to add the **asBytes = yes** property for the field that corresponds to the SQL column that is defined as "FOR BIT DATA".

- In VisualAge Generator, you use CHAR fields for SQL columns that contain DATE, TIME, or TIMESTAMP data. The SQL date/time format used by DB2 on the host might differ from that returned by JDBC. When you are debugging a program intended for the host environment or generating a Java program that uses a host DB2 database, the interface is through JDBC. Therefore, you might need to use one of the following techniques to obtain the date/time format that you require:
 - **Technique 1:** This technique uses DB2 Version 9, Fix Pack 4 or higher. Set your connection URL to include the following DB2 properties:
 - **dateFormat**
 - **timeFormat**
 - **timestampFormat**
 - **Technique 2:** This technique uses DB2 Version 8. Use the DB2 APP Driver instead of the default Universal Driver. To use the APP Driver to control the date, time, and timestamp formats, follow these steps:
 1. To use the APP Driver, set the following information:

Field	Value
Driver name	com.ibm.db2.jdbc.app.DB2Driver
Driver location	db2java.zip
Sample connection URL	jdbc:db2:MYHOSTDB

2. To set the date format, enter the following command from a DB2 Command Prompt window:

```
db2 update cli cfg for section COMMON using DateTimeStringFormat xxx
```

where xxx is EUR, ISO, USA and so on based on the DB2 date format that you want to use.

3. Confirm the setting by entering the following command:

```
db2 get cli cfg
```

4. Start your EGL developer product.

- **Technique 3:** This technique uses EGL only. Set the **sqlDataCode** property for the CHAR fields that contain SQL DATE, TIME, or TIMESTAMP columns. Use the following values for the **sqlDataCode** property:

Data type	sqlDataCode
DATE	385
TIME	389
TIMESTAMP	393

In addition, you might also need to set the following system variables so that they correspond to the date, time, and timestamp formats in DB2:

- **strLib.defaultDateFormat**
- **strLib.defaultTimeFormat**
- **strLib.defaultTimestampFormat**

You can initialize the default date/time formats either directly in your programs or by setting the following build descriptor options:

- **defaultDateFormat**
- **defaultTimeFormat**
- **defaultTimeStampFormat**

For details, refer to the *EGL Language Reference* and the *EGL Generation Guide*.

- Other changes in the EZE words related to SQL when using JDBC:
 - EZESQISL (EGL **vgVar.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.sqlData.sqlerrmc**) is not supported.
 - EZESQWN6 (EGL **sysVar.sqlData.sqlwarn[7]**) is not supported.
 - EZESQLCA (EGL **sysVar.sqlData.sqlca**) fields are limited. They do not include values for EZESQRRM and EZESQWN6.

Differences in DL/I support

The following differences occur with DL/I support:

- VisualAge Generator always uses CBLTDLI when it generates IMS or DL/I support. For CBLTDLI, the IMS or DL/I PSBs do not need to include a PCB name for the PCBs. EGL-generated programs use either CBLTDLI or AIBTDLI depending on the **callInterface** property that is specified for the program. The migration tool converts the programs to use CBLTDLI because that is what VisualAge Generator uses. However, regardless of the setting for the **callInterface** property, the EGL debug facility always uses AIBTDLI. AIBTDLI requires a PCB name for the PCBs in the IMS or DL/I PSB. Therefore, to use the EGL debug facility, you must add a PCB name for all database PCBs in your IMS or DL/I PSBs. You can add the PCB name either by specifying a label on each PCB marco or by specifying the PCBNAME parameter on each PCB macro. Because the EGL debug facility only supports debugging DL/I databases, you only need to add the PCB name for database PCBs. If you want to change your programs to use AIBTDLI, you must include the PCB name for all the PCBs in the IMS or DL/I PCB and specify the corresponding PCB name in the EGL PSB.

The migration tool attempts to prepare the EGL PSB for use by the debugger by setting the **pcbName** property to database name from the VAGen PSB. When you add the PCB name to your IMS or DL/I PSBs, you can use the database name from the VAGen PSB as a starting point to minimize the changes to the EGL PSB. However, the VAGen PSB tolerates the same database name for multiple PCBs in a PSB because the names are not used in the generated COBOL. IMS and DL/I PSBs require that the PCB names be unique. Therefore, the PCB names provided by the migration tool might not be acceptable. The migration tool issues an error message when there are multiple PCBs with the same database name in a PSB.

The following example shows a VAGen PSB, the corresponding EGL PSB produced by the migration tool, and the IMS PSB both before and after adding the PCB name. The VAGen PSB has PCBs numbered 3 and 4 with the same database name, DDDDDDDD. The migrated EGL PSB uses the database name as the value for the **pcbName** property. The IMS PSB cannot use DDDDDDDD for both PCBs, so you must use a different name DDDDDDDX and then update the corresponding PCB in the EGL PSB to specify the correct name.

PSB Example

Note:

- The PCB names are in bold
- Only the PCB names for the database PCBs must be added to use the EGL debugger

VAGen PSB

PCBs

1	TP	(alternate I/O PCB)
2	TP	(express alternate I/O PCB)
3	DB	DatabaseName = DDDDDDDD , <i>otherPCBInformation</i>
4	DB	DatabaseName = DDDDDDDD , <i>otherPCBInformation</i>
5	DB	DatabaseName = OTHERDB , <i>otherPCBInformation</i>
6	DB	DatabaseName = ELAWORK
7	GSAM	DatabaseName = GGGGGGGG

EGL PSBRecord created by the migration tool

```

iopcb IO_PCBRecord { @PCB { pcbType = PCBKind.TP } };
pcb0 IO_PCBRecord { redefines = iopcb};
ELAALT ALT_PCBRecord { @PCB {pcbType = PCBKind.TP } };
pcb1 ALT_PCBRecord { redefines = ELAALT };
ELAEXP ALT_PCBRecord { @PCB {pcbType = PCBKind.TP } };
pcb2 ALT_PCBRecord { redefines = ELAEXP };
DDDDDDDD_dbPCB DB_PCBRecord
{ @PCB { pcbType = PCBKind.DB, pcbName = "DDDDDDDD",
  otherPCBInformation
} }
pcb3 DB_PCBRecord { redefines = DDDDDDDD_dbPCB };
DDDDDDDD_4_dbPCB DB_PCBRecord
{ @PCB { pcbType = PCBKind.DB, pcbName = "DDDDDDDD",
  // pcbName in the above line must be changed to DDDDDDDX
  // to match the IMS PSB
  otherPCBInformation
} }
pcb4 DB_PCBRecord { redefines = DDDDDDDD_4_dbPCB };
OTHERDB_dbPCB DB_PCBRecord
{ @PCB { pcbType = PCBKind.DB, pcbName = "OTHERDB",
  otherPCBInformation
} }
pcb5 DB_PCBRecord { redefines = OTHERDB_dbPCB };
ELAWORK DB_PCBRecord
{ @PCB { pcbType = PCBKind.DB } }
pcb6 DB_PCBRecord { redefines = ELAWORK };
GGGGGGGG_gsamPCB GSAM_PCBRecord
{ @PCB { pcbType = PCBKind.GSAM, pcbName = "GGGGGGGG" } } ;
pcb7 GSAM_PCBRecord { redefines = GGGGGGGG_gsamPCB };

```

Original IMS PSB

```

TITLE 'PSB FOR PROCESSING SAMPLE DATABASES'
PRINT NOGEN
PCB TYPE=TP,MODIFY=YES
PCB TYPE=TP,MODIFY=YES,EXPRESS=YES
PCB TYPE=DB,DBDNAME=XXXXXXXX,otherPCBInformation
PCB TYPE=DB,DBDNAME=XXXXXXXX,otherPCBInformation
PCB TYPE=DB,DBDNAME=XXXXXXXX,otherPCBInformation
PCB WORKDB=ELAWORK
PCB TYPE=GSAM,DBDNAME=YYYYYYYY,otherPCBInformation
PSBGEN LANG=ASSEM,CMPAT=YES,otherPSBInformation
END

```

Modified IMS PSB

```

TITLE 'PSB FOR PROCESSING SAMPLE DATABASES'
PRINT NOGEN
ELAALT PCB TYPE=TP,MODIFY=YES
ELAEXP PCB TYPE=TP,MODIFY=YES,EXPRESS=YES
DDDDDDDD PCB TYPE=DB,DBDNAME=XXXXXXXX,otherPCBInformation
DDDDDDDX PCB TYPE=DB,DBDNAME=XXXXXXXX,otherPCBInformation
OTHERDB PCB TYPE=DB,DBDNAME=XXXXXXXX,otherPCBInformation

```

```

ELAWORK PCB      WORKDB=ELAWORK
GGGGGGGG PCB      TYPE=GSAM,DBDNAME=YYYYYYYY,otherPCBInformation
                PSBGEN LANG=ASSEM,CMPAT=YES,otherPSBInformation
                END

```

- In general, the same restrictions on testing IMS and DL/I programs apply to both VAGen and EGL programs:
 - The following things are supported by EGL debug:
 - Calls to a program running in the IMS/VS environment
 - Use of DL/I database I/O through the EGL I/O statements
 - The following things are not supported by EGL debug:
 - Use of DL/I database I/O if the database is located on z/OS CICS or VSE.
 - Serial files that are associated with the IMS message queue or GSAM files. For debug use **seqws** instead.
 - System functions such as AUDIT (EGL **sysLib.audit()**) or CREATX (EGL **VGLib.startTransaction()**)
 - **vgLib.VGTDLI()**, **dliLib.EGLTDLI()**, or **dliLib.AIBTDLI()** system functions to make IMS or DL/I calls that perform any of the following actions:
 - Access to the I/O PCB, TP PCBs, GSAM PCBs, or FastPath databases
 - CHKP, GSCD, PCB, TERM, or XRST calls

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:

- The following differences apply to text forms:
 - 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 83 for details.
 - One of the values specified for the **screenSizes** property for a floating text form must match one of the **screenSize** entries specified for the **@ScreenFloatingArea** property for the FormGroup. Similarly, one of the values specified for the **formSize** property for a floating print form must match one of the **pageSize** entries specified for the **@PrintFloatingArea** property for the FormGroup.
- 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** 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** state being set as it is 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 help for more information about indexed records and alternate index records and their use with **set record position**, **get next**, and **get previous**.

- Interactive Test Facility (in VisualAge Generator) uses DB2 or ODBC. Generated COBOL (in both VisualAge Generator and EGL) use DB2. Generated C++ (in VisualAge Generator) uses DB2 or ODBC. EGL debug and Java generation use JDBC. If you generate COBOL or previously generated C++, this results in differences in when you debug your program using the EGL debugger. See “Differences in SQL support” on page 233 for details.
- Interactive Test Facility (in VisualAge Generator) uses CBLTDLI. EGL debug uses AIBTDLI. For more information see “Differences in DL/I support” on page 235.
- Handling of NUM or NUMC data that contains invalid data differs from VisualAge Generator. In VisualAge Generator, if you clear the **Stop on invalid numeric data** preference, invalid data, including blanks, is tolerated by ITF for NUM and NUMC fields. If a NUM or NUMC field contains blanks, it is treated as though the value is 0. For example, if the field is compared to 0, the comparison tests true. If a NUM or NUMC field contains invalid data other than blanks, the comparison tests false. In effect, this simulates the use of the COBOL generation option /SPZERO, which specifies that blanks are tolerated in NUM and NUMC fields and are treated as though the field contains 0. Any other use of invalid data in the NUM or NUMC fields results in a runtime error, including an abend. If you do not use the /SPZERO generation option, any invalid data in an NUM or NUMC field, including blanks, results in a runtime error. The /SPZERO generation option has no effect on BIN, PACK, or PACF fields. In EGL, the **spacesZero** build descriptor option provides the same support as /SPZERO for generated COBOL programs. The EGL debugger does not have a preference that corresponds to the VAGen **Stop on invalid numeric data** preference. Instead, both the EGL debugger and Java generation honor the **spacesZero** build descriptor option. Therefore, if you use **spacesZero** for COBOL generation, you should add this option to your debug build descriptor part.

Differences in generated COBOL

The following differences occur for generated COBOL code:

- For z/OS, 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 uses CALL, DXFR, or XFER as a way of transferring to an EGL program.
 - An EGL program uses **call**, **transfer**, 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.

- For z/OS CICS, there are certain situations in which you must relink all programs in the run unit to use the IBM Rational COBOL Runtime for zSeries. For details, see “VisualAge Generator and EGL interoperability on z/OS CICS” on page 473.
- For iSeries, you must generate the programs in accordance with the rules described in “VisualAge Generator and EGL interoperability on iSeries” on page 473.
- The following differences apply to 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 83 for details.
- For z/OS Batch and IMS BMP, if you are writing print forms to a data set, be sure that you specify the following attributes:
 - LRECL=137, BLKSIZE=141, RECFM=VBA if the form group does not contain any DBCS forms
 - LRECL=654, BLKSIZE=658, RECFM=VBA if the form group contains any DBCS forms

In some cases, VisualAge Generator tolerated smaller values for LRECL and BLKSIZE. EGL requires the correct LRECL and BLKSIZE.

Differences in generated Java

The following differences occur for generated Java code:

- VAGen-generated Java programs use a **vgj.properties** file to control the runtime environment. EGL-generated Java programs use either *programName.properties* or **rununit.properties**, depending on the value of the **genProperties** build descriptor option. See “vgj.properties” on page 392 for the correspondence between the VisualAge Generator and EGL runtime properties.
- EGL generated Java programs are not fully compatible with VisualAge Generator programs.

The following program interactions are supported:

- An EGL program can call a VAGen generated Java or C++ program using a remote call. Similarly, a VAGen generated Java or C++ program can call an EGL program using a remote call.
- An EGL VGWebTransaction program can use the **show VGUIRecord** statement to indirectly transfer to a VAGen Web Transaction program. Similarly, a VAGen Web Transaction program can use the XFER with a UI record statement to indirectly transfer to an EGL VGWebTransaction program.

The following program interactions are not supported:

- An EGL program cannot call a VAGen generated Java or C++ program using a local call. Similarly, a VAGen generated Java or C++ program cannot call an EGL program using a local call.
- An EGL program cannot transfer to a VAGen program using the **transfer** or **show TextForm** statements. Similarly, a VAGen program cannot transfer to an EGL program using the DXFR or XFER statements.

Differences between host and workstation environments

If you change from generating for a host environment (such as CICS) to generating for a workstation environment (such as native AIX), you need to consider the following differences:

- The collating sequence for the host environments is EBCDIC. The collating sequence for Java generation is UNICODE. Therefore, you need to review the following areas of your code:
 - Range match valid tables
 - Specific values coded for a high-value or low-value of a key
 - Comparison of keys for a 2-file match
- Users need to know the following information:

- The mapping for the function keys because workstation keyboards do not typically have keys for PF13 - PF24 and PA1 - PA3. For the keyboard mapping, see Table 68 on page 232.
- It is not necessary to enter SO/SI characters when shifting out/into DBCS mode.
- The total length of all records passed on a call from AIX native to CICS cannot exceed 32567. This is a CICS restriction. If you previously generated for CICS and used the default parmform=COMMPTR, then the total length of all records might have exceeded this limit. If you previously used parmform=COMMDATA, then the total length of all records is within this limit.
- Also review the information in these sections:
 - “Differences between distributed CICS and native workstation environments”
 - “Differences between generated C++ and generated Java” on page 243

Differences between distributed CICS and native workstation environments

To run generated EGL code in a workstation 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 257 to determine the corresponding EGL language element.

- The following general differences occur:
 - Multiple users cannot run in the same address space on a server. Users run on their own workstations.
 - Client unit of work is not supported.
 - 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 243.
 - 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.
 - VAGen-generated programs for distributed CICS use environment variables to control the runtime environment. EGL-generated Java programs use either *programName.properties* or *rununit.properties*, depending on the value of the **genProperties** build descriptor option. See “Runtime environment variables” on page 390 for the correspondence between the VisualAge Generator environment variables and the EGL runtime properties.
- 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 CICS for z/OS environment. If you use this technique, be sure to set the EGL **eliminateSystemDependentCode** build descriptor option to YES.

- EZELOC (EGL **sysVar.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.
- The following differences occur when transferring between programs:
 - 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** and **show** are not supported to non-EGL programs.
- The following commit and rollback differences occur:
 - 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.
- The following CALL CREATX (EGL **sysLib.startTransaction()**) differences occur:
 - 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 **vgLib.connectionService()**). In VisualAge Generator, for the CICS environment, EZECONCT ignores the

password. In EGL, for native environments, **vgLib.connectionService()** uses the password. See “Differences between generated C++ and generated Java” on page 243 for additional differences due to changing to Java generation.

- The following differences occur for the EZE special data word:
 - 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 *recordName.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, *recordName.resourceAssociation* also contains the system resource name associated with a record. However, the format of the information varies depending on the runtime environment and the file type. Therefore, because you are changing both the runtime environment and the file type, you must review any use of *recordName.resourceAssociation* to ensure that the information provided by your program is correct for your native environment and file type.
 - EZEDESTP (EGL **converseVar.printerAssociation**). In VisualAge Generator, for the CICS environment, EZEDESTP contains the destination associated with the print file. In EGL, for native environments, **converseVar.printerAssociation** also contains the file name associated with the print file. However, the format of the information varies depending on the runtime environment and the file type. Therefore, because you are changing both the runtime environment and the file type, you must review any use of **converseVar.printerAssociation** to ensure that the information provided by your program is correct for your native environment and file type.
 - EZELETERM (EGL **sysVar.terminalID**). In VisualAge Generator, for the CICS environment, EZELETERM contains the CICS terminal identifier and is equivalent to EZEUSR. In EGL, for native environments, **sysVar.terminalID** is initialized from the **user.name** Java Virtual Machine system property. If this property cannot be retrieved, **sysVar.terminalID** contains blanks.
 - EZERCODE (EGL **sysVar.returnValue**). 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:
 - *RSnnnnnnn*, where *nnnnnnn* is a VAGen return code based on file access and the problem that occurred.
 - *nnnnnnnnn*, 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.
 - EZESEGT (EGL **sysVar.transactionID**). In VisualAge Generator, for the CICS environment, EZESEGT is initialized to the current transaction ID and also used to set a new transaction ID to take effect after a CONVERSE. EZESEGT

can be used to control program logic. In EGL, for native environments, EZESEGT 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 EZEELTERM. In EGL, for native environments, **sysVar.sessionID** is initialized from the **user.name** Java Virtual Machine system property. If this property cannot be retrieved, **sysVar.sessionID** 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 **user.name** Java Virtual Machine system property. 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:

- VAGen-generated C++ programs use environment variables to control the runtime environment. EGL-generated Java programs use either *programName.properties* or *rununit.properties*, depending on the value of the **genProperties** build descriptor option. See “Runtime environment variables” on page 390 for the correspondence between the VisualAge Generator environment variables and the EGL runtime properties.
- 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++.
- A call from a generated Java program to a native C++ or VAGen generated C++ program is always a remote call even if the programs are running on the same workstation. Therefore, you must add a linkage options element for this situation.
- 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 following name resolution rules for the **transfer** or **show** statements apply:
 - If the **transfer** or **show** statement explicitly specifies both the package name and program name, this is the program that is used (for example: `transfer to program mypackage.program1`).
 - If the **transfer** or **show** statement explicitly specifies only the program name, then the first of the following criteria that applies is used to resolve the program name:
 - The file containing the **transfer** or **show** statement explicitly imports the package and program (for example: `import mypackage.program1` and `transfer to program program1`).
 - The build descriptor option **programPackageName** points to the package that contains the transfer-to program.
 - The file containing the **transfer** or **show** statement imports a package that contains the transfer-to program (for example: `import mypackage.*` and `transfer to program program1`).

- The linkage options part used at generation includes a **transferToProgram** or **transferToTransaction** entry for the transfer-from program that specifies the transfer-to program and the package that contains it. For the **show** statement, the transfer link entry must be in the form of a **transferToTransaction** entry.
- The transfer-to program is in the same package as the **transfer** or **show** statement.
- If the **transfer** or **show** statement uses **sysVar.transferName** and the transfer-to program is in a different package, you must use a linkage options part or the **programPackageName** build descriptor option to specify the package that contains the transfer-to program.
- The following general differences occur:
 - 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 **resourceAssociations** build descriptor option to point to your resource associations part.
 - Set the **genProperties** build descriptor option to GLOBAL or PROGRAM. Refer to the online help for details of these build descriptor options.
- The following differences occur for text forms:
 - 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 83 for details.
 - One of the values specified for the **screenSizes** property for a floating text form must match one of the **screenSize** entries specified for the **@ScreenFloatingArea** property for the FormGroup. Similarly, one of the values specified for the **formSize** property for a floating print form must match one of the **pageSize** entries specified for the **@PrintFloatingArea** property for the FormGroup.
- 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** 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** state being set the same way 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 help for more information about indexed records and alternate index records and their use with **set record position**, **get next**, and **get previous**.
- The following differences occur for SQL:
 - In VisualAge Generator, generated C++ uses either DB2 or ODBC. EGL debug and Java generation use JDBC. This results in differences in both debug and runtime. See “Differences in SQL support” on page 233 for details.
- The following differences occur for EZE special data words:
 - 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 ELA 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.returnValue**). 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

VisualAge Generator migration tool extended reserved words

The VisualAge Generator migration tools rename data items, records, PSBs, maps, and functions when they conflict with an EGL reserved word and provide error messages when other parts conflict with an EGL reserved word. In fact, to avoid name conflicts, the migration tool renames parts or issues error messages based on an extended reserved word list. This list contains the following words:

- EGL reserved words (for example, **program**, **sql**, or **YES**)
- EGL part stereotypes (for example, **SQLRecord**, **BasicProgram**, or **TextForm**)
- EGL properties (for example, **column** or **fieldLen**)
- EGL exceptions (for example, **AnyException** or **FileIOException**)
- EGL enumeration names (for example, **colorKind** or **signKind**)
- EGL system library and system variable names (for example, **sysLib** and **sysVar**)
- EGL reserved records (for example, **IO_PCBRECORD** and **DB_PCBRECORD**)
- SQL reserved words (for example, **authorization** or **doubleprecision**)
- Additional words that the migration tool uses for specific purposes (for example, **psb** as the variable name of the **PSBRecord** for a program).

EGL reserved words

There are a large number of reserved words in 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 following words are reserved in EGL:

Table 69. EGL reserved words

Letter	Reserved words
A	absolute, add, all, and, any, as
B	bigInt, bin, bind, blob, boolean, by, byName, byPosition
C	call, case, char, clob, close, const, continue, converse, current
D	DataItem, DataTable, date, dbChar, decimal, decrement, delete, display, dli
E	else, embed, end, escape, execute, exit, extends
F	false, field, first, float, for, forEach, form, FormGroup, forUpdate, forward, freeSql, from, function
G	get, goto, group
H	handler, hex, hold
I	if, implements, import, in, inOut, inParent, insert, int, interface, interval, into, is, isa
L	label, languageBundle, last, library, like
M	matches, mbChar, money, move
N	new, next, nil, no, not, nullable, num, number, numc
O	of, onEvent, onException, open, openUI, or, otherwise, out
P	pacf, package, passing, prepare, previous, print, private, program
R	record, ref, relative, replace, return, returning, returns

Table 69. EGL reserved words (continued)

Letter	Reserved words
S	scroll, self, service, set, show, singleRow, smallFloat, smallInt, sql, sqlCondition, stack, static, string
T	this, time, timeStamp, to, transaction, transfer, true, try, type
U	unicode, update, url, use, using, usingKeys, usingPCB
W	when, where, while, with, wrap
Y	yes

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

EGL enumeration words

VisualAge Generator has a fixed list of valid values for each property. EGL provides an enumeration list that contains the valid values for the corresponding property. For example, EGL stores the valid values for the **align** property in an EGL enumeration called **AlignKind**. In addition, EGL permits the use of variables and expressions for property values. When EGL resolves a property value, EGL looks first for a variable by that name. If there is no variable by that name, then EGL looks at the enumeration that corresponds to the property to validate the property value. For example, in a form, if a variable field has the property **align** = center, EGL looks first for a variable named center within the form. If there is no variable named center within the form, EGL then uses **AlignKind.center**. Table 70 shows the EGL enumeration names, the list of valid values in the enumeration, and the parts where the property is used in a way that can cause a collision between variable names and the values in the enumeration.

In addition, there are several properties that can be set using EGL library constants and variables. Table 71 lists the EGL properties that use EGL library constants and variables for reference purposes. However, the migration tool only uses the EGL library variables and constants that are indicated by an asterisk (*) in the table.

To avoid conflicts when you write new code, you can use one of the following techniques:

- Avoid the use of variable names that match any of the list of valid values when you are defining variables that might have a collision with that name.
- Always qualify the enumeration value for a property (for example, always specify **align** = **AlignKind.center**).
- Qualify the enumeration value for a property whenever there is a conflict (for example, specify **align** = **AlignKind.center** whenever there is a field named center within a specific form).

The migration tool uses a combination of the three techniques, based on anticipated frequency of use, the properties that the migration tool actually uses, and migration performance considerations. To minimize the need to rename your parts and variables, the migration tool does the following things:

- The tool always renames parts or variables named YES or NO because these are EGL reserved words.
- The tool always renames parts or variables named PF n , where n is between 1 and 24. In effect, the migration tool treats these values as reserved words. This improves the appearance of the **helpKey** and **validationBypassKeys** properties for programs and forms without a migration performance impact.

- Within a FormGroup, the migration tool does the following things:
 - Always fully qualifies the **deviceType** property value. This improves performance by avoiding the need to review all the field names in all the forms within the FormGroup.
 - Never uses the **helpKey** or **validationBypassKeys** properties because there are no VAGen equivalent properties for a map group.
- Within a form, the migration tool only qualifies property values on the form if one or more fields within the form is named the same as one of the enumeration values, EGL constants, or EGL variables that can be used on a form. If there is a conflict for any field on the form, the migration tool fully qualifies all property values for all fields within the form with the corresponding enumeration or library name.
- Within a VGUI record, the migration tool only qualifies property values in the record if one or more fields within the record is named the same as one of the enumeration values or EGL variables that can be used in a VGUI record. If there is a conflict for any field in the record, the migration tool fully qualifies all property values for all fields within the record with the corresponding enumeration or library name.
- Within a PSBRecord, the migration tool always fully qualifies the **pcbType** property.
- Within a program, the migration tool always fully qualifies the **callInterface** property.
- The migration tool never qualifies property values within other part types because there is no possibility of a conflict between a part name and the enumeration values.

Note: Table 70 on page 251 and Table 71 on page 253 are for reference purposes. Therefore, the tables include Library, ConsoleUI, JSFHandler, Service, and Interface parts, which the migration tool never creates.

Table 70. EGL enumerations

EGL enumeration	Valid values list which imposes restrictions on variable names	Parts with possible collision
AlignKind	center, left, none, right	Form
CallingConventionKind	I4GL (lib is reserved for future use)	Library
CaseFormatKind	defaultCase, lower, upper	ConsoleUI
ColorKind	black, blue, cyan, defaultColor, green, magenta, red, white, yellow Note: black is only valid for ConsoleUI	Form ConsoleUI
ConvertDirection	local, remote	not applicable
DataSource	databaseConnection, reportData, sqlStatement	Report
DeviceTypeKind	doubleByte, singleByte	FormGroup
DisplayUseKind	button, hyperlink, input, output, secret, table	Any record used with a JSFHandler
DLICallInterfaceKind	AIBTDLI, CBLTDLI	Program

Table 70. EGL enumerations (continued)

EGL enumeration	Valid values list which imposes restrictions on variable names	Parts with possible collision
EventKind	afterDelete, afterField, afterInsert, afterOpenUI, afterRow, beforeDelete, beforeField, beforeInsert, beforeOpenUI, beforeRow, menuAction, onKey	ConsoleUI
ExportFormat	html, pdf, text, xml	Report
HighlightKind	blink, defaultHighlight, noHighlight, reverse, underline	Form ConsoleUI
IndexOrientationKind	across, down	Form
IntensityKind	bold, defaultHighlight, dim, invisible, normalIntensity	Form ConsoleUI
LineWrapKind	character, compress, word	ConsoleUI
OrderingKind	byKey, byInsertion, none	Dictionary
OutlineKind	bottom, left, right, top, box, noOutline	Form
PCBKind	TP, DB, GSAM	PSBRecord
PfKeyKind	pfn , where $(1 \leq n \leq 24)$	Form FormGroup Program
ProtectKind	noProtect, protect, skipProtect	Form
ScopeKind	application, request, session	Any record used with a JSFHandler
SelectTypeKind	index, value	Any record used with a JSFHandler
SignKind	leading, none, parens, trailing	Form VGUI record Any record used with a JSFHandler
UITypeKind	hidden, input, inputOutput, none, output, programLink, submit, submitBypass, uiForm	VGUI record
WindowAttributeKind	color, commentLine, errorLine, formLine, highlight, intensity, menuLine, messageLine, promptLine	ConsoleUI

Table 71. EGL Enumerations that use sysLib constants and sysVar variables

EGL property	EGL constants and variables (* = value used by migration tool)	Parts with possible collision
dateFormat	<ul style="list-style-type: none"> • strLib.defaultDateFormat* • strLib.eurDateFormat • strLib.isoDateFormat • strLib.jisDateFormat • strLib.usaDateFormat • vgVar.systemGregorianCalendar* • vgVar.systemJulianCalendar* <p>Note: The migration tool only uses defaultDateFormat in DataItem and VGUI record parts.</p>	ConsoleUI Form, VGUI record
fillCharacter	<ul style="list-style-type: none"> • strLib.nullFill* 	Form
timeFormat	<ul style="list-style-type: none"> • strLib.defaultTimeFormat • strLib.eurTimeFormat • strLib.isoTimeFormat • strLib.jisTimeFormat • strLib.usaTimeFormat 	Any record used with a JSFHandler
timeStampFormat	<ul style="list-style-type: none"> • strLib.db2TimeStampFormat • strLib.odbcTimeStampFormat 	ConsoleUI

SQL reserved words

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

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
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, graphic, 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

Letter	Reserved words
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, sqlcondition, 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, varGaphic, 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, use the following techniques:

- 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 **defaultSelectCondition** for a record, specify:
defaultSelectCondition = #sqlCondition{ "reservedWord" = ... }

For example:

```
defaultSelectCondition = #sqlCondition{ "FROM" = ... }
```

- To use one of the reserved words as an SQL column name 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 DataTable contents after the DataTable structure so that is the syntax shown in the Tables section. EGL syntax also permits the DataTable contents to be placed before the DataTable 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.
- [] - encloses an EGL list of values; values are always separated by commas

General syntax conventions

There are some differences in the overall syntax of VisualAge Generator and EGL.

Table 72. General syntax conventions

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<p>Comments are specified in the following formats:</p> <ul style="list-style-type: none"> • Prologs for programs, tables, and records. • Descriptions for items and functions. • Comments within functions are indicated by: <ul style="list-style-type: none"> – 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. 	<p>Comments are specified in the following formats:</p> <ul style="list-style-type: none"> • // 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. 	<p>The migration tool converts in the following way:</p> <ul style="list-style-type: none"> • Prologs and part descriptions are converted to EGL // line comments. • Descriptions for items used as fields in records, tables, maps, function local storage, function parameters, or function return values are converted to EGL // line comments. • Comments within functions are converted to /* comment */ • Informational comments added by the migration tool are in the form of EGL // line comments.
<p>Decimal point can be either a period or a comma depending on your locale.</p>	<p>Decimal point during development is always the period. Generation and runtime use either a period or a comma depending on the runtime locale and the decimalSymbol and symbolSeparator build descriptor options.</p>	<p>During migration, if your locale uses the comma for the decimal point by default or if you select the Migration Syntax Preference Convert decimal comma to decimal point, the migration tool converts the comma to a period.</p>
<p>Properties are entered in specialized editors using check boxes, drop-down lists, and so on.</p>	<p>Properties are entered in a text editor and must be separated by a comma.</p> <p>There are a few specialized editors such as the Source Assistant for DataItem parts and the EGL Form Editor for form parts.</p>	<p>No special considerations.</p>
<p>Lists of values are entered in specialized editors. For example, the table names for an SQL record are entered in the SQL Row Properties window.</p>	<p>Lists of values must be enclosed in square brackets [].</p>	<p>No special considerations.</p>
<p>Property values that reference other parts are entered in specialized editors. For example, the Edit routine for a map variable field is entered on the Edits page of the Variable Field Properties window.</p>	<p>Property values that refer to other parts or variables that must be in the naming scope for the current part are not enclosed in quotes (for example, the keyItems property in an SQL record). Property values that refer to other parts or variables that can be outside the naming scope for the current part must be enclosed in double quotes (for example, the recordNumItem property for a relative record).</p>	<p>No special considerations.</p>

Table 72. General syntax conventions (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
Resolution of names within statements is context sensitive and is based on the statement type.	Resolution of names within statements always follows the same rules regardless of the statement type.	No special considerations. Most differences in name resolution result in EGL validation messages in the Problems view. If a message occurs, see “Reference information for messages - name resolution and qualification rules” on page 450.

Data item

The data item section is organized into the following tables:

- Data item - general syntax, data type, length, decimals, and description, Table 73 on page 259
- Default map properties and User Interface properties - general information, Table 74 on page 261
- Default map properties and User Interface properties - general edits, Table 75 on page 261
- Default map properties and User Interface properties - numeric edits, Table 76 on page 263
- Default map properties and User Interface properties - error messages, Table 77 on page 264
- User Interface properties - label and help, Table 78 on page 265

Note: In EGL, there is only one set of edit and message properties for a DataItem part. The migration tool merges the map and UI properties for the data item.

Table 73. Data item — 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: <ul style="list-style-type: none"> • itemName • Basic information: <ul style="list-style-type: none"> – Data type – Length – Decimals – Description • Default Map Properties • User Interface (UI) Properties 	EGL syntax example: <pre>// Description DataItem itemName dataType(lengthInformation) { [{formattingProperties}] [{validationProperties}] [{JSFHandlerFieldProperties}] } end</pre>	<p>The migration tool uses the VAGen data type, length, and decimals to determine the EGL <i>dataType</i> and <i>lengthInformation</i>.</p> <p>The migration tool merges the VAGen default map properties and the UI properties into the single set of EGL formatting, validation, and JSFHandler field properties.</p>

Table 73. Data item — general syntax, data type, length, decimals, and description (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<p>Character item types:</p> <ul style="list-style-type: none"> • Char • Hex • DBCS • Mixed • Unicode (VisualAge for Java only) <p>Length is the number of characters. In the record editor you can also show the number of bytes.</p>	<p>Corresponding character item types:</p> <ul style="list-style-type: none"> • CHAR • HEX • DBCHAR • MBCHAR • UNICODE <p>Length is the number of characters.</p>	<p>The migration tool converts character data items to the corresponding type and length.</p>
<p>Numeric character (zoned decimal) types:</p> <ul style="list-style-type: none"> • Num • Numc <p>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.</p>	<p>Corresponding numeric types:</p> <ul style="list-style-type: none"> • NUM • NUMC <p>Precision is the total number of digits. Scale is the number of digits to the right of the decimal point.</p> <p>The maximum precision for NUM fields is 32 for debug and Java generation or 31 for COBOL generation. The maximum precision for NUMC fields is 18.</p>	<p>The migration tool converts to the corresponding type, precision, and scale. The migration tool omits the scale if decimals is 0.</p>
<p>Packed decimal types:</p> <ul style="list-style-type: none"> • Pacf • Pack <p>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 the Execution time statement build option. In the record editor you can also show the number of bytes.</p>	<p>Corresponding numeric types:</p> <ul style="list-style-type: none"> • PACF • DECIMAL <p>Precision is the total number of digits. Scale is the number of digits to the right of the decimal point.</p> <p>The maximum precision for PACF fields is 18. The maximum precision for DECIMAL fields is 32 for debug and Java generation or 31 for COBOL generation.</p> <p>The length for PACF must be odd or 18. The length for DECIMAL fields can be odd or even. Even lengths are supported for DataItem part definitions and all record types.</p> <p>At test and generation, if you use VisualAge Generator compatibility mode, EGL does the following things for DECIMAL items with even precision:</p> <ul style="list-style-type: none"> • Increases the precision by one in all records. • EGL uses a temporary variable with the even precision in SQL WHERE clauses or PREPARE statements. 	<p>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 DataItem part definition, by default the migration tool migrates it as the even length.</p> <p>If you select the Do not honor evensql=y for items or variables migration preference, the migration tool automatically uses odd precision for a Pack item (or 18 if the item is the maximum length) and issues a warning message for the affected data item part or nonshared record item.</p>

Table 73. Data item — general syntax, data type, length, decimals, and description (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<p>Binary item types:</p> <ul style="list-style-type: none"> • Bin, length 4, no decimals • Bin, length 9, no decimals • Bin, length 18, no decimals • Bin, length 4, 9, or 18 with decimals 	<p>Corresponding binary types:</p> <ul style="list-style-type: none"> • SMALLINT (no precision or scale) • INT (no precision or scale) • BIGINT (no precision or scale) • BIN with precision and scale 	<p>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.</p>
Description	Not applicable.	The migration tool converts the item description to a comment that precedes the DataItem definition.

Table 74. Default map properties and User Interface properties - general information

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<p>Data items can have both default map properties and user interface (UI) properties specified. The following kinds of properties are supported:</p> <ul style="list-style-type: none"> • formatting edits • validation edits • error messages <p>UI properties also include a label and help text.</p> <p>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.</p>	<p>DataItem parts can have the following kinds of properties:</p> <ul style="list-style-type: none"> • formatting properties • validation properties • JSFHandler field properties <p>The categories for some properties are changed from VisualAge Generator. For example, error messages are grouped with the validation properties. JSFHandler field properties include the UI label and help text. The EGL column in the following tables shows the category for the EGL property.</p>	<p>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 67 for details and potential problems.</p> <p>Also see information about map item edits for shared data items in “Map edit routine for shared data items” on page 68 for details and potential problems.</p>

Table 75. Default map properties and User Interface properties - general edits

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<p>Edit type (UI only) - values:</p> <ul style="list-style-type: none"> • None • Boolean • Date • Time 	<p>EGL supports multiple properties:</p> <ul style="list-style-type: none"> • not applicable • isBoolean = yes • dateFormat = dateFormat • timeFormat = "HH:mm:ss" <p>(formatting properties)</p>	No special considerations.
Edit function (UI only)	validatorFunction (validation property)	No special considerations.
Edit table (UI only)	validatorDataTable (validation property)	No special considerations.
Run edit function on Web (UI only)	runValidatorFromProgram	The EGL property is the reverse of the VAGen property. The migration tool converts yes to no and no to yes.

Table 75. Default map properties and User Interface properties - general edits (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<p>Edit routine (map only)</p>	<p>validatorFunction OR validatorDataTable (validation property)</p>	<p>If the UI edit function and edit table are not specified, the migration tool sets the EGL property in the following way:</p> <ul style="list-style-type: none"> • Sets the validatorFunction property if the map edit routine is EZEC10 or EZEC11. • Sets the validatorFunction property if the edit routine is a function. • Sets the validatorDataTable property if the edit routine is a table. <p>If the UI edit function or edit table are specified, the migration tool does not migrate the map Edit routine.</p> <p>Special considerations apply if the edit routine is not available during migration. See information about map edit routines in “Map edit routine for shared data items” on page 68 for additional details and potential problems.</p>
<p>Justify - Left Right None (map only)</p> <p>Note:</p> <ul style="list-style-type: none"> • For map items, the default is right for numeric fields and left for all other fields. • For UI items, justify is not supported. 	<p>align = left right none (formatting property)</p> <p>Note:</p> <ul style="list-style-type: none"> • For form fields, the default is right for numeric fields and left for all other fields. • For JSFHandler fields and VGUI record fields, align is not supported. 	<p>No special considerations.</p>
<p>Date edit mask (map only)</p> <p>The following values are valid:</p> <ul style="list-style-type: none"> • SYSGREGRN • SYSJULIAN • dateEditPattern 	<p>dateFormat = <i>value</i></p> <p>The following values are valid:</p> <ul style="list-style-type: none"> • systemGregorianCalendarFormat • systemJulianDateFormat • "dateEditPattern" <p>(formatting property)</p> <p>Note: In the <i>dateEditPattern</i>, the migration tool converts to the following EGL notation:</p> <ul style="list-style-type: none"> • yy or yyyy indicates the year. • MM indicates the month. • dd indicates the day of the month. • DDD indicates the day of the year. 	<p>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.</p>
<p>Minimum input</p>	<p>minimumInput (validation property)</p>	<p>No special considerations.</p>

Table 75. 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: <ul style="list-style-type: none"> 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: <ul style="list-style-type: none"> The same default fillCharacter is used for JSFHandler fields, form fields, and VGUI record fields unless overridden in the specific page, form, or VGUI record. strLib.nullFill is the EGL constant for the null fill character. Alternatively, use "" (two consecutive double quotes). Non-blank characters are permitted for UNICODE fields in a VGUI record. 	The migration tool converts N to one of the following values: <ul style="list-style-type: none"> N for a UI fill character nullFill for a map field character 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 70 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 76. 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.	validValues = [[<i>minimumValue</i> , <i>maximumValue</i>]] (validation property) Note: Multiple pairs of values and single values can be listed in the validValues property.	The migration tool combines the minimum and maximum value into the EGL validValues property.

Table 76. Default map properties and User Interface properties - numeric edits (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
Sign - None Leading Trailing Note: <ul style="list-style-type: none"> The default sign for numeric items in a UI record is Leading. The default sign for numeric items on a map is None. 	sign = none leading trailing (formatting property) Note: The default sign for a numeric field is always leading .	For a numeric field, the migration tool migrates based on the first of the following criteria that applies: <ul style="list-style-type: none"> If the UI sign edit is specified, the tool migrates to the corresponding sign property. If a UI edit type of date, time, or Boolean is specified, the tool sets the sign property to none. If there are any other UI edits specified, the tool sets the sign property to leading. If the map sign edit is specified, the tool migrates to the corresponding sign property. If the map sign edit is not specified, the tool sets the sign property to none.
Currency (both map and UI) Currency symbol (UI only)	currency = yes no currencySymbol = "symbol" (formatting property) Note: <ul style="list-style-type: none"> The currencySymbol also applies to forms. If currency = yes, but the currencySymbol is not specified; the actual currency symbol used at runtime is set the same way it is in VisualAge Generator. 	The migration tool migration tool migrates based on the first of the following criteria that applies: <ul style="list-style-type: none"> If the UI Currency symbol is specified, the tool migrates to currency = yes, currencySymbol = "symbol". 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 77. 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)	validatorDataTableMsgKey (validation property)	No special considerations.
EZE function (UI only)	validatorFunctionMsgKey (validation property)	No special considerations.

Table 77. Default map properties and User Interface properties - error messages (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
Edit routine (map only)	validatorDataTableMsgKey OR validatorFunctionMsgKey (validation properties)	<p>Special considerations apply. See “Shared edits and messages” on page 67 for details on how the migration tool determines whether to migrate the map edit routine message. If the migration tool migrates the map edit routine message, the tool sets the EGL property in the following way:</p> <ul style="list-style-type: none"> • Sets validatorFunctionMsgKey if the edit routine is EZEC10 or EZEC11. • Sets validatorDataTableMsgKey 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. <p>Special considerations apply. See information about ambiguous data items and map edit routines in “Map edit routine for shared data items” on page 68 for additional details and potential problems.</p>
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	validValuesMsgKey (validation property)	No special considerations.

Table 78. User Interface properties - label and help

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
UI label	displayName (JSFHandler field property)	No special considerations.
Help text	help (JSFHandler field property)	No special considerations.

Record

The record section is organized into the following tables:

- Record - general syntax, record type, properties, and prolog, Table 79 on page 266
- Record - record structure for most record types, Table 80 on page 268
- Record - SQL properties and SQL record structure, Table 81 on page 270
- Record - DL/I properties and DL/I record structure, Table 82 on page 274
- Record - UI record properties and UI record structure, Table 83 on page 276
- Record - UI item properties - general, Table 84 on page 277

- Record - UI item properties - edits, Table 85 on page 278
- Record - UI item properties - error messages, Table 86 on page 279
- Record - UI item properties - help, Table 87 on page 279
- Record - UI item properties - submit, Table 88 on page 280
- Record - UI item properties - program link, Table 89 on page 280

Note: The migration tool always converts VAGen records to EGL structured Record parts.

Table 79. Record - general syntax, record type, properties, and prolog

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<p>VAGen record part:</p> <ul style="list-style-type: none"> • recordName • Basic information <ul style="list-style-type: none"> – Record type – Record structure (item list) • Properties (vary based on record type) • Prolog <p>Note: The record structure can be given by specifying an alternate specification record or by including the item list.</p>	<p>EGL record example:</p> <pre> **** Record=recordName**** // prolog //***** Record recordName type recordType { [recordProperties] } recordStructure end // end recordName </pre> <p>Note: The record structure can be given by specifying an embed keyword or by including the item list.</p>	No special considerations.
<p>Record types:</p> <ul style="list-style-type: none"> • Working Storage • Redefined • Serial • Indexed • Relative • Message Queue • SQL Row • User Interface • DL/I Segment 	<p>EGL Record types:</p> <ul style="list-style-type: none"> • BasicRecord • BasicRecord • SerialRecord • IndexedRecord • RelativeRecord • MQRecord • SQLRecord • VGUIRecord • DLISegment 	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 70 for details and potential problems.
<p>Working storage record properties:</p> <ul style="list-style-type: none"> • Alternate specification 	<p>BasicRecord properties:</p> <ul style="list-style-type: none"> • embed keyword 	The migration tool migrates an alternate specification to the embed keyword.

Table 79. Record - general syntax, record type, properties, and prolog (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<p>Redefined record properties:</p> <ul style="list-style-type: none"> • Redefinition <p>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.</p>	<p>BasicRecord properties:</p> <ul style="list-style-type: none"> • 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. 	<p>The migration tool includes a comment with the record definition to provide the name of the record that was redefined.</p> <p>The migration tool also includes the redefines property on the declaration statement for the record in programs that use the record.</p> <p>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 70 for details and potential problems.</p>
<p>Serial record properties:</p> <ul style="list-style-type: none"> • File name • Alternate specification • Variable length item • Occurrences item 	<p>serialRecord properties:</p> <ul style="list-style-type: none"> • fileName • embed keyword • lengthItem • numElementsItem 	<p>The migration tool migrates an alternate specification to the embed keyword.</p>
<p>Indexed record properties:</p> <ul style="list-style-type: none"> • File name • Record ID • Alternate specification • Variable length item • Occurrences item 	<p>indexedRecord properties:</p> <ul style="list-style-type: none"> • fileName • keyItem • embed keyword • lengthItem • numElementsItem 	<p>The migration tool migrates an alternate specification to the embed keyword.</p>
<p>Relative record properties:</p> <ul style="list-style-type: none"> • File name • Record ID • Alternate specification 	<p>relativeRecord properties:</p> <ul style="list-style-type: none"> • fileName • recordNumItem • embed keyword 	<p>The migration tool migrates an alternate specification to the embed keyword.</p>
<p>Message Queue record properties:</p> <ul style="list-style-type: none"> • 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 	<p>mqRecord properties:</p> <ul style="list-style-type: none"> • queueName • embed keyword • includeMsgInTransaction • openQueueExclusive • lengthItem • numElementsItem • queueDescriptorRecord • openOptionsRecord • msgDescriptorRecord • getOptionsRecord • putOptionsRecord 	<p>The migration tool migrates an alternate specification to the embed keyword.</p>
<p>SQL row record properties:</p> <ul style="list-style-type: none"> • See Table 81 on page 270. 	<p>SQL row record properties:</p> <ul style="list-style-type: none"> • See Table 81 on page 270. 	<p>No special considerations.</p>

Table 79. Record - general syntax, record type, properties, and prolog (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
DL/I segment record properties: • See Table 82 on page 274.	DL/I segment record properties: • See Table 82 on page 274.	No special considerations.
UI record properties: • See Table 83 on page 276.	UI record properties: • See Table 83 on page 276.	No special considerations.
Prolog	Not applicable.	The migration tool converts the prolog to a comment that precedes the record definition.

Table 80. Record - 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 keyword specifies the record that provides the field structure for the current record. RecordA embeds RecordB. For example: embed RecordB;	The migration tool migrates an alternate specification to the embed keyword Special considerations apply for level 77 items in working storage records. See information in “Level 77 items in records” on page 71 for details and potential problems.
Record structure - variation 2 with Shared Items: • itemName • Occurs • Shared • 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: <i>levelNumber itemName itemName [occurs];</i> Note: Type, Length, Decimals and Description are <i>not</i> visible in the editor.	If you select the Convert shared data items to primitive item definitions migration syntax preference, and the data item part is available, the migration tool converts the shared item to an EGL variable that is defined using a primitive type definition based on the type, length, and decimals specified for the data item part. Migration of type, length, and decimals information is the same as described in Table 73 on page 259. If you clear the Convert shared data items to primitive item definitions migration syntax preference, or the data item part is not available, the migration tool 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. 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 71 for details and potential problems.

Table 80. Record - record structure for most record types (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<p>Record structure - variation 2 with Nonshared Items:</p> <ul style="list-style-type: none"> • itemName • Occurs • Type • Length • Decimals • Nonshared • Description • levelNumber is hidden, but is based on the data item hierarchy within the record. <p>Note: Type, Length, Decimals and Description are stored with the item in the record.</p>	<p>Record structure - variation 2 with EGL primitive types example:</p> <pre>levelNumber itemName dataType(lengthInformation) [occurs]; // Description</pre> <p>Note: Type, Length, Decimals and Description are visible in the editor.</p>	<p>The migration tool converts a nonshared item 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 73 on page 259.</p> <p>The migration tool omits the occurs information if occurs is 1.</p> <p>Special considerations apply for level 77 items in working storage records. See information in “Level 77 items in records” on page 71 for details and potential problems.</p>

Table 81. Record - SQL properties and SQL record structure

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<p>SQL record properties:</p> <ul style="list-style-type: none"> • Default key item • Alternate specification • SQL tables: <ul style="list-style-type: none"> – Label – Name <p>Note:</p> <ul style="list-style-type: none"> • 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 is substituted at run time. Table name host variables start with a semicolon (:). 	<p>SQLRecord properties:</p> <ul style="list-style-type: none"> • keyItems • embed keyword • tableNames or tableNameVariables or both <p>Note:</p> <ul style="list-style-type: none"> • The keyItems property 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 is substituted at run time. The table names in the tableNameVariables property do not start with a semicolon. The tableNames and tableNameVariables properties can both be used in the same record definition. 	<p>The migration tool builds the keyItems property in the following way:</p> <ul style="list-style-type: none"> • If the VAGen alternate specification is not specified, 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 specified, 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 properties. <p>The migration tool builds the lists for the tableNames and tableNameVariables properties in the following way:</p> <ul style="list-style-type: none"> • 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. <p>Special considerations apply. See information about SQL alternate specification in “Alternate specification records” on page 72 for details and potential problems.</p>

Table 81. Record - SQL properties and SQL record structure (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<p>SQL Default Conditions:</p> <ul style="list-style-type: none"> • whereClauseText <p>Note:</p> <ul style="list-style-type: none"> • 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. 	<p>Example of default selection conditions:</p> <pre>defaultSelectCondition = #sqlCondition{ whereClauseText }</pre> <p>Note:</p> <ul style="list-style-type: none"> • The defaultSelectCondition property is used for the same purpose as in VisualAge Generator. • !itemColumnName variables are not supported. Actual SQL column names must be used. 	<p>The migration tool converts any !itemColumnName variables to their corresponding SQL column name.</p> <p>Special considerations apply. See information about SQL alternate specification records" on page 72 for details and potential problems.</p>
<p>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.</p>	<p>Record structure - variation 1: The EGL embed keyword specifies the record that provides the field structure for the current record. RecordA embeds RecordB. For example:</p> <pre>embed RecordB;</pre>	<p>The migration tool migrates an alternate specification to the embed keyword.</p>

Table 81. Record - SQL properties and SQL record structure (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<p>Record structure - variation 2 with Shared Items:</p> <ul style="list-style-type: none"> • itemName • Read Only • Key • SQL Column Name • SQL Code • Shared <p>Note:</p> <ul style="list-style-type: none"> • Type, Length, Decimals and Description are visible in the Record Editor, but are not stored in the record. • Level numbers are never used in SQL records. • 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. 	<p>Record structure - variation 2 with EGL type definitions example:</p> <pre>levelNumber itemName itemName { [sqlDataCode=sqlCodeNumber] [column="SQLColumnName"] [isReadOnly=yes] [isSQLNullable = yes] [sqlVariableLen = yes] };</pre> <p>Note:</p> <ul style="list-style-type: none"> • Type, Length, Decimals and Description are <i>not</i> visible in the editor. • Level numbers are optional in SQL records. If level numbers are included, the SQL record is a structured record. If level numbers are not included, the SQL record is not a structured record. Non-structured records permit the use of the EGL data types of BLOB, CLOB, and STRING. However, other behavior of non-structured records is not compatible with VAGen behavior. 	<p>If you select the Convert shared data items to primitive item definitions preference, and the data item part is available, the migration tool does the following things:</p> <ul style="list-style-type: none"> • 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 as described in Table 73 on page 259. • Includes the sqlDataCode property for hex items. • Sets the sqlVariableLen property to YES for CHAR, DBCHAR, or UNICODE fields if the VAGen SQL data code indicates the item is variable length. The migration tool omits the sqlVariableLen property if the VAGen SQL data code indicates the item is fixed length. <p>If you clear the Convert shared data items to primitive item definitions preference, or the data item part is not available, the migration tool does the following things:</p> <ul style="list-style-type: none"> • 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 sqlVariableLen property to YES if the VAGen SQL data code indicates the item is variable length. The migration tool omits the sqlVariableLen property if the VAGen SQL data code indicates the item is fixed length. <p>The migration tool does the following things:</p> <ul style="list-style-type: none"> • Includes any key=yes items in the EGL keyItems property for the SQLRecord. • Always adds a level number to items in the SQLRecord so that the record is a structured record. This preserves VAGen behavior.

Table 81. Record - SQL properties and SQL record structure (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<p>Record structure - variation 2 with Nonshared Items:</p> <ul style="list-style-type: none"> • itemName • Type • Length • Decimals • Read Only • Key • SQL Column Name • SQL Code • Nonshared • Description <p>Note:</p> <ul style="list-style-type: none"> • Type, Length, Decimals and Description are stored with the item in the record. • Level numbers are never used in SQL records. • 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. 	<p>Record structure - variation 2 with EGL primitive types example:</p> <pre>levelNumber itemName dataType(lengthInformation) // Description { [sqlDataCode=sqlCodeNumber] [column="SQLColumnName"] [isReadOnly=yes] [isSQLNullable = yes] [sqlVariableLen = yes] };</pre> <p>Note:</p> <ul style="list-style-type: none"> • Type, Length, Decimals and Description are visible in the editor. • Level numbers are optional in SQL records. If level numbers are included, the SQL record is a structured record. If level numbers are not included, the SQL record is not a structured record. Non-structured records permit the use of the EGL data types of BLOB, CLOB, and STRING. However, other behavior of non-structured records is not compatible with VAGen behavior. 	<p>The migration tool converts the nonshared item 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 73 on page 259.</p> <p>The migration tool includes the sqlDataCode property only for hex items.</p> <p>The migration tool sets the sqlVariableLen property to 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 sqlVariableLen property if the VAGen SQL data code indicates that the item is fixed length.</p> <p>The migration tool does the following things:</p> <ul style="list-style-type: none"> • Includes any key=yes items in the EGL keyItems property for the SQLRecord. • Always adds a level number to items in the SQLRecord so that the record is a structured record. This preserves VAGen behavior.
<p>VAGen data type - Char</p> <ul style="list-style-type: none"> • data code - 453 • data code - 449 or 457 	<p>EGL data type:</p> <ul style="list-style-type: none"> • CHAR; omit sqlVariableLen • CHAR, with sqlVariableLen set to YES 	No special considerations.
<p>VAGen data type - DBCS</p> <ul style="list-style-type: none"> • data code - 469 • data code - 465 or 473 	<p>EGL data type:</p> <ul style="list-style-type: none"> • DBCHAR; omit sqlVariableLen • DBCHAR, with sqlVariableLen set to YES 	No special considerations.
<p>VAGen data type - Unicode</p> <ul style="list-style-type: none"> • data code - 469 • data code - 465 or 473 	<p>EGL data type:</p> <ul style="list-style-type: none"> • UNICODE; omit sqlVariableLen • UNICODE, with sqlVariableLen set to YES 	No special considerations.

Table 81. Record - SQL properties and SQL record structure (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<p>SQL Column Name</p> <p>Note: The SQL Column Name is required.</p>	<p>column = "SQLColumnName"</p> <p>Note: The column property is optional. If the column property is omitted, it defaults to the field name.</p>	<p>The migration tool migrates based on the Omit column name preference.</p> <ul style="list-style-type: none"> If you select the preference, the tool does the following things: <ul style="list-style-type: none"> Omits the column property if the SQL Column Name is the same as the field name. Includes the column property if the SQL Column Name is different from the field name. If you clear the preference, the tool always includes the column property.
<p>No corresponding property.</p> <p>Note: VisualAge Generator always includes the null indicator variable for SQL items.</p>	<p>isSQLNullable = yes no</p> <p>Note: The default for isSQLNullable is no.</p>	<p>The migration tool migrates based on the Omit isSQLNullable property preference.</p> <ul style="list-style-type: none"> If you select the preference, the tool does not include the isSQLNullable property, which defaults to no. If you clear the preference, the tool always includes isSQLNullable = yes. This preserves VAGen behavior.
<p>Read Only</p> <p>Note: Read Only is always explicitly set. Read Only must always be yes if there are multiple tables specified for the SQL record.</p>	<p>isReadOnly = YES NO</p> <p>Note: isReadOnly defaults to YES if there are multiple tables specified for the SQL record. isReadOnly defaults to NO if there is only one table specified for the SQL record.</p>	<p>The migration tool migrates based on the Omit isReadOnly property preference.</p> <ul style="list-style-type: none"> If you select the preference, the tool only includes the isReadOnly property if there is a single table specified for the record and the VAGen Read Only property is set to yes. If you clear the preference, the tool includes the isReadOnly property whenever the VAGen Read Only property is set to yes.

Table 82. Records - DL/I properties and DL/I record structure

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<p>DL/I record properties:</p> <ul style="list-style-type: none"> Key item Alternate specification Record length item <p>Note: The record name must be the same as the segment name in the DL/I PSB.</p>	<p>DLISegment properties:</p> <ul style="list-style-type: none"> keyItem embed keyword lengthItem <p>Note: EGL permits the record name to differ from the segment name in the DL/I PSB. In this situation, the EGL segmentName property provides the name used in the DL/I PSB.</p>	<p>If the migration tool renames the record due to a conflict with an EGL reserved word or because the record name starts with the # or @ symbol, the tool includes the segmentName property to provide the original record name.</p>

Table 82. Records - DL/I properties and DL/I record structure (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<p>Record structure - variation 1: Alternate specification. If RecordA provides an alternate specification of RecordB, RecordB provides all the items for RecordA. There is no item structure in RecordA.</p> <p>Note: The field names in the alternate specification must be the same as the field names in the DL/I PSB.</p>	<p>Record structure - variation 1: The EGL embed keyword specifies the record that provides the field structure for the current record. RecordA embeds RecordB. For example:</p> <pre>embed RecordB;</pre> <p>EGL permits the field names in the record to differ from the field names in the DL/I PSB. If the embedded record is not a DLISegment and the field names differ, set the dliFieldName property in the following way:</p> <pre>embed RecordB { fieldInRecordB {dliFieldName="nameInPSB"} } ;</pre>	<p>The migration tool migrates an alternate specification to the embed keyword.</p> <p>If the alternate specification record is not a DL/I segment, the tool overrides the dliFieldName property for any item that was renamed due to a conflict with an EGL reserved word or because the field name starts with the # or @ symbol.</p> <p>Special considerations apply. For details, see "Alternate specification records" on page 72.</p>
<p>Record structure - variation 2 with Shared Items:</p> <ul style="list-style-type: none"> • itemName • Occurs • Shared • levelNumber is hidden, but is based on the data item hierarchy within the record. <p>Note:</p> <ul style="list-style-type: none"> • Type, Length, Decimals and Description are visible in the Record Editor, but are not stored in the record. • The field names must be the same as the field names in the DL/I PSB. 	<p>Record structure - variation 2 with EGL type definitions example:</p> <pre>levelNumber itemName itemName [occurs] {dliFieldName="nameInPSB"};</pre> <p>Note:</p> <ul style="list-style-type: none"> • Type, Length, Decimals and Description are <i>not</i> visible in the editor. • EGL permits the field names in the record to differ from the field names in the DL/I PSB. 	<p>If you select the Convert shared data items to primitive item definitions migration syntax preference, and the data item part is available, the migration tool converts the shared item to an EGL variable that is defined using a primitive type definition based on the type, length, and decimals specified for the data item part. Migration of type, length, and decimals information is the same as described in Table 73 on page 259.</p> <p>If you clear the Convert shared data items to primitive item definitions migration syntax preference, or the data item part is not available, the migration tool 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.</p> <p>The migration tool omits the occurs information if occurs is 1.</p> <p>If an item is renamed due to a conflict with an EGL reserved word or because the name starts with the # or @ symbol, the migration tool includes the dliFieldName property.</p>

Table 82. Records - DL/I properties and DL/I record structure (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<p>Record structure - variation 2 with Nonshared Items:</p> <ul style="list-style-type: none"> • itemName • Occurs • Type • Length • Decimals • Nonshared • Description • levelNumber is hidden, but is based on the data item hierarchy within the record. <p>Note:</p> <ul style="list-style-type: none"> • Type, Length, Decimals and Description are stored in the record. • The field names must be the same as the field names in the DL/I PSB. 	<p>Record structure - variation 2 with EGL primitive types example:</p> <pre>levelNumber itemName dataType(lengthInformation) [occurs] {dliFieldName="nameInPSB"}; // Description</pre> <p>Note:</p> <ul style="list-style-type: none"> • Type, Length, Decimals and Description are visible in the editor. • EGL permits the field names in the record to differ from the field names in the DL/I PSB. 	<p>The migration tool converts a nonshared item 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 73 on page 259.</p> <p>The migration tool omits the occurs information if occurs is 1.</p> <p>If an item is renamed due to a conflict with an EGL reserved word or because the name starts with the # or @ symbol, the migration tool includes the dliFieldName property.</p>

Table 83. Records - UI record properties and record structure

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<p>UI record properties:</p> <ul style="list-style-type: none"> • General <ul style="list-style-type: none"> – UI title – Submit value item – Edit function – Run edit function on Web • Input Edit Order • Help text 	<p>VGUI record properties:</p> <ul style="list-style-type: none"> • General properties <ul style="list-style-type: none"> – title – commandValueItem – validatorFunction – runValidatorFromProgram • validationOrder • help <p>Note: The validationOrder property is specified on each item in the record.</p> <p>The following example shows a VGUI record definition:</p> <pre>Record recordName type VGUIRecord {throwNrfEofExceptions = yes, handleHardIOErrors = no, V60ExceptionCompatibility = yes, I4GLItemsNullable = no, textLiteralDefaultIsString = no, localSQLScope = yes, alias="originalVAGenName", commandValueItem=itemX, validatorFunction=functionY, runValidatorFromProgram=yes, title="Page Title", help="help text line" } recordStructure end // end recordName</pre>	<p>The EGL runValidatorFromProgram property is the reverse of the VAGen property. The migration tool converts yes to NO and no to YES.</p> <p>The migration tool always includes the following properties to preserve VAGen behavior:</p> <ul style="list-style-type: none"> • throwNrfEofExceptions • handleHardIOErrors • V60ExceptionCompatibility • I4GLItemsNullable • textLiteralDefaultIsString • localSQLScope

Table 83. Records - UI record properties and record structure (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
Not applicable.	alias	<p>If the record name conflicts with an EGL reserved word or starts with the # or @ symbol, the migration tool does the following things:</p> <ul style="list-style-type: none"> • Renames the UI record. • Sets the alias property to the original VAGen UI record name. <p>Special considerations apply. See “Reserved words and UI record names” on page 75 for additional details.</p>
<p>The record structure is similar to that of a working storage record, except there is additional information for each data item:</p> <ul style="list-style-type: none"> • UI type • UI properties 	<p>The record structure is similar to that of a basic record except that there are additional properties for each field:</p> <ul style="list-style-type: none"> • uiType • validation properties, formatting properties, and JSFHandler field properties 	No special considerations.

Table 84. Record - UI item properties - general

VisualAge Generator 4.5	EGL	Migration tool considerations
Not applicable	alias	<p>If the field name conflicts with the migration tool extended reserved word list or starts with the # or @ symbol, the migration tool does the following things:</p> <ul style="list-style-type: none"> • Renames the field. • Sets the alias property to the original field name. <p>Special considerations apply. See “Reserved words and UI record names” on page 75 for additional details.</p>
<p>UI types - values are:</p> <ul style="list-style-type: none"> • Form • Hidden • Input • Input/Output • None • Output • Program Link • Submit • Submit Bypass 	<p>uiType - values are:</p> <ul style="list-style-type: none"> • uiForm • hidden • input • inputOutput • none • output • programLink • submit • submitBypass 	No special considerations.
UI label	displayName (JSFHandler field property)	No special considerations.
<p>Array items:</p> <ul style="list-style-type: none"> • Occurrences item • Selected index item 	<p>Structured field arrays:</p> <ul style="list-style-type: none"> • numElementsItem • selectedIndexItem 	No special considerations.

Table 85. Record - UI item properties - edits

VisualAge Generator 4.5	EGL	Migration tool considerations
Edit type – values are: <ul style="list-style-type: none"> • None • Boolean • Date • Time 	EGL supports multiple properties: <ul style="list-style-type: none"> • Not applicable • isBoolean = yes • dateFormat = defaultDateFormat • timeFormat = "HH:mm:ss" (formatting properties)	No special considerations.
Edit function	validatorFunction (validation property)	No special considerations.
Run edit function on Web	runValidatorFromProgram (validation property)	The EGL property is the reverse of the VAGen property. The migration tool converts yes to NO and no to YES.
Edit table	validatorDataTable (validation property)	No special considerations.
Minimum input	minimumInput (validation property)	No special considerations.
Fill character Note: <ul style="list-style-type: none"> • The following default fill characters are used: <ul style="list-style-type: none"> – Blank for character, mixed, and numeric items. – 0 for hex items. • The fill character must be blank for DBCS and unicode items. • Null is not a valid fill character. 	fillCharacter (formatting property) Note: <ul style="list-style-type: none"> • The following default fill characters are used: <ul style="list-style-type: none"> – Blank for character, MBCHAR, and numeric items. – 0 for HEX items. • The fill character must be blank for DBCHAR. Any character is valid as the fill character for UNICODE. • null is not a valid fill character. 	No special considerations.
Fold	upperCase (formatting property)	No special considerations.
Input required	inputRequired (validation property)	No special considerations.
Check SO/SI space	needsSOSI (validation property)	No special considerations.
Currency and Currency symbol	currency = YES NO currencySymbol = "symbol"	No special considerations.
Minimum value and Maximum value Note: If either Minimum value or Maximum value is specified, both must be specified.	validValues = [[minimumValue, maximumValue]] (validation property)	The migration tool combines the Minimum value and Maximum value into the EGL validValues property.

Table 85. Record - UI item properties - edits (continued)

VisualAge Generator 4.5	EGL	Migration tool considerations
Sign - None Leading Trailing Note: The default sign for numeric items in a UI record is Leading.	sign = none leading trailing (formatting property) Note: The default sign for a numeric field is always leading .	<ul style="list-style-type: none"> For a numeric field with a UI type of hidden, input, output, or input/output, the migration tool converts the sign based on the first of the following criteria that applies: <ul style="list-style-type: none"> If the UI sign edit is specified, the tool migrates to the corresponding sign property. If a UI edit type of date, time, or Boolean is specified, the tool sets the sign property to none. The tool sets the sign property to the default value of leading. For a numeric field with any other UI type, the migration tool omits the sign property.
Separator	numericSeparator (formatting property)	No special considerations.
Zero edit	zeroFormat (formatting property)	No special considerations.

Table 86. Record -- UI item properties - error messages

VisualAge Generator 4.5	EGL	Migration tool considerations
Edit table	validatorTableMsgKey (validation property)	No special considerations.
EZE function	validatorFunctionMsgKey (validation property)	No special considerations.
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	validValuesMsgKey (validation property)	No special considerations.

Table 87. UI item properties - help

VisualAge Generator 4.5	EGL	Migration tool considerations
Help text	help (JSFHandler field property)	No special considerations.

Table 88. UI item properties - submit

VisualAge Generator 4.5	EGL	Migration tool considerations
Initial value	<p>Initializer in the following format for a field that is not an array:</p> <pre>= "initialValue"</pre> <p>Initializer in the following format for a structured field array:</p> <pre>= ["initialValue1", "initialValue2"]</pre>	No special considerations.

Table 89. Record - UI item properties - program link

VisualAge Generator 4.5	EGL	Migration tool considerations
<p>Program link information:</p> <ul style="list-style-type: none"> • Program • First UI record • Open as new window • Link parameters 	<p>Program link information:</p> <ul style="list-style-type: none"> • programName • uiRecordName • newWindow • linkParms <p>EGL combines the VAGen program link properties into a complex property in the following way:</p> <pre>@programLinkData { programName = "PRGA", uiRecordName = "MYUI", newWindow = yes [, linkParmsInfo] }</pre> <p>Note: See the next row in the table for the details of the optional <i>linkParmsInfo</i>.</p>	No special considerations.
<p>Link parameters:</p> <ul style="list-style-type: none"> • Item in the First UI record • Value <ul style="list-style-type: none"> – Literal – Item in the current record <p>Note:</p> <ul style="list-style-type: none"> • When using the program link customizations for a form UI type, data within the current form is automatically moved by name to the First UI record of the next program. Additional fields within the First UI record can be initialized by listing them in the Link parameters. • When using the program link customizations for a program link UI type, only fields in the First UI record that are explicitly listed in the Link parameters are initialized. 	<p>EGL combines the VAGen link parameters into a complex property:</p> <pre>linkParms = [@LinkParameter { name = "item1InFirstUI", value = "literal" }, @LinkParameter { name = "item2InFirstUI", valueRef = itemInCurrentUI}]</pre> <p>Note: EGL follows the same rules as VisualAge Generator for program link customizations for both the uiForm and the programLink UI types.</p>	No special considerations.

Tables

The VAGen tables section is organized into the following tables:

- VAGen tables - general syntax, table type, properties, and prolog, Table 90 on page 281
- VAGen tables - table structure, Table 91 on page 282
- VAGen tables — table contents, Table 92 on page 283

Table 90. Tables — general syntax, table type, properties, and prolog

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
VAGen table part: <ul style="list-style-type: none"> • tableName • Basic information <ul style="list-style-type: none"> – Table type – Table structure (item list) • Properties • Prolog • Table Contents <p>Note: Table Contents are optional.</p>	EGL syntax example: <pre> /*** DataTable=tableName***/ // prolog /******* DataTable tableName type tableType { [otherTableProperties] [alias = "originalTableName"] } tableStructure [{ contents = [{rowContents}] }] end // end tableName </pre> <p>Note: The contents property is required.</p>	The migration tool does not rename tables for you even if the name conflicts 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 the information about table names in “Reserved words and table names” on page 76 for details.
Table types: <ul style="list-style-type: none"> • Unspecified • Match Invalid • Match Valid • Range Match Valid • Message 	DataTable types: <ul style="list-style-type: none"> • basicTable • matchInvalidTable • matchValidTable • rangeChkTable • msgTable 	No special considerations.
Properties — Runtime attributes: <ul style="list-style-type: none"> • Resident • Shared 	DataTable properties: <ul style="list-style-type: none"> • 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 91. Tables — Table structure

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<p>VAGen Table structure - with Shared Items:</p> <ul style="list-style-type: none"> • itemName • Shared • levelNumber is hidden, but it is based on the data item hierarchy within the record. <p>Note: Type, Length, Decimals and Description are visible in the Table Editor, but are not stored in the table.</p>	<p>DataTable structure - with EGL type definitions:</p> <pre>levelNumber itemName itemName ;</pre> <p>Note: Type, Length, Decimals, and Description are <i>not</i> visible in the editor.</p>	<p>If you select the Convert shared data items to primitive item definitions migration syntax preference, and the data item part is available, the migration tool 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. Migration of type, length, and decimals information is the same as described in Table 73 on page 259.</p> <p>If you clear the Convert shared data items to primitive item definitions migration syntax preference, or the data item part is not available, the migration tool 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.</p>
<p>VAGen Table structure — with Nonshared Items:</p> <ul style="list-style-type: none"> • itemName • Type • Length • Decimals • Nonshared • Description • levelNumber is hidden, but it is based on the data item hierarchy within the table. <p>Note: Type, length, decimals, and description are stored with the item in the table.</p>	<p>DataTable structure — with EGL primitive types:</p> <pre>levelNumber itemName dataType(lengthInformation) ; // Description</pre> <p>Note: Type, Length, Decimals, and Description are visible in the editor.</p>	<p>The migration tool converts a nonshared item 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 73 on page 259.</p>

Table 92. Tables — table contents

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<p>Table contents:</p> <ul style="list-style-type: none"> 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. 	<p>DataTable contents:</p> <ul style="list-style-type: none"> The contents of each row are 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. <p>Example:</p> <pre>contents = [[rowContents] { , [rowContents] }] where rowContents = value { , value }</pre>	<p>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.</p> <p>The migration tool also encloses character data, including hex data, in double-quotes.</p>

Map groups

The map groups section is organized into the following tables.

- Map Groups — general information, Table 93 on page 283
- Map Groups — general syntax and floating areas, Table 94 on page 284
- Map Groups — device names, types, and sizes, Table 95 on page 286

Table 93. Map Groups — general information

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<p>The map group part is only required if there are floating areas.</p> <p>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.</p>	<p>The FormGroup is required.</p>	<p>The migration tool creates a FormGroup part if one does not exist in the migration set.</p>
<p>Map names consist of a map group name and a map name.</p>	<p>The form name does not include the FormGroup name.</p> <p>A form can be defined (nested) within a FormGroup.</p> <p>Alternatively, a form can be outside the FormGroup. In this case, the FormGroup must include a use statement to specify the form name and an import statement to import the package in which the form is 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 FormGroups.</p>	<p>The migration tool migrates all maps to forms. The tool does not attempt to identify common, identical map definitions across multiple map groups.</p> <p>If you migrate in single file mode, the migration tool includes a use statement for each form within a FormGroup. You should move the forms so that they are nested within the FormGroup.</p> <p>If you migrate using Stage 1 – 3 migration, the migration tool automatically nests all forms within the FormGroup.</p>

Table 93. Map Groups — general information (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<p>When a program specifies a map group, the program can use any map within the map group just by referencing the map name.</p> <p>VisualAge Generator only considers the maps that are explicitly referenced in the program in a CONVERSE, DISPLAY, or CLOSE I/O option, in an XFER statement, as a program parameter, or the first map when resolving an unqualified field name.</p>	<p>When a program includes a use statement to indicate which FormGroup it is using, the program can reference any form within the FormGroup just by referencing the form name.</p> <p>If the use statement specifies the FormGroup without listing the specific forms, EGL considers all the forms in the FormGroup when resolving an unqualified field name in the program.</p> <p>If the use statement specifies forms within the FormGroup statement, EGL considers only the specified forms in the FormGroup when resolving an unqualified field name in the program.</p>	<p>The migration tool creates the use statement so that it only lists the specific forms referenced in the program.</p>

Table 94. Map Groups — general syntax and floating areas

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<p>The map group part can contain the following information:</p> <ul style="list-style-type: none"> • Map group name • Floating area information <ul style="list-style-type: none"> – Device name – Device size – Size <ul style="list-style-type: none"> - Lines - Columns – Position <ul style="list-style-type: none"> - Starting line - Starting column 	<p>The FormGroup can contain the following information:</p> <ul style="list-style-type: none"> • FormGroup name • FormGroup properties • Screen floating area information • Print floating area information • use statements for the forms that are included in the FormGroup. <p>The following example shows the format of a FormGroup:</p> <pre> FormGroup groupName { [alias="generationName"] [ScreenFloatingAreas = [@ScreenFloatingArea { screenFloatingAreaInfo }] ,] [PrintFloatingAreas = [@PrintFloatingArea { printFloatingAreaInfo }] ,] } Form formName type TextForm {formProperties} [variableFields] [constantFields] end // end formName use formName2; end // end groupName </pre>	<p>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).</p> <p>See Table 95 on page 286 for information about determining whether the device is display or printer.</p>

Table 94. Map Groups — general syntax and floating areas (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
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 77 for details and potential problems.
<p>Floating area information:</p> <ul style="list-style-type: none"> • Device name • Device size (rows x columns) • Floating area specification <ul style="list-style-type: none"> – Size <ul style="list-style-type: none"> - Lines - Columns – Position <ul style="list-style-type: none"> - Starting line - Starting column <p>Note:</p> <ul style="list-style-type: none"> • In VisualAge Generator, you define the size and starting position of the floating area. • Different floating area specifications are permitted for devices that have the same size. 	<p>Floating area information:</p> <ul style="list-style-type: none"> • Device size • Margin information <p>Print floating area information also includes the device type.</p> <p>Here is an example of the screen floating area that is used for text forms:</p> <pre>ScreenFloatingAreas = [@ScreenFloatingArea { screenSize=[lines,columns], topMargin=nn, bottomMargin=nn, leftMargin=nn, rightMargin=nn }]</pre> <p>Here is an example of the print floating area that is used for print forms:</p> <pre>PrintFloatingAreas = [@PrintFloatingArea { deviceType=singleByte, pageSize=[lines,columns], topMargin=nn, bottomMargin=nn, leftMargin=nn, rightMargin=nn }]</pre> <p>Note: Only one floating area specification is permitted for a screenSize or pageSize.</p>	<p>The migration tool uses the VAGen device type to determine whether the floating area specification is for display maps (screenFloatingArea) or print maps (printFloatingArea).</p> <p>The migration tool computes the margin information in the following way:</p> <ul style="list-style-type: none"> • 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. <p>See Table 95 on page 286 for information about determining whether the device is a display or printer.</p>
<p>Printer types:</p> <ul style="list-style-type: none"> • Printer • DBCS printer 	<p>deviceType=singleByte doubleByte</p> <p>Note: The deviceType property is only specified for print forms.</p>	<p>The migration tool sets the EGL deviceType property based on the VAGen printer type.</p> <p>The migration tool always qualifies the deviceType value with DeviceTypeKind (for example, deviceType = DeviceTypeKind.doubleByte). This avoids any name conflicts with variable fields on forms within the FormGroup.</p>

Table 95. 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	No special considerations.
3277-1	12 x 40	Display	No special considerations.
3643-4	16 x 64	Display	No special considerations.
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.
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	No special considerations.
5550D	24 x 80	DBCS Display	No special considerations.
3767, PRINT-B, PRINTER	255 x 132	Printer	For the @printFloatingArea complex property, the EGL deviceType = singleByte
5550P	255 x 158	DBCS Printer	For the @printFloatingArea complex property, the EGL deviceType = doubleByte

Maps

The maps section is organized into the following tables.

- Maps — general information, Table 96 on page 286
- Display maps — general syntax, map type, and properties, Table 97 on page 288
- Printer maps — general syntax, map type, and properties, Table 98 on page 290
- Map constant and variable fields — general information, Table 99 on page 291
- Map constant and variable fields — general syntax, data type, length, decimals, and description, Table 100 on page 293
- Map constant and variable fields — attributes, Table 101 on page 296
- Map variable fields — general edits, Table 102 on page 298
- Map variable fields — numeric edits, Table 103 on page 299
- Map variable fields — error messages, Table 104 on page 300

Table 96. Maps — general information

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
There are two types of maps: <ul style="list-style-type: none"> • Display maps • Printer maps 	There are two types of forms: <ul style="list-style-type: none"> • Text forms • Print forms 	No special considerations.

Table 96. 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.	<p>The form name does not include the FormGroup name.</p> <p>A form can be defined (nested) within a FormGroup.</p> <p>Alternatively, a form can be outside the FormGroup. In this case, the FormGroup must include a use statement to specify the form name and an import statement to import the package in which the form is 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 FormGroups.</p>	<p>The migration tool migrates all maps to forms. The tool does not attempt to identify common, identical map definitions across multiple map groups.</p> <p>If you migrate in single file mode, the migration tool includes a use statement for each form within a FormGroup. You should move the forms so that they are nested within the FormGroup.</p> <p>If you migrate using Stage 1 – 3 migration, the migration tool automatically nests all forms within the FormGroup.</p>
<p>When a program specifies a map group, the program can use any map within the map group just by referencing the map name.</p> <p>VisualAge Generator only considers the maps that are explicitly referenced in the program in a CONVERSE, DISPLAY, or CLOSE I/O option, in an XFER statement, as a program parameter, or the first map when resolving an unqualified field name.</p>	<p>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.</p> <p>If the use statement specifies the FormGroup without listing the specific forms, EGL considers all the forms in the FormGroup when resolving an unqualified field name in the program.</p> <p>If the use statement specifies forms within the FormGroup statement, EGL considers only the specified forms in the FormGroup when resolving an unqualified field name in the program.</p>	<p>The migration tool creates the use statement so that it only lists the specific forms referenced in the program.</p>

Table 97. Display maps — general syntax, map type, and properties

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<p>Display maps can contain the following information:</p> <ul style="list-style-type: none"> • Map group name and map name • Map properties <ul style="list-style-type: none"> – General properties <ul style="list-style-type: none"> - Help map name - Help key - Bypass keys - Variable field folding – Layout properties <ul style="list-style-type: none"> - Map size - Starting position - Floating map – Devices <ul style="list-style-type: none"> - Type (Display or Print) - Supported devices • Constant fields • Variable fields • Field edit order for variable fields 	<p>Text form parts can contain the following information:</p> <ul style="list-style-type: none"> • Form name • Form type • Form properties • Constant fields • Variable fields • Validation order for variable fields <p>The following example shows the format of a text form created by the migration tool:</p> <pre>Form mapName type TextForm { screenSize=[sizeList], formSize=[24,80], position=[1,1], helpForm="helpFormName", helpKey=pf1, validationBypassKeys=[pf3], msgField="VAGEN_EZMSG"} [variableFields] [constantFields] end // end mapName</pre>	<p>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).</p> <p>See Table 95 on page 286 for information about determining whether the device is a display or printer.</p>
Help map name	helpForm	No special considerations.
Help key	helpKey	No special considerations.
Bypass keys	validationBypassKeys	No special considerations.
You can specify a maximum of 5 Bypass keys for a map.	You can specify a maximum of 5 validationBypassKeys for a form.	
Variable field folding	Not supported for a form. Each CHAR or MBCCHAR variable field on the form must specify whether the data the user enters is to be automatically converted to upper case.	<p>The migration tool does the following things:</p> <ul style="list-style-type: none"> • If Variable field folding is specified for the entire map, the migration tool sets the upperCase property to YES for every CHAR and MBCCHAR field. • If Variable field folding is not specified for the entire map, the migration tool uses the Fold information specified for each CHAR and MBCCHAR field to determine whether to set the upperCase property for that field.
Map size — Lines and Columns	formSize = [Lines, Columns]	No special considerations.
Starting position - Line and Column NEXT,SAME is required if the map is a floating map.	<p>position = [Line, Column]</p> <p>If the position property is omitted, the form is a floating form</p>	If Floating map is selected, the migration tool omits the position property.
Floating map	Not applicable. If the position property is omitted, the form is a floating form.	If Floating map is selected, the migration tool omits the position property.

Table 97. Display maps — general syntax, map type, and properties (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
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 = <code>[[<i>Lines</i>, <i>Columns</i>], [<i>Lines</i>, <i>Columns</i>]]</code> Note: Include a [<i>Lines</i> , <i>Columns</i>] 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.
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 the migration tool extended reserved word list or because the map name starts with the # or @ symbol. The migration tool also includes the alias property for a map in the help map group if the map has to be renamed due to a conflict with the name of a map in the main map group for the program. Special considerations apply. See “Map names and help map names” on page 79 for details.

Table 98. Printer maps — general syntax, map type, and properties

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<p>Printer maps can contain the following the information:</p> <ul style="list-style-type: none"> • Map group name and map name • Map properties <ul style="list-style-type: none"> – General properties <ul style="list-style-type: none"> - Help map name - Help key - Bypass keys - Variable field folding - SO/SI take position – Layout properties <ul style="list-style-type: none"> - Map size - Starting position - Floating map – Devices <ul style="list-style-type: none"> - Type (Display or print) - Supported devices • Constant fields • Variable fields • Field edit order for variable fields 	<p>Print forms can contain the following information:</p> <ul style="list-style-type: none"> • Form name • Form properties • Constant fields • Variable fields <p>The following example shows the format of a print form created by the migration tool:</p> <pre>Form mapName type printForm {formSize=[255,158], position=[1,1], addSpaceForSOSI=yes } [variableFields] [constantFields] end // end mapName</pre>	<p>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).</p> <p>The migration tool always omits the following properties for print forms:</p> <ul style="list-style-type: none"> • General properties <ul style="list-style-type: none"> – Help map name – Help key – Bypass keys – Variable field folding • Devices <ul style="list-style-type: none"> – Supported devices • Field edit order for variable fields <p>See Table 95 on page 286 for information about determining whether the device is a display or printer.</p>
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	formSize = [Lines, Columns]	No special considerations.
Starting position - Line and Column NEXT,SAME is required if the map is a floating map.	<p>position = [Line, Column]</p> <p>If the position property is omitted, the form is a floating form.</p>	If Floating map is selected, the migration tool omits the position property.
Floating map	Not applicable. If the position property is omitted, the form is a floating form.	If Floating map is selected, the migration tool omits the position property.
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.

Table 98. Printer maps — general syntax, map type, and properties (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
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 the migration tool extended reserved word list or because the map name starts with the # or @ symbol. The migration tool also includes the alias property for a map in the help map group if the map has to be renamed due to a conflict with the name of a map in the main map group for the program. Special considerations apply. See “Map names and help map names” on page 79 for details.

Table 99. 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 in one of the following ways: <ul style="list-style-type: none"> • as a variable field • as a constant field • as 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 , skipProtect , 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: <ul style="list-style-type: none"> • 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.
Constant fields on printer maps can have attributes that do not really apply to printers. For example: <ul style="list-style-type: none"> • 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.

Table 99. Map constant and variable fields — general information (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<p>Variable fields on printer maps can specify attributes that do not really apply to printers. For example:</p> <ul style="list-style-type: none"> • Color • Intensity • Highlighting other than underscore • Protection • Initial cursor field • Input required • Require fill on input • Numeric attribute • Modified data tag • Light pen detect 	<p>Variable fields on print forms cannot specify properties that to not make sense for a print form.</p>	<p>The migration tool omits properties for variable fields on print forms if the properties are not supported.</p>
<p>Variable fields on printer maps can specify edits that do not really apply to printers. For example:</p> <ul style="list-style-type: none"> • Edit routine • Minimum input • Fold • Hex edit • Input required • Minimum value • Maximum value • Edit messages 	<p>Variable fields on print forms cannot specify properties that to not make sense for a print form.</p>	<p>The migration tool omits properties for variable fields on print forms if the properties are not supported.</p>

Table 100. 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
<p>A variable field on a map can contain the following information:</p> <ul style="list-style-type: none"> • Name • Information based on what you dropped on the map: <ul style="list-style-type: none"> – Data type – Position • Basic information: <ul style="list-style-type: none"> – Description – Initial value – Length in bytes – Array index – Numeric edit • Attributes • Edits, including number of decimals • Error messages <p>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.</p>	<p>A variable field on a form can contain the following information:</p> <ul style="list-style-type: none"> • 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 <p>In general, VAGen map and EGL form information corresponds in the following ways:</p> <ul style="list-style-type: none"> • 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. <p>Here is an example of an EGL variable field:</p> <pre>itemName dataType(lengthInformation) // description { position=[row,column], fieldLen=length, validationOrder=n, [presentationProperties] [formattingProperties] [value="initialValue"] [arrayInformation] }</pre> <p>Note: position is the position of the attribute byte. fieldLen is the length of the field in bytes, excluding the attribute byte. The primitive information given in <i>dataType(lengthInformation)</i> is the length of the data value.</p>	<p>The migration tool sets the EGL fieldLen property to the VAGen length in bytes. The tool sets the <i>lengthInformation</i> for the <i>dataType</i> in the following way:</p> <ul style="list-style-type: none"> • For CHAR, DBCHAR, and MBCHAR fields, migration tool sets the <i>lengthInformation</i> 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 things: <ul style="list-style-type: none"> – 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. <p>Special considerations apply. For more information, see the following sections:</p> <ul style="list-style-type: none"> • “Numeric variable fields” on page 81 • “Unnamed map variable fields” on page 85 • “Fields at row=0, column=0” on page 86

Table 100. 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
<p>A constant field on a map can contain the following information:</p> <ul style="list-style-type: none"> Information based on what you dropped on the map: <ul style="list-style-type: none"> Data type Position Basic information: <ul style="list-style-type: none"> Initial value Length in bytes Attributes <p>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.</p>	<p>A constant field on a form can contain the following information:</p> <ul style="list-style-type: none"> Position Field length Presentation properties Value <p>In general, VAGen map and EGL form information corresponds in the following ways:</p> <ul style="list-style-type: none"> VAGen attributes correspond to EGL presentation properties. Attributes that apply only to input editing are not supported for EGL constant fields. <p>The data type for a constant is determined based on the value property.</p> <p>Here is an example of an EGL constant field:</p> <pre>{ position=[row,column], fieldLen=length, [presentationProperties] [value="initialValue"] }</pre> <p>Note: position is the position of the attribute byte. fieldLen is the length of the field in bytes, excluding the attribute byte.</p>	<p>The migration tool sets the EGL fieldLen property to the VisualAge Generator Length.</p> <p>Special considerations apply. For more information, see the following sections:</p> <ul style="list-style-type: none"> “Unprotected map constants” on page 86 “Fields at row=0, column=0” on page 86
<p>Data type:</p> <ul style="list-style-type: none"> Character constant Character variable DBCS constant DBCS variable Mixed constant Mixed variable Character variable with the Numeric edit selected <p>Note: The type is determined based on the type of field you drop on the map and whether you select the Numeric edit box.</p>	<p>EGL data type:</p> <ul style="list-style-type: none"> Not applicable CHAR Not applicable DBCHAR Not applicable MBCHAR NUM 	<p>No special considerations.</p>
Description	Not applicable.	The migration tool converts the description to a comment that follows the data type and length information.

Table 100. 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: <ul style="list-style-type: none"> 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 the field in the program when the program starts. 	No special considerations.
Length	An EGL variable field has the following characteristics: <ul style="list-style-type: none"> A length, which is the number of characters or digits in the field. A fieldLen property, 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 property. Special considerations apply for numeric fields. See “Numeric variable fields” on page 81 for details.
Array index Note: <ul style="list-style-type: none"> 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. 	<pre> itemName datatype(lengthInfo) [arraySize] { propertiesForIndex1 , indexOrientation = across down, columns = n1, linesBetweenRows = n2, spacesBetweenColumns = n3, this[n] { propertiesForIndexN } } </pre> <pre> itemName datatype(lengthInfo) [arraySize] { propertiesForIndex1, this[n] { positionForIndexN, propertiesForIndexN } } </pre> Note: <ul style="list-style-type: none"> The array size is specified immediately after the <i>datatype</i> and <i>lengthInfo</i>. You can override cursor location, and presentation properties such as color, highlight, intensity, and protect. You can also override the value property. 	For standard arrays, the migration tool uses the indexOrientation , columns , linesBetweenRows , and spacesBetweenColumns properties to provide position information. The tool only includes this[n] if the cursor location or presentation properties of an array element differ from the first element of the array. For nonstandard arrays, the migration tool includes this[n] for each element of the array after the first to provide the position = [row,column] property. The tool also includes the cursor location or presentation properties for an array element if they differ from the first element of the array.

Table 100. 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: <ul style="list-style-type: none"> Field Edit Order is specified from the Define menu. 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 property if it is not included in the External Source Format for the map.

Table 101. Map constant and variable fields — attributes

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
Intensity: <ul style="list-style-type: none"> Normal Dark Bright 	intensity: <ul style="list-style-type: none"> normalIntensity invisible bold <p>(presentation property)</p>	No special considerations.
Highlight: <ul style="list-style-type: none"> No highlight Blink Reverse video Underscore 	highlight: <ul style="list-style-type: none"> noHighlight blink reverse underline <p>(presentation property)</p>	No special considerations.
Protection: <ul style="list-style-type: none"> Unprotected Protected Autoskip 	protect: <ul style="list-style-type: none"> noProtect protect skipProtect <p>(presentation property)</p>	No special considerations.
Color: <ul style="list-style-type: none"> Mono Blue Red Pink Green Turquoise Yellow White 	color: <ul style="list-style-type: none"> defaultColor blue red magenta green cyan yellow white <p>(presentation property)</p>	No special considerations.

Table 101. Map constant and variable fields — attributes (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
Initial cursor field	cursor = yes no (form field property)	No special considerations.
Input required	inputRequired (validation property)	The migration tool merges the VAGen Input required attribute and the Input required edit in the following way: <ul style="list-style-type: none"> • If either the Input required attribute or the Input required edit is selected, the migration tool includes the inputRequired property. • If neither is selected, the migration tool omits the inputRequired property.
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 things: <ul style="list-style-type: none"> • Includes the isDecimalDigit property for CHAR fields. • Omits the isDecimalDigit property for numeric fields. EGL provides a software edit for numeric fields to maintain compatibility with VAGen. <p>See “Map fields and the numeric hardware attribute” on page 83 for additional details.</p>
Modified data tag	modified (presentation property)	No special considerations.
Light pen detect	detectable (presentation property)	No special considerations.
Outlining: <ul style="list-style-type: none"> • left • right • over • under • box 	outline: <ul style="list-style-type: none"> • left • right • top • bottom • box <p>(presentation property)</p>	No special considerations.

Table 102. Map variable fields — general edits

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<p>Edit routine</p>	<p>validatorFunction OR validatorDataTable (validation property)</p>	<p>The migration tool does the following things:</p> <ul style="list-style-type: none"> • Sets the validatorFunction property if the map edit routine is EZEC10 or EZEC11. • Sets the validatorFunction property if the edit routine is a function. • Sets the validatorDataTable property if the edit routine is a table. <p>Note: Special considerations apply if the edit routine is not available during migration. See “Map variable fields and edit routines” on page 82 for additional details and potential problems.</p>
<p>Justify - Left Right None</p> <p>Note: For map items, the default is right for numeric fields and left for all other fields.</p>	<p>align = left right none (formatting property)</p> <p>Note: For form fields, the default is right for numeric fields and left for all other fields.</p>	<p>No special considerations.</p>
<p>Date edit mask</p> <p>The following values are available:</p> <ul style="list-style-type: none"> • SYSGREGRN • SYSJULIAN • dateEditPattern 	<p>dateFormat = value</p> <p>The following EGL values correspond to the VAGen values:</p> <ul style="list-style-type: none"> • systemGregorianCalendar • systemJulianDateFormat • "dateEditPattern" <p>(formatting property)</p> <p>Note: In the <i>dateEditPattern</i>, the migration tool converts to the following EGL notation:</p> <ul style="list-style-type: none"> • yy or yyyy indicates the year. • MM indicates the month. • dd indicates the day of the month. • DDD indicates the day of the year. 	<p>No special considerations.</p>
<p>Minimum input</p>	<p>minimumInput (validation property)</p>	<p>No special considerations.</p>
<p>Fill character</p> <p>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.</p>	<p>fillCharacter (formatting property)</p> <p>Note:</p> <ul style="list-style-type: none"> • The default fill character for items used on a map is null for CHAR, DBCHAR, or MBCHAR fields; blank for numeric fields; and 0 for HEX fields. • strLib.nullFill is the EGL constant for the null fill character. Alternatively, use "" (two consecutive quotation marks). 	<p>No special considerations.</p>

Table 102. Map variable fields — general edits (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
Fold	upperCase (formatting property)	The migration tool does the following things: <ul style="list-style-type: none"> • If Variable field folding is specified for the entire map, the migration tool sets the upperCase property to YES for every CHAR and MBCHAR field. • If Variable field folding is not specified for the entire map, the migration tool uses the Fold information specified for each CHAR and MBCHAR field to determine whether to set the upperCase property for that field.
Hex edit	isHexDigit (validation property)	No special considerations.
Input required	inputRequired (validation property)	The migration tool merges the VAGen Input required attribute and the Input required edit in the following way: <ul style="list-style-type: none"> • If either the Input required attribute or the Input required edit is selected, the migration tool includes the inputRequired property. • If neither is selected, the migration tool omits the inputRequired property.
Check SO/SI space	needsSOSI (validation property)	No special considerations.

Table 103. 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.	validValues = [[<i>minimumValue</i> , <i>maximumValue</i>]] (validation property) Note: Multiple pairs of values and single values can be listed in the validValues property.	The migration tool combines the Minimum value and Maximum value into the EGL validValues property.
Sign = None Leading Trailing Note: The default sign for numeric items on a map is None.	sign = none leading trailing (formatting property) Note: The default sign for a numeric field is always leading .	For a numeric field, the migration tool migrates the sign based on the first of the following criteria that applies: <ul style="list-style-type: none"> • If the map sign edit is specified, the tool migrates to the corresponding sign property. • If the map sign edit is not specified, the tool sets the sign property to none.

Table 103. Map variable fields — numeric edits (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
Currency	currency = YES NO currencySymbol = "symbol" (formatting property) Note: If currency = YES but the currencySymbol is not specified, the actual currency symbol used at runtime is set the same way it is in VisualAge Generator.	The migration tool only sets currency to YES or NO. The tool never sets currencySymbol = "symbol" for form variable fields because there was no equivalent information in VisualAge Generator.
Separator	numericSeparator (formatting property)	No special considerations.
Zero edit	zeroFormat (formatting property)	No special considerations.

Table 104. Map variable fields — error messages

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
Edit routine	validatorFunctionMsgKey OR validatorDataTableMsgKey (validation properties)	The migration tool migrates the edit routine message in the following way: <ul style="list-style-type: none"> • Sets validatorFunctionMsgKey if the edit routine is EZEC10 or EZEC11. • Sets validatorDataTableMsgKey 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 “Map variable fields and edit routines” on page 82 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	validValuesMsgKey (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 105 on page 301
- Programs - program specifications, properties, tables and additional records list Table 106 on page 303
- Programs - main functions and flow statements Table 107 on page 307

Table 105. Programs — general syntax, program type, called parameters, and prolog

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<p>Program part:</p> <ul style="list-style-type: none"> • <code>programName</code> • Program type • Specifications (vary based on program type): <ul style="list-style-type: none"> – 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 <ul style="list-style-type: none"> – Main Functions – Flow Statements (hidden in the structure diagram, but can be specified for any main function) 	<p>EGL syntax sample:</p> <pre> /** *** Program=<code>programName</code> // prolog //***** Program <code>programName</code> type <code>eglProgramType</code> //vagenProgramType [(<code>calledParameters</code>)] { [alias= "<code>originalProgramName</code>"] includeReferencedFunctions =yes, allowUnqualifiedItemReferences =yes, throwNrfEofExceptions=yes, handleHardIOErrors=no V60ExceptionCompatibility = yes, I4GLItemsNullable = no, textLiteralDefaultIsString = no, localSQLScope=yes, [<code>propertiesBasedOnType</code>] } [<code>dataDeclarations</code>] [<code>useDeclarations</code>] function main () { functionLabel: functionName () ; [{<code>functionFlowStatements</code>}] } end // end main end // end <code>programName</code> </pre>	<p>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 87.</p> <p>The migration tool includes the VAGen program type as a comment in the program definition.</p> <p>The migration tool migrates the Tables and Additional Records list in the following way:</p> <ul style="list-style-type: none"> • Tables migrate to <code>dataDeclarations</code>. • Records migrate to <code>useDeclarations</code>. <p>The migration tool always includes the following properties to preserve VAGen behavior:</p> <ul style="list-style-type: none"> • includeReferencedFunctions • allowUnqualifiedItemReferences • throwNrfEofExceptions • handleHardIOErrors • V60ExceptionCompatibility • I4GLItemsNullable • textLiteralDefaultIsString • localSQLScope
<p>Programs types:</p> <ul style="list-style-type: none"> • Main transaction • Called Transaction • Main Batch • Called Batch • Web Transaction <p>Note: See later row on "Main Transaction Execution Mode Values" for additional details.</p>	<p>EGL program types:</p> <ul style="list-style-type: none"> • <code>textUIProgram</code> • <code>textUIProgram</code> • <code>basicProgram</code> • <code>basicProgram</code> • <code>VGWebTransaction</code> 	<p>The migration tool includes the VAGen program type as a comment in the program definition. See Table 106 on page 303 for information on how the segmentation values correspond to EGL properties.</p>

Table 105. Programs — general syntax, program type, called parameters, and prolog (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<p>Called Parameters</p> <p>Note:</p> <ul style="list-style-type: none"> • 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. • Two special function words are supported as parameters: <ul style="list-style-type: none"> – EZEDLPSB – EZEDLPCB[<i>n</i>], where <i>n</i> is a numeric literal <p>See the next rows of this table for details.</p>	<p>EGL called parameters example:</p> <pre>(parameterName typeInfo { , parameterName typeInfo })</pre> <p>Note:</p> <ul style="list-style-type: none"> • Parameters must be separated by commas. • A parameter can be a DataItem, record, or form. There is no direct correspondence to the VAGen parameter types. • The EGL <i>typeInfo</i> can be: <ul style="list-style-type: none"> – a primitive type for a DataItem – a type definition for a DataItem, Record, or Form. 	<p>The migration tool includes the original VAGen parameter type as a comment.</p> <p>If you select the Convert shared data items to primitive item definitions migration syntax preference, and the data item part is available, the migration tool converts the shared item to an EGL parameter that is declared using a primitive definition based on the type, length, and decimals specified for the data item part. Migration of type, length, and decimals information is the same as described in Table 73 on page 259.</p> <p>If you clear the Convert shared data items to primitive item definitions migration syntax preference, or the data item part is not available, the migration tool converts the shared item to an EGL parameter that is declared using a type definition. For migration, the type definition is always the same as the item name.</p> <p>Special considerations apply. See “Redefined records” on page 70 for details and potential problems.</p>
Called parameter: EZEDLPSB	<p>EGL called parameter example to pass the PSB:</p> <p>parameter list:</p> <pre>(psbData PSBDataRecord)</pre> <p>program properties:</p> <pre>{ @DLI { psb = psb, psbParm = psbData }}</pre>	No special considerations.

Table 105. Programs — general syntax, program type, called parameters, and prolog (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
Called parameter: EZEDLPCB[<i>n</i>] where <i>n</i> is a numeric literal	<p>EGL called parameter example to pass the PCBs:</p> <pre>parameter list: (pcbName pcbType_PCBRecord) program properties: { @DLI { psb = psb, pcbParms = [pcbList] } }</pre> <p>Note:</p> <ul style="list-style-type: none"> The <i>pcbList</i> is used to match the name of the PCB parameter to its corresponding position within the PSBRecord for the program. The values for <i>pcbType</i> are: <ul style="list-style-type: none"> IO ALT DB GSAM 	<p>The migration tool always uses a <i>pcbName</i> in the form: <i>pcbn</i>, where <i>n</i> is the same numeric literal used in VAGen called parameter list.</p> <p>If the PSB part specified by the program is available, the migration tool does the following things:</p> <ul style="list-style-type: none"> Includes the <i>pcbType</i> information. Includes the <i>pcbList</i> information to associate each PCB parameter with the corresponding PCB in the EGL PSBRecord. <p>Special considerations apply if the PSB part for the program is not available. For details, see “Program with EZEDLPCB in called parameter list” on page 91.</p>
Prolog	Not applicable.	The migration tool converts the prolog to a comment that precedes the program definition.

Table 106. Programs — program specifications, properties, tables and additional records list

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<p>General information:</p> <ul style="list-style-type: none"> Some VAGen properties and specifications migrate to EGL properties, data declarations, or use declarations. 	<p>General information:</p> <ul style="list-style-type: none"> The rows that follow indicate whether the corresponding EGL language element is a program property, data declaration, or use declaration. 	No special considerations.
<p>Main Transaction Execution Mode values:</p> <ul style="list-style-type: none"> Nonsegmented Segmented Single segment <p>Note: Called transactions always run in nonsegmented mode.</p>	<p>segmented — values:</p> <ul style="list-style-type: none"> segmented = no segmented = yes segmented = yes <p>(program property)</p> <p>Note: The segmented property is not specified for called programs.</p>	<p>If the segmented information is not in the External Source Format file, the migration tool sets the segmented property to the default value of NO.</p>

Table 106. Programs — program specifications, properties, tables and additional records list (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<p>Working Storage record (Specifications)</p> <ul style="list-style-type: none"> The Working Storage record can be specified for both main and called programs. It is sometimes referred to as the primary working storage record for the program. The primary working storage record is always initialized. 	<p>inputRecord (program property)</p> <ul style="list-style-type: none"> The inputRecord property can only be specified for main programs. The input record is always initialized. A data declaration is also required. 	<p>The migration tool converts the primary working storage record to the inputRecord property for main programs.</p> <p>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.</p> <p>If the primary working storage record contains level 77 items, the migration tool includes a data declaration statement for the level 77 record.</p> <p>See “Level 77 items in records” on page 71 for details and potential problems.</p>
<p>PSB (Specifications)</p> <p>Note: In VisualAge Generator, the PSB is a part type.</p>	<p>EGL uses both program properties and a record declaration to specify a PSB:</p> <p>program properties: <code>{ @DLI { psb = psbName, callInterface = DLICallInterfaceKind.CBLTDLI, handleHardDLIErrors = yes }}</code></p> <p>program record declaration: <code>psbName psbRecordName ;</code></p> <p>Note: In EGL, the PSB is a stereotype of the Record part.</p>	<p>The migration tool always uses <i>psb</i> as value for the psb property. This ensures that any function that needs to reference a variable in the PSB can qualify the variable with <i>psb</i>.</p> <p>The migration tool always includes the following properties for an IMS or DL/I program to preserve VAGen behavior:</p> <ul style="list-style-type: none"> callInterface handleHardDLIErrors <p>The migration tool includes a declaration of a variable called <i>psb</i> and sets <i>psbRecordName</i> to the name of the VAGen PSB part after any required renaming.</p>
Firstmap (Specifications)	inputForm (program property)	No special considerations.
First UI Record (Specifications)	inputUIRecord (program property)	No special considerations.
Map Group (Specifications)	<p>use <i>FormGroup.formName</i> {, <i>FormGroupName</i>}</p> <p>Note: use <i>FormGroup</i> is also permitted. In this case, all forms within the <i>FormGroup</i> are used to resolve unqualified variable names in the program.</p>	<p>The migration tool includes the following maps:</p> <ul style="list-style-type: none"> Maps used in CONVERSE, DISPLAY, and CLOSE I/O options. Maps used in an XFER with a map statement. Maps listed in the called parameter list for the program. The map specified as the First Map for the program.
Help Map Group (Specifications)	use <i>FormGroup</i> { helpGroup=yes } (use declaration)	No special considerations.

Table 106. Programs — program specifications, properties, tables and additional records list (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool 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 88 for details and potential problems.
<p>Keys assignment:</p> <ul style="list-style-type: none"> • Help key (1 key) • Bypass keys (up to 5 keys) • F1–12 = F13–24 <p>(Program property)</p> <p>Note: The keys assignment is specified once for the program and applies to both the map group and the help map group.</p>	<p>EGL keys assignment example:</p> <pre>use FormGroup { [helpGroup = yes,] helpKey = <i>pfNumber</i>, validationBypassKeys = [<i>pfNumberList</i>], pfKeyEquate = yes no } ;</pre> <p>(use declaration properties for the FormGroup and help FormGroup for the program)</p> <p>Note:</p> <ul style="list-style-type: none"> • The values in the validationBypassKeys list must be separated by commas. • The validationBypassKeys property is not specified in the use declaration for the help FormGroup for the program. 	<p>The migration tool includes the EGL equivalent of the keys assignment information on the use declaration statements for both the FormGroup and the help FormGroup. The migration tool omits the validationBypassKeys property from the use declaration for the help FormGroup.</p>

Table 106. Programs — program specifications, properties, tables and additional records list (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<p>Tables and Additional Records:</p> <ul style="list-style-type: none"> Records <p>Note:</p> <ul style="list-style-type: none"> 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. 	<p>EGL additional record example:</p> <pre>recordName recordName [{ redefines = otherRecord}];</pre> <p>(data declaration)</p> <p>Note:</p> <ul style="list-style-type: none"> The redefines property must be specified when the record is declared in a program 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. 	<p>The migration tool always uses the same record name as the type definition.</p> <p>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 70 for details and potential problems.</p> <p>The migration tool also includes data declarations for all records that are used as I/O objects by the program.</p> <p>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.</p> <p>If the program specifies a PSB, the migration tool includes data declarations for all DL/I segments used as I/O objects or in the hierarchical path to an I/O object.</p> <p>For Web transaction programs, the migration tool includes data declarations for UI records that are referenced in a CONVERSE I/O statement or in an XFER statement.</p>
<p>Tables and Additional Records:</p> <ul style="list-style-type: none"> Tables You can specify Keep After Use for each table. 	<p>EGL use declaration example:</p> <pre>use tableName [{deleteAfterUse = yes}];</pre> <p>(use declaration)</p>	<p>The migration tool converts tables on the Tables and Additional Records list to use declarations.</p> <p>The deleteAfterUse property has the opposite meaning from the VAGen Keep After Use. The migration tool reverses yes and no.</p> <p>If you select the Do not include deleteAfterUse for tables migration preference, the migration tool automatically omits the deleteAfterUse property and issues a warning message for the affected program and table.</p>

Table 107. Programs — main functions and flow statements

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<p>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.</p> <p>Each main function can have flow statements. These statements do not appear in the Structure Diagram, but can be accessed from the diagram.</p>	<p>EGL programs specify only one main function. This function is always named <i>main</i>.</p> <p>There are no flow statements.</p> <p>The following example shows the syntax for an EGL main function:</p> <pre>function main () { functionLabel: functionName () ; [{ functionFlowStatements }] } end // end main</pre>	<p>The migration tool builds the EGL main function. The tool includes the following information within the EGL main function for each VAGen main function:</p> <ul style="list-style-type: none"> • <i>functionLabel</i> so that the VAGen main function can be referenced in an EGL exit stack <i>functionLabel</i> statement. The tool always sets the <i>functionLabel</i> to the <i>functionName</i>. • function invocation statement to invoke the VAGen main function. • flow statements, if any, for the VAGen main function. <p>See the following sections for details on the migration of flow statements:</p> <ul style="list-style-type: none"> • See “Statements” on page 325 • See “EZE words” on page 339 • See “Service routines” on page 349

Functions

The following tables compare the VAGen function part with the EGL function 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 108 on page 308
- Functions - EXECUTE I/O option, Table 109 on page 310
- Functions - I/O options for maps and UI records, Table 110 on page 311
- Functions - I/O for files or databases — general information and I/O error routine, Table 111 on page 311
- Functions - I/O options for serial, indexed, relative, and message queue records, Table 112 on page 312
- Functions - I/O options for default (unmodified) SQL statements, without Execution time statement build, Table 113 on page 313
- Functions - I/O options for modified SQL statements, without Execution time statement build, Table 114 on page 315
- Functions - I/O options for SQL statements with Execution time statement build, Table 115 on page 318
- Functions - I/O options for default (unmodified) DL/I statements, Table 116 on page 321
- Functions - Segment Search Arguments for modified DL/I statements, Table 117 on page 322

Table 108. Functions — general syntax, description, parameters, return value, and local storage

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<p>Function parts can contain the following information:</p> <ul style="list-style-type: none"> • Function name • Function parameters • Function return value • Function local storage • Properties: <ul style="list-style-type: none"> – Error routine – Description • I/O option • I/O object • SQL statement • DL/I call • Statements before the I/O option • Statements after the I/O option <p>Note: Only one I/O option is permitted in a function.</p>	<p>Functions can contain the following information:</p> <ul style="list-style-type: none"> • functionName • functionParameterList • returnItemType • dataDeclarations • Statements before the I/O statement • I/O statement • Statements after the I/O Statements <p>The following example shows the format of a function created by the migration tool:</p> <pre>// Description Function functionName (functionParameterList) [returns(returnItemType)] [dataDeclarations] [beforeStatements] [IOStatement] [afterStatements] end // end functionName</pre> <p>Note:</p> <ul style="list-style-type: none"> • The VAGen I/O option, I/O object, error routine, SQL statement, and DL/I call are used to create the EGL <i>IOStatement</i>. • More than one <i>IOStatement</i> is permitted in a function. 	<p>The migration tool converts all VAGen function parts to EGL standalone function parts.</p> <p>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.</p> <p>See the following sections for details on the migration of before and after statements:</p> <ul style="list-style-type: none"> • For statements, see “Statements” on page 325. • For EZE words, see “EZE words” on page 339. • For service routines, see “Service routines” on page 349. <p>See the following tables for details on I/O options and error routines:</p> <ul style="list-style-type: none"> • See Table 109 on page 310 for the EXECUTE I/O option. • See Table 110 on page 311 for I/O options for maps and UI records. • See Table 111 on page 311 general information on file and database I/O. • See Table 112 on page 312 for I/O options for serial, indexed, relative, and message queue records. <p>For details on SQL statements, see the following tables in this section:</p> <ul style="list-style-type: none"> • See Table 113 on page 313 for I/O options for unmodified SQL statements. • See Table 114 on page 315 for I/O options for modified SQL statements without Execution time statement build. • See Table 115 on page 318 for I/O options for SQL statements with Execution time statement build. <p>For details on DL/I calls, see the following tables in this section:</p> <ul style="list-style-type: none"> • See Table 116 on page 321 for I/O options for default (unmodified) DL/I statements. • See Table 117 on page 322 for Segment Search Arguments for modified DL/I statements.

Table 108. Functions — general syntax, description, parameters, return value, and local storage (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
Description	Not applicable	The migration tool converts the function description to a comment that precedes the Function definition.
<p>Function parameters:</p> <ul style="list-style-type: none"> Function parameters are entered in a special window. Items used as function parameters can be shared or nonshared. The definition for nonshared items is stored in the function. 	<p>Function parameters:</p> <ul style="list-style-type: none"> Parameters must be separated by commas. Each parameter has type information. Optionally, each parameter has parameter type information. <p>The following example shows the format of function parameters:</p> <pre>(parameterName typeInfo [parameterType] { , parameterName typeInfo [parameterType] })</pre> <p>The following specific example shows function parameters:</p> <pre>(parmSharedItem parmSharedItem field, parmNonSharedItem char(10) sqlNullable, parmRecord parmRecord)</pre>	<p>The migration tool sets the <i>typeInfo</i> in the following way:</p> <ul style="list-style-type: none"> For a record, the <i>typeInfo</i> is a type definition that specifies the same record name. If the item type is one of the VAGen Any* types, the <i>typeInfo</i> is the corresponding EGL special item type. If the item is a shared data item, then the migration tool does the following things: <ul style="list-style-type: none"> If you select the Convert shared data items to primitive item definitions migration syntax preference, and the data item part is available, the migration tool converts the shared item to an EGL function parameter that is declared using a primitive definition based on the type, length, and decimals specified for the data item part. Migration of type, length, and decimals information is the same as described in Table 73 on page 259. If you clear the Convert shared data items to primitive item definitions migration syntax preference, or the data item part is not available, the migration tool converts the shared item to an EGL function parameter that is declared using a type definition. For migration, the type definition is always the same as the item name. If the item is a nonshared data item, then the <i>typeInfo</i> is migrated based on the item type, length, and decimals, and follows the rules described in Table 73 on page 259.
<p>Function parameters:</p> <ul style="list-style-type: none"> Function parameter types: <ul style="list-style-type: none"> Record Item Map item SQL item 	<p>Function parameters:</p> <ul style="list-style-type: none"> Function parameter types: <ul style="list-style-type: none"> Not applicable Not applicable field sqlNullable 	
<p>Function parameters:</p> <ul style="list-style-type: none"> Special item types, length is not specified: <ul style="list-style-type: none"> AnyChar AnyDBCS AnyMix AnyHex AnyUnicode AnyNumeric 	<p>Function parameters:</p> <ul style="list-style-type: none"> Special item types, length is not specified: <ul style="list-style-type: none"> CHAR DBCHAR MBCHAR HEX UNICODE number 	
<p>Function return value:</p> <ul style="list-style-type: none"> Data type Length Decimals Description 	<p>EGL returns value:</p> <ul style="list-style-type: none"> The following is an example of the returns statement format: <pre>returns(returnItemType) // Description</pre>	<p>If the function includes a return value, the migration tool migrates the data type, length, and decimals based the rules described in Table 73 on page 259.</p>

Table 108. Functions — general syntax, description, parameters, return value, and local storage (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<p>Function local storage:</p> <ul style="list-style-type: none"> 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. 	<p>Function variable declarations:</p> <ul style="list-style-type: none"> Function variable declarations must include variable names and their associated type information. The following example shows the format of function variable declarations: <pre>// Function Declarations variableName typeInfo ; { variableName typeInfo ; }</pre> 	<p>The migration tool sets the <i>typeInfo</i> in the following way:</p> <ul style="list-style-type: none"> For a record, the <i>typeInfo</i> is a type definition that specifies the same record name. If the item is a shared data item, then the migration tool does the following things: <ul style="list-style-type: none"> If you select the Convert shared data items to primitive item definitions migration syntax preference, and the data item part is available, the migration tool converts the shared item to an EGL variable that is declared using a primitive definition based on the type, length, and decimals specified for the data item part. Migration of type, length, and decimals information is the same as described in Table 73 on page 259. If you clear the Convert shared data items to primitive item definitions migration syntax preference, or the data item part is not available, the migration tool converts the shared item to an EGL variable that is declared using a type definition. For migration, the type definition is always the same as the item name. If the item is a nonshared data item, then the <i>typeInfo</i> is migrated based on the item type, length, and decimals, and follows the rules described in Table 73 on page 259.
<p>Function local storage:</p> <ul style="list-style-type: none"> Function local storage types: <ul style="list-style-type: none"> Record Item 	<p>Function local storage:</p> <ul style="list-style-type: none"> Function local storage types: <ul style="list-style-type: none"> Not applicable Not applicable 	

Table 109. Functions — EXECUTE I/O option

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<ul style="list-style-type: none"> I/O object: none I/O option: EXECUTE 	No equivalent statement.	The migration tool eliminates the EXECUTE I/O option.

Table 110. Functions — I/O options for maps and UI records

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<ul style="list-style-type: none"> I/O object: mapName I/O option: DISPLAY <p>Note: DISPLAY is used for both display and printer maps.</p>	<p>To display a text form, use the display statement. To print a print form, use the print statement.</p> <p>The following examples show a display statement and a print statement:</p> <pre>display mapName;</pre> <pre>print mapName;</pre> <p>Note: In VisualAge Generator compatibility mode, display printForm is treated as though it is print printForm.</p>	<p>The migration tool converts to the display or print statement based on the map type. See “DISPLAY I/O option for maps” on page 94 for details and potential problems.</p>
<ul style="list-style-type: none"> I/O object: mapName I/O option: CONVERSE 	<p>Use the converse statement.</p> <p>The following example shows a converse statement:</p> <pre>converse mapName;</pre>	<p>No special considerations.</p>
<ul style="list-style-type: none"> I/O object: UIRecordName I/O option: CONVERSE 	<p>Use the converse statement.</p> <p>The following example shows a converse statement:</p> <pre>converse UIRecordName;</pre>	<p>No special considerations.</p>

Table 111. Functions — I/O for files or databases — general information and I/O error routine

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<p>VAGen record I/O:</p> <ul style="list-style-type: none"> I/O option I/O object (always a record) I/O error routine (optional) <p>Note: The record can be a serial, indexed, relative, message queue, SQL row, or DL/I segment record.</p>	<p>EGL record I/O:</p> <ul style="list-style-type: none"> An I/O statement Record name try ... onException statements with error routine name (optional) <p>If an I/O error routine is specified, the statements are enclosed within a try block. The following example shows file or database I/O with an error routine:</p> <pre>try add recordName ; [onException error-routine ;] end</pre>	<p>The migration tool does the following things:</p> <ul style="list-style-type: none"> Changes the VAGen I/O option to the corresponding EGL I/O statement. Includes the try ... onException statements if there is a VAGen error routine.

Table 111. Functions — I/O for files or databases — general information and I/O error routine (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<p>An error routine is optional for functions that do file or database I/O.</p> <p>Note: The error routine is invoked if there is a soft error or if EZEFECE = 1.</p>	<p>An error routine is optional for functions that do I/O for records. The following example shows I/O without an error routine:</p> <pre>add recordName;</pre> <p>The following example shows I/O with an error routine:</p> <pre>try add recordName ; onException error-routine ; end</pre> <p>Note: The onException statement is invoked if there is a soft error or if vgVar.handleHardIOErrors is set to 1. If the function does DL/I I/O, the onException statement is also invoked if vgVar.handleHardDLIErrors is set to 1.</p>	<p>The migration tool does the following things:</p> <ul style="list-style-type: none"> • If the error-routine is not specified, the tool does not include the try ... onException statements. • If an error-routine is specified, the tool includes the try block. • 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 the I/O error routine for a function, the <i>mainFunctionLabel</i> is always the same as the <i>mainFunctionName</i>.
<p>Error routine values:</p> <p>EZECLOS EZEFL0 EZERTN mainFunctionName nonmainFunctionName</p>	<p>onException block statements:</p> <pre>onException exit program; onException exit stack; Omit the onException statement. onException exit stack mainFunctionLabel; onException nonmainFunctionName();</pre>	<p>Special considerations apply for the migration of error routines that are function names. See “I/O error routine” on page 95 for details and potential problems.</p> <p>Special considerations also apply for SQL I/O functions that specify Execution time statement build. For more information, see “SQL I/O and Execution time statement build” on page 102.</p>

Table 112. 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
<ul style="list-style-type: none"> • I/O object: recordName • I/O option: ADD 	<p>Use the add statement. The following is an example:</p> <pre>add recordName;</pre>	No special considerations.
<ul style="list-style-type: none"> • I/O object: recordName • I/O option: SCAN 	<p>Use the get next statement. The following is an example:</p> <pre>get next recordName;</pre>	No special considerations.
<ul style="list-style-type: none"> • I/O object: recordName • I/O option: SCANBACK 	<p>Use the get previous statement. The following is an example:</p> <pre>get previous recordName;</pre>	No special considerations.
<ul style="list-style-type: none"> • I/O object: recordName • I/O option: CLOSE 	<p>Use the close statement. The following is an example:</p> <pre>close recordName;</pre>	No special considerations.
<ul style="list-style-type: none"> • I/O object: recordName • I/O option: INQUIRY 	<p>Use the get statement. The following is an example:</p> <pre>get recordName;</pre>	No special considerations.

Table 112. 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
<ul style="list-style-type: none"> I/O object: recordName I/O option: UPDATE 	Use the get forUpdate statement. The following is an example: get recordName forUpdate ;	No special considerations.
<ul style="list-style-type: none"> I/O object: recordName I/O option: DELETE 	Use the delete statement. The following is an example: delete recordName;	No special considerations.
<ul style="list-style-type: none"> I/O object: recordName I/O option: REPLACE 	Use the replace statement. The following is an example: replace recordName;	No special considerations.

Table 113. 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
<ul style="list-style-type: none"> I/O object: recordName I/O option: ADD 	Use the add statement. The following is an example: add recordName;	No special considerations.
<ul style="list-style-type: none"> I/O object: recordName I/O option: SCAN 	Use the get next statement. The following is an example: get next recordName;	No special considerations.
<ul style="list-style-type: none"> I/O object: recordName I/O option: CLOSE 	Use the close statement. The following is an example: close recordName;	No special considerations.
<ul style="list-style-type: none"> 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 . The following example shows a get statement without single row select: get recordName; The following example shows single row select: get recordName singleRow ;	If Single row select is specified in VisualAge Generator, the migration tool includes the EGL singleRow option.
<ul style="list-style-type: none"> 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 <i>resultSetID</i> when migrating an UPDATE I/O option for an SQL record. The tool sets the <i>resultSetID</i> to the function name followed by a customer-specified suffix. You can control the suffix with the Stage 2 VAGen Migration Preferences. Special considerations apply if the migration tool cannot determine if the record is SQL or non-SQL. See “SQL I/O with multiple UPDATE or SETUPD functions” on page 104 for details and potential problems.
<ul style="list-style-type: none"> I/O object: recordName I/O option: DELETE 	Use the delete statement. The following is an example: delete recordName;	No special considerations.

Table 113. 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
<ul style="list-style-type: none"> I/O object: recordName I/O option: REPLACE <p>(with or without UPDATE/SETUPD functionName)</p>	<p>Use the replace statement. The following are some examples:</p> <pre>replace recordName; replace recordName from resultSetID;</pre>	<p>If the UPDATE/SETUPD function name was included in VisualAge Generator, the migration tool includes the <i>resultSetID</i> and sets the <i>resultSetID</i> to the UPDATE/SETUPD function name followed by a customer-specified suffix. You can control the suffix with the Stage 2 VAGen Migration Preferences.</p>
<ul style="list-style-type: none"> I/O object: recordName I/O option: SETINQ <p>(with and without Declare cursor with hold)</p>	<p>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:</p> <pre>open resultSetID for recordName; open resultSetID hold for recordName;</pre>	<p>The migration tool sets the <i>resultSetID</i> to the function name followed by a customer-specified suffix. You can control the suffix with the Stage 2 VAGen Migration Preferences.</p> <p>If Declare cursor with hold is selected in VisualAge Generator, the migration tool includes the EGL hold option.</p>
<ul style="list-style-type: none"> I/O object: recordName I/O option: SETUPD <p>(with and without Declare cursor with hold)</p>	<p>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 statements:</p> <pre>open resultSetID forUpdate for recordName; open resultSetID hold forUpdate for recordName;</pre>	<p>The tool sets the <i>resultSetID</i> to the function name followed by a customer-specified suffix. You can control the suffix with the Stage 2 VAGen Migration Preferences.</p> <p>If Declare cursor with hold is selected in VisualAge Generator, the migration tool includes the EGL hold option.</p>
<ul style="list-style-type: none"> I/O object: recordName I/O option: SQLEXEC <p>with Model SQL Statement</p> <p>Note:</p> <ul style="list-style-type: none"> The SQL record name is included in this form of SQLEXEC. The values for the model type are: <ul style="list-style-type: none"> 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 no errors occur at runtime. 	<p>Use the execute statement. The following is an example:</p> <pre>execute modelType for recordName;</pre> <p>Note: <i>modelType</i> is either update or delete.</p>	<p>The migration tool sets the EGL <i>modelType</i> based on the VAGen Model SQL Statement value.</p> <p>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 statements based on the I/O error routine for the function.</p>

Table 114. 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
<p>General Information for modified SQL statements:</p> <ul style="list-style-type: none"> VisualAge Generator builds the table clause from the SQL row record at test and generation time. The following table clauses are used: <ul style="list-style-type: none"> insert into <i>sqlTableName</i> for the ADD I/O option from <i>sqlTableName</i> <i>sqlTableLabel</i> for the INQUIRY, UPDATE, SETINQ, and SETUPD I/O options update <i>sqlTableName</i> 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. SQL clauses are written in SQL syntax. 	<p>General Information for modified SQL statements:</p> <ul style="list-style-type: none"> 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 following table clauses are used: <ul style="list-style-type: none"> insert into <i>sqlTableName</i> for the add statement. from <i>sqlTableName</i> <i>sqlTableLabel</i> for the get and open statements. update <i>sqlTableName</i> 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. SQL clauses are written in SQL syntax. 	<p>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.</p> <p>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.</p> <p>The migration tool converts any !itemColumnName variables to their corresponding SQL column name.</p> <p>The migration tool converts VAGen comments (/*) to SQL comments (—)</p> <p>Special considerations apply if the SQL record and its alternate specification record, if any, are not available during migration. For more information and potential problems, see the following sections:</p> <ul style="list-style-type: none"> “SQL I/O statements” on page 96 “SQL I/O and missing required SQL clauses” on page 98 “SQL I/O and !itemColumnName” on page 103
<ul style="list-style-type: none"> I/O object: recordName I/O option: ADD <p>Clauses that can be modified:</p> <ul style="list-style-type: none"> INSERTCOLNAMES VALUES 	<p>Use the add statement. The following is an example:</p> <pre>add recordName with #sql{ insert into sqlTablename (columnName1, columnName2) values (valueInfo1, valueInfo2) };</pre>	<p>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 96 for details and potential problems.</p>

Table 114. 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
<ul style="list-style-type: none"> I/O object: recordName I/O option: INQUIRY <p>(with and without Single row select)</p> <p>Clauses that can be modified:</p> <ul style="list-style-type: none"> SELECT INTO WHERE, GROUP BY, HAVING ORDER BY 	<p>Use the get statement.</p> <p>The following is an example with single row select:</p> <pre> get recordName singleRow with #sql{ select Name1, Name2, Age from sqlTable1 sqlLabel1, sqlTable2 sqlLabel2 where Name1 = :NameX order by Age } into nameA, nameB, myage; </pre>	<p>If Single row select is specified in VisualAge Generator, the migration tool includes the EGL singleRow option.</p> <p>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 96 for details and potential problems.</p>
<ul style="list-style-type: none"> I/O object: recordName I/O option: UPDATE <p>Clauses that can be modified:</p> <ul style="list-style-type: none"> SELECT INTO WHERE FOR UPDATE OF 	<p>Use the get forUpdate statement .</p> <p>The following is an example:</p> <pre> 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; </pre>	<p>The migration tool always includes the <i>resultSetID</i> when migrating an UPDATE I/O option for an SQL record. The tool sets the <i>resultSetID</i> to the function name followed by a customer-specified suffix. You can control the suffix with the Stage 2 VAGen Migration Preferences.</p> <p>The migration tool creates the FROM clause based on the table name and table label in the record definition. Special considerations apply. See “SQL I/O statements” on page 96 for details and potential problems.</p>
<ul style="list-style-type: none"> I/O object: recordName I/O option: REPLACE <p>(optional UPDATE/SETUPD functionName)</p> <p>Clause that can be modified:</p> <ul style="list-style-type: none"> SET 	<p>Use the replace statement.</p> <p>The following is an example of the replace statement:</p> <pre> replace recordName with #sql{ update sqlTableName set columnName1 = value1, columnName2 = value2 } from resultSetID; </pre>	<p>If an UPDATE/SETUPD function name is included in VisualAge Generator, the migration tool includes the from <i>resultSetID</i> clause. The migration tool sets the <i>resultSetID</i> to the UPDATE/SETUPD function name followed by a customer-specified suffix. You can control the suffix with the Stage 2 VAGen Migration Preferences.</p> <p>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 96 for details and potential problems.</p>

Table 114. 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
<ul style="list-style-type: none"> I/O object: recordName I/O object: SETINQ <p>(with or without Declare cursor with hold)</p> <p>Clauses that can be modified:</p> <ul style="list-style-type: none"> SELECT INTO WHERE, GROUP BY, HAVING ORDER BY 	<p>Use the open statement. If you are doing a Declare cursor with hold, also use the hold option.</p> <p>The following is an example of an open statement using the hold option:</p> <pre>open resultSetID hold with #sql{ select Name1, Name2 from sqlTable1 sqlLabel1, sqlTable2 sqlLabel2 where Name1 > :Name2 order by Name1 } into Name1, Name2 for recordName;</pre>	<p>The migration tool sets the <i>resultSetID</i> to the function name followed by a customer-specified suffix. You can control the suffix with the Stage 2 VAGen Migration Preferences.</p> <p>If Declare cursor with hold is selected in VisualAge Generator, the migration tool includes the EGL hold option.</p> <p>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 96 for details and potential problems.</p>
<ul style="list-style-type: none"> I/O object: record I/O option: SETUPD <p>(with or without Declare cursor with hold)</p> <p>Clauses that can be modified:</p> <ul style="list-style-type: none"> SELECT INTO WHERE FOR UPDATE OF 	<p>Use the open forUpdate statement. The following is an example using the hold option:</p> <pre>open resultSetID hold forUpdate with #sql{ select Column1, Column2 from sqlTable1 sqlLabel1 where Column1 > :Item1 for update of Column2 } into Item1, Item2 for recordName;</pre>	<p>The migration tool sets the <i>resultSetID</i> to the function name followed by a customer-specified suffix. You can control the suffix with the Stage 2 VAGen Migration Preferences.</p> <p>If Declare cursor with hold is selected in VisualAge Generator, the migration tool includes the EGL hold option.</p> <p>The migration tool creates the FROM clause based on the table name and table label in the record definition. Special considerations apply. See “SQL I/O statements” on page 96 for details and potential problems.</p>
<ul style="list-style-type: none"> I/O object: record I/O option: SQLEXEC <p>with Model SQL Statement</p> <p>Note:</p> <ul style="list-style-type: none"> The SQL record name is optional in this form of SQLEXEC. The following values are valid for the model type: <ul style="list-style-type: none"> None Update Delete 	<p>Use the execute statement. The following is an example of the statement:</p> <pre>execute modelType #sql{ UPDATE mysqltable set Column1 = Column1 * 2 where Column2 = :Column2 } for recordName;</pre> <p>Note: The values for <i>modelType</i> include Update and Delete.</p>	<p>The migration tool does the following things:</p> <ul style="list-style-type: none"> Converts SQLEXEC to the execute statement. Uses the I/O object, if it is specified, as the <i>recordName</i> in the for clause. <p>The migration tool includes the VAGen Model SQL Statement value, if any, as a comment on the EGL execute statement.</p> <p>The migration tool migrates the VAGen SQLEXEC clauses to EGL SQL clauses.</p>

Table 115. Functions - I/O options for SQL statements with Execution time statement build

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<p>Execution time statement build can only be used with the following I/O options:</p> <ul style="list-style-type: none"> • INQUIRY • UPDATE • SETINQ • SETUPD • SQLEXEC <p>Note:</p> <ul style="list-style-type: none"> • You specify Execution time statement build to cause VisualAge Generator to prepare the SQL statement dynamically every time the I/O statement is executed. • You can use Execution time statement build with either unmodified (default) SQL or modified SQL. 	<p>In EGL, you code the EGL 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. The following example shows the EGL equivalent of a VAGen INQUIRY I/O option with Execution time statement build:</p> <pre>prepare prepID from "sqlStatementString" for recordName; get recordName with prepID into itemList;</pre> <p>Note:</p> <ul style="list-style-type: none"> • The <i>sqlStatementString</i> in the prepare statement is a concatenated string of constants and variables that is written in SQL notation. The following example shows a WHERE clause that uses both column names and variables: <pre>[other clauses] + " where columnName = " + itemName + " AND columnName2 = " + itemName2 + [other clauses]</pre> • The examples shown in the rest of this table do not include splitting the variables outside the double quotes. • The SQL statement must be explicitly specified in the prepare statement. • Additional EGL logic is required to obtain the same error handling as in VisualAge Generator. For examples, see the following two rows of this table. 	<p>The migration tool sets the <i>prepID</i> to the function name followed by a customer-specified suffix. You can control the suffix with the Stage 2 VAGen Migration Preferences.</p> <p>The migration tool uses the SQL clauses in the function and the table names and table name variables in the record definition to build the <i>sqlStatementString</i>. The migration tool builds the <i>sqlStatementString</i> in the following way:</p> <ul style="list-style-type: none"> • Does all the processing as though the Execution time statement build is not specified, including: <ul style="list-style-type: none"> – Using the table names 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. – Creating default clauses as necessary based on the record definition. – Converting !itemColumnName variables to their corresponding SQL column name. – Converting VAGen comments (/*) to EGL comments (//) in the prepare statement. <p>Then the migration tool does additional processing to create the <i>sqlStatementString</i>, including:</p> <ul style="list-style-type: none"> • 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. <p>Special considerations apply if the SQL record and its alternate specification record, if any, are not available during migration. For more information and potential problems, see the following sections:</p> <ul style="list-style-type: none"> • "SQL I/O statements" on page 96 • "SQL I/O and missing required SQL clauses" on page 98 • "SQL I/O and !itemColumnName" on page 103

Table 115. Functions - I/O options for SQL statements with Execution time statement build (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
For the INQUIRY, UPDATE, SETINQ, and SETUPD I/O options, the entire I/O statement is treated as a unit for the purposes of error handling. If a soft error occurs during the SQL PREPARE step, processing still continues with the other SQL statements required for the I/O option.	Each I/O statement is treated separately for error handling purposes. To achieve the same error handling as in VisualAge Generator, use the following as an example: <pre> try prepare prepID from "sqlStatementString" for recordName; end if (recordName not HardIOError) // continue on soft error try get recordName with prepID into itemList ; onException errorRoutine; end else // hard error on prepare errorRoutine; end </pre>	<p>If the VAGen function specifies an I/O error routine, the migration tool includes special error handling logic in EGL so that processing continues if a soft error occurs on the EGL prepare statement. For the relationship between the VAGen I/O error routine and the EGL onException statement, see Table 111 on page 311.</p> <p>If the VAGen function does not specify an I/O error routine, the migration tool cannot include the extra logic so it issues a warning message instead.</p> <p>The remaining rows of this table do not show the error handling logic, but only the prepare and get or open statements.</p>
For an SQLEXEC I/O option, the statement is treated as an SQL EXECUTE IMMEDIATE, which does the prepare, execute and destroy as a single SQL command. If a soft error occurs, processing continues or not based on SQL handling for the specific error.	The closest equivalent EGL code is a prepare statement followed by an execute statement. The following is an example: <pre> try prepare prepID from "sqlStatementString" for recordName; execute prepID for recordName ; onException errorRoutine; end </pre>	<p>If the VAGen function uses default SQL and specifies a record and a model option of Update or Delete, the migration tool uses the record definition to create the corresponding default SQL clauses for the prepare statement and execute statements.</p> <p>If the VAGen function uses modified SQL, the migration tool uses that SQL to create the prepare and execute statements.</p> <p>The remaining rows of this table do not show the error handling logic, but only the prepare and execute statements.</p>
<ul style="list-style-type: none"> I/O object: record I/O option: INQUIRY <p>(Single row select is not supported with Execution time statement build.)</p> <p>Clauses that can be modified:</p> <ul style="list-style-type: none"> SELECT INTO WHERE, GROUP BY, HAVING ORDER BY 	Use the prepare statement followed by a get statement. The following is an example: <pre> prepare prepID from " select columnName " + ", columnName2 " + " from table1 t1 " + "[where whereClause]" + "[order by orderByClause]" for recordName; get recordName with prepID into itemList; </pre>	No special considerations.

Table 115. Functions - I/O options for SQL statements with Exection time statement build (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<ul style="list-style-type: none"> I/O object: record I/O option: UPDATE <p>Clauses that can be modified:</p> <ul style="list-style-type: none"> SELECT INTO WHERE FOR UPDATE OF 	<p>Use the prepare statement followed by a get forUpdate statement. The following is an example:</p> <pre> 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; </pre>	<p>The migration tool always includes the <i>resultSetID</i> when migrating an UPDATE I/O option for an SQL record. The tool sets the <i>resultSetID</i> to the function name followed by a customer-specified suffix. You can control the suffix with the Stage 2 VAGEN Migration Preferences.</p>
<ul style="list-style-type: none"> I/O object: record I/O option: SETINQ <p>(with or without Declare cursor with hold)</p> <p>Clauses that can be modified:</p> <ul style="list-style-type: none"> SELECT INTO WHERE, GROUP BY, HAVING ORDER BY 	<p>Use the prepare statement followed by an open statement. The following is an example:</p> <pre> 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; </pre>	<p>The migration tool sets the <i>resultSetID</i> to the function name followed by a customer-specified suffix. You can control the suffix with the Stage 2 VAGEN Migration Preferences.</p> <p>If Declare cursor with hold is selected in VisualAge Generator, the migration tool includes the EGL hold option.</p>
<ul style="list-style-type: none"> I/O object: record I/O option: SETUPD <p>(with or without Declare cursor with hold)</p> <p>Clauses that can be modified:</p> <ul style="list-style-type: none"> SELECT INTO WHERE FOR UPDATE OF 	<p>Use the prepare statement followed by an open forUpdate statement. The following is an example:</p> <pre> 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; </pre>	<p>The tool sets the <i>resultSetID</i> to the function name followed by a customer-specified suffix. You can control the suffix with the Stage 2 VAGEN Migration Preferences.</p> <p>If Declare cursor with hold is selected in VisualAge Generator, the migration tool includes the EGL hold option.</p>
<ul style="list-style-type: none"> I/O object: record I/O option: SQLEXEC <p>with Model SQL Statement</p> <p>Note:</p> <ul style="list-style-type: none"> The SQL record name is optional in this form of SQLEXEC. The following values are valid for the model type: <ul style="list-style-type: none"> None Update Delete 	<p>Use the prepare statement followed by an execute statement. The following is an example:</p> <pre> prepare prepID from " grant " + group_privileges + " on " + table_name + " to " + userid [for recordName] ; execute prepID [for recordName] ; // model = type </pre>	<p>If the VAGEN function uses default SQL and specifies a record and a Model option of Update or Delete, the migration tool uses the record definition to create the corresponding default SQL clauses for the prepare statement and execute statements. The tool includes the VAGEN Model SQL Statement value, if any, as a comment on the EGL execute statement.</p> <p>If the VAGEN function uses modified SQL, the migration tool converts the VAGEN SQLEXEC clauses to EGL SQL clauses.</p>

Table 116. Functions — I/O options for default (unmodified) DL/I statements

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<ul style="list-style-type: none"> I/O object: recordName I/O option: ADD PSB: psbName Database identifier: databaseName pcbNumber 	<p>Use the add statement. The following is an example:</p> <pre>add recordName [usingPCB pcbInfo] ;</pre>	<p>The migration tool includes the usingPCB keyword if the VAGen function specifies the Database identifier. The migration tool sets the <i>pcbInfo</i> value based on the information stored in the VAGen function:</p> <ul style="list-style-type: none"> <i>databaseName</i> followed by a customer-specified suffix. You can control the suffix with the Stage 2 VAGen Migration Preferences. <i>pcbn</i>, where <i>n</i> is the <i>pcbNumber</i> specified in VisualAge Generator.
<ul style="list-style-type: none"> I/O object: recordName I/O option: SCAN PSB: psbName Database identifier: databaseName pcbNumber SCAN options: <ul style="list-style-type: none"> Scan update Scan parent 	<p>Use the get next statement. The following is an example:</p> <pre>get next recordName [usingPCB pcbInfo] ;</pre> <p>The following example shows both Scan update and Scan parent:</p> <pre>get next inParent recordName forUpdate [usingPCB pcbInfo] ;</pre>	<p>The migration tool includes the usingPCB keyword if the VAGen function specifies the Database identifier as described for the ADD I/O option.</p> <p>The migration tool includes the inParent keyword if the VAGen function specifies the Scan parent option.</p> <p>The migration tool includes the forUpdate keyword if the VAGen function specifies the Scan update option.</p>
<ul style="list-style-type: none"> I/O object: recordName I/O option: INQUIRY PSB: psbName Database identifier: databaseName pcbNumber 	<p>Use the get statement. The following is an example:</p> <pre>get recordName [usingPCB pcbInfo] ;</pre>	<p>The migration tool includes the usingPCB keyword if the VAGen function specifies the Database identifier as described for the ADD I/O option.</p>
<ul style="list-style-type: none"> I/O object: recordName I/O option: UPDATE PSB: psbName Database identifier: databaseName pcbNumber 	<p>Use the get forUpdate statement. The following is an example:</p> <pre>get recordName forUpdate [usingPCB pcbInfo] ;</pre>	<p>The migration tool includes the usingPCB keyword if the VAGen function specifies the Database identifier as described for the ADD I/O option.</p>
<ul style="list-style-type: none"> I/O object: recordName I/O option: DELETE PSB: psbName Database identifier: databaseName pcbNumber 	<p>Use the delete statement. The following is an example:</p> <pre>delete recordName [usingPCB pcbInfo] ;</pre>	<p>The migration tool includes the usingPCB keyword if the VAGen function specifies the Database identifier as described for the ADD I/O option.</p>
<ul style="list-style-type: none"> I/O object: recordName I/O option: REPLACE PSB: psbName Database identifier: databaseName pcbNumber 	<p>Use the replace statement. The following is an example:</p> <pre>replace recordName [usingPCB pcbInfo] ;</pre>	<p>The migration tool includes the usingPCB keyword if the VAGen function specifies the Database identifier as described for the ADD I/O option.</p>

Table 117. Functions - Segment Search Arguments for modified DL/I statements

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<p>General information specified for a modified DL/I call with SSAs:</p> <ul style="list-style-type: none"> • I/O option • I/O object • PSB • Database identifier • Scan update • Scan parent • Zero, one or more SSAs that include: <ul style="list-style-type: none"> – Segment Name – Command Codes – Boolean Operator – Segment Field – Relational Operator – Comparison Value Item <p>Note:</p> <ul style="list-style-type: none"> • SSAs are entered in a specialized DL/I Call Editor. • VisualAge Generator formats the SSAs at generation time. 	<p>General format for a modified DL/I call with SSAs is that SSAs are entered between braces in the following way:</p> <pre>with #dli { <i>dliFunction</i> <i>SSAList</i> } ;</pre> <p>For example:</p> <pre><i>ioOptionWithModifiers</i> <i>recordNameList</i> [usingPCB <i>pcbInfo</i>] with #dli { <i>dliFunction</i> <i>segmentName1*cmdCodes</i> (<i>segmentFieldA</i> <i>relationalOperatorA</i> :<i>comparisonValueItemA</i> <i>booleanOperator</i> <i>segmentFieldB</i> <i>relationalOperatorB</i> :<i>comparisonValueItemB</i>) <i>segmentName2*cmdCodes</i> (<i>segmentFieldC</i> <i>relationalOperatorC</i> :<i>comparisonValueItemC</i>) }</pre> <p>Note:</p> <ul style="list-style-type: none"> • SSAs are entered in a text editor. • EGL formats the SSAs at generation time according to the DL/I formatting rules. 	<p>The migration tool builds the <i>recordNameList</i> based on a combination of the I/O object and the command codes specified for the SSAs.</p>
<p>Modified DL/I call with all SSAs deleted.</p> <p>Note: This technique is used in scanning all segments in the database.</p>	<p>General format for a modified DL/I call with all SSAs deleted:</p> <pre><i>ioOptionWithModifiers</i> <i>recordNameList</i> [usingPCB <i>pcbInfo</i>] with #dli{ <i>dliFunction</i> } ;</pre>	<p>The migration tool sets the <i>recordNameList</i> to the name of the I/O object.</p>
<p>VisualAge Generator I/O options:</p> <ul style="list-style-type: none"> • ADD • SCAN • SCAN - Scan update • SCAN - Scan parent • SCAN - Scan update and Scan parent • INQUIRY • UPDATE • DELETE • REPLACE 	<p>Corresponding EGL <i>ioOptionWithModifiers</i>:</p> <ul style="list-style-type: none"> • add • get next • get next forUpdate • get next inParent • get next inParent forUpdate • get • get forUpdate • delete • replace 	<p>No special considerations. The <i>ioOptionWithModifiers</i> are the same values that are used when converting unmodified DL/I I/O options.</p>

Table 117. Functions - Segment Search Arguments for modified DL/I statements (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<p>VisualAge Generator I/O options:</p> <ul style="list-style-type: none"> • ADD • SCAN • SCAN - Scan update • SCAN - Scan parent • SCAN - Scan update and Scan parent • INQUIRY • UPDATE • DELETE • REPLACE 	<p>Corresponding EGL <i>dliFunction</i>:</p> <ul style="list-style-type: none"> • ISRT • GN • GHN • GNP • GHNP • GU • GHU • DLET • REPL <p>Note: The <i>dliFunction</i> must be consistent with the <i>ioOptionWithModifiers</i>.</p>	No special considerations.
<p>VisualAge Generator I/O option, SSAs, and Command Codes</p> <p>Note:</p> <ul style="list-style-type: none"> • Command codes are optional. There is a maximum of 4 command codes for an SSA. • VisualAge Generator does special processing at generation and runtime to support the D and N command codes that are related to path calls. 	<p>EGL I/O option, <i>recordNameList</i>, SSAs and Command Codes</p> <p>Note:</p> <ul style="list-style-type: none"> • The * is only specified if one or more optional command codes are specified. There is a maximum of 4 command codes for an SSA. • The EGL <i>recordNameList</i> varies based on the I/O option and the command codes specified for the SSAs. See the next rows of this table for details. 	The migration tool builds the <i>recordNameList</i> based on a combination of the I/O object and the command codes specified for the SSAs.
<p>VisualAge Generator I/O object and Command Codes without the D or N Command Codes</p> <p>Note: Only the I/O object (target DL/I segment) is retrieved or updated in the database.</p>	The EGL <i>recordNameList</i> is the name of the target DL/I segment.	The migration tool sets the <i>recordNameList</i> to the name of the I/O object.
<p>VisualAge Generator ADD I/O option with the D Command Code</p> <p>Note:</p> <ul style="list-style-type: none"> • The D command code indicates the top segment of the hierarchy that is to be inserted into the database. All segments after the D command code are also inserted. • Only one segment can specify the D command code. • The target segment never specifies the D command code. 	<p>Use the add statement. All segments that are to be inserted must be listed as objects of the I/O statement. The following is an example for specifying a DL/I Insert call:</p> <pre> add recordName1 , recordName2 , recordName3 [usingPCB pcbInfo] with #dli{ ISRT recordName1*D recordName2 recordName3 } ; </pre>	The migration tool creates the <i>recordNameList</i> for the I/O statement to include the DL/I segment record that specifies the D command code and all subsequent DL/I segment records in the hierarchy.

Table 117. Functions - Segment Search Arguments for modified DL/I statements (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<p>VisualAge Generator SCAN, INQUIRY, and UPDATE I/O options with the D Command Code</p> <p>Note:</p> <ul style="list-style-type: none"> The D command code indicates a segment of the hierarchy that is to be retrieved from the database. Each segment other than the target segment that is to be retrieved from the database must specify the D command code. The target segment is always retrieved. Multiple SSAs can specify the D command code. The target segment never specifies the D command code. 	<p>Use the form of the get statement that corresponds to the I/O option. All segments that are to be retrieved from the database must be listed as objects of the I/O statement. The following is an example for specifying a DL/I Get Hold Next in Parent call:</p> <pre> get next inParent recordName1 , recordName3 forUpdate [usingPCB pcbInfo] with #dli{ GHNP recordName1*D recordName2 recordName3 } ; </pre>	<p>The migration tool creates the <i>recordNameList</i> for the I/O statement to include each DL/I segment record that specifies a D command code and the target segment.</p>
<p>VisualAge Generator REPLACE I/O option with the N Command Code</p> <p>Note:</p> <ul style="list-style-type: none"> The N command code indicates that a segment is not to be replaced even though it was retrieved for update with the D command code on a previous SCAN or UPDATE. Multiple segments can specify the N command code. The target segment never specifies the N command code. 	<p>Use the replace statement. Only the target segment is specified in the <i>recordNameList</i>. The following is an example for specifying a DL/I Replace call:</p> <pre> replace recordName3 [usingPCB pcbInfo] with #dli{ REPL recordName1*N recordName3 } ; </pre>	<p>The migration tool sets the <i>recordNameList</i> to the name of the I/O object.</p>
<p>VisualAge Generator Relational Operators for SSAs:</p> <ul style="list-style-type: none"> EQ and = GT and > LT and < GE and >= and => LE and <= and =< NE and ^= and ^=^ 	<p>Corresponding EGL relational operators for SSAs:</p> <ul style="list-style-type: none"> = > < >= <= != <p>Note:</p> <ul style="list-style-type: none"> EGL only supports one variant of the relational operators for SSAs. EGL converts != to a not equal operator that is acceptable to DL/I. 	<p>No special considerations.</p>

Table 117. Functions - Segment Search Arguments for modified DL/I statements (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<p>VisualAge Generator Boolean Operators for SSAs:</p> <ul style="list-style-type: none"> • AND and & • OR and <p>Note: VisualAge Generator does not provide support for the dependent OR.</p>	<p>Corresponding EGL Boolean operators for SSAs:</p> <ul style="list-style-type: none"> • & • (vertical bar) <p>Note:</p> <ul style="list-style-type: none"> • EGL only supports one variant of the Boolean operators for SSAs. • EGL also supports the dependent OR (# symbol). 	<p>No special considerations.</p>
<p>Comparison Value Item</p> <p>Note:</p> <ul style="list-style-type: none"> • The comparison value item can be any item in the program. • If the item is not qualified, VisualAge Generator looks first for the item in the segment record associated with the current SSA. 	<p>Comparison value item</p> <p>Note:</p> <ul style="list-style-type: none"> • The comparison value item can be a literal or any item in the program. • If the item is not qualified, EGL looks first in the target segment, which is the lowest segment in the hierarchy. • The comparison value item must be preceded by a semicolon to indicate a host variable. For example: :qualifier.itemName <p>EGL removes the semicolon during generation.</p>	<p>If a comparison value item is not qualified, the migration tool checks the DL/I segment record associated with the current SSA to determine if the item is in that record. If so, the migration tool includes the DL/I segment record as the qualifier for the comparison value item.</p> <p>Special considerations apply if the DL/I segment record is not available or if the comparison value item is not in the record. For details and potential problems, see “DL/I I/O and comparison value items” on page 105.</p>

Statements

The statements section is organized into the following tables:

- General rules - data item qualification and numeric literals, Table 118 on page 326
- Function invocation, Table 119 on page 327
- Assignment, MOVE, and MOVEA, Table 120 on page 327
- SET, Table 121 on page 329
- RETR and FIND, Table 122 on page 331
- IF, WHILE, and TEST, including EZE Aid, EZESYS, and I/O error values, Table 123 on page 332
- CALL, Table 124 on page 337
- DXFR, Table 125 on page 338
- XFER, Table 126 on page 338

Table 118. Statements - General rules - data item qualification and numeric literals

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<p>Item qualification rules: If an item is not qualified, VisualAge Generator looks for the item in the following order:</p> <ul style="list-style-type: none"> • Items in the function local storage or parameter list. • The I/O object for the function and records in the function 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 primary working storage record for the program, 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. 	<p>Item qualification rules: If a field is not qualified or is partially qualified, EGL looks for the field in the following places, in order:</p> <ul style="list-style-type: none"> • Variables declared at the function level, including item variable names and record variable names in the function local storage and parameter list. • I/O objects, fields within the I/O objects, and fields in any record variables declared in the function local storage or parameter list. • Record variable names and item variable names declared at the program level or specified in the parameter list for the program • Forms listed on the use forms statement for the program. If the use statement only specifies the FormGroup name, then all forms in the FormGroup are considered. • DataTable names specified in the use declarations for the program. • Fields in any record variables, forms, or DataTables declared for the program. If the use form statement only specifies the FormGroup name, then all fields on all forms in the FormGroup are considered. • Fields in a user library specified in the use declarations for the program. • Fields in a system library. • EGL does not permit implicit items. Every item must be explicitly defined. 	<p>See “Level 77 items in records” on page 71 and “Implicit data items in programs” on page 88 for details and potential problems.</p>
<p>Numeric literals:</p> <ul style="list-style-type: none"> • Not enclosed in quotes. • Can use either a period (.) or a comma (,) as the decimal point, depending on the national language. • Floating point literals are not supported. 	<p>Numeric literals:</p> <ul style="list-style-type: none"> • 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. • Floating point literals are supported. 	<p>The migration tool converts the commas used as decimal points to a period except for initial values of form variable fields.</p>

Table 118. Statements - General rules - data item qualification and numeric literals (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
Character, mixed, and DBCS literals: <ul style="list-style-type: none"> • Must be enclosed in quotes. • If a character literal is enclosed in single quotes, the literal is folded to uppercase. • If a character literal is enclosed in double quotes, the literal is used as entered. 	Character, mixed, and DBCS literals: <ul style="list-style-type: none"> • Must be enclosed in double quotes. • The literal is used as entered. • HEX and UNICODE literals are also supported. • Can optionally specify a one-letter prefix to indicate the type of the literal: <ul style="list-style-type: none"> – C - CHAR – D - DBCHAR – M - MBCHAR – X - HEX • Can optionally specify a two-letter prefix to indicate an unprintable character such as a form feed: <ul style="list-style-type: none"> – CX - CHAR – DX - DBCHAR – MX - MBCHAR – UX - UNICODE • If there is no prefix, the literal is treated as a STRING data type. 	The migration tool does not include the one- or two-character prefix for a literal. The migration tool sets the textLiteralDefaultIsString property to NO for all programs and VGUI records.

Table 119. Statements — Function invocation

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
VAGen syntax example: <code>functionName([argumentList]);</code> Note: A function invocation is a single, complete statement.	EGL syntax example: <code>functionName([argumentList]);</code> Note: A function invocation can be used as an argument to another function or call statement or as an operand in other statements such as an if or while statement	See “EZE words” on page 339 for the EGL equivalent system library functions. See Table 111 on page 311 for function invocations from an I/O error routine.
In flow statements: <code>functionName();</code>	Flow statements are not supported. goto <code>functionName;</code>	No special considerations.

Table 120. Statements — Assignment, MOVE, and MOVEA

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
VAGen syntax example: <code>target = functionName ([argumentList]);</code>	EGL syntax example: <code>target = functionName ([argumentList]);</code>	See “EZE words” on page 339 for the EGL equivalent system library functions.
<code>target = numericExpression;</code> OR <code>target = numericExpression (R;</code>	<code>target = numericExpression ;</code> OR <code>target = mathLib.round (numericExpression, -mathlib.decimals(target)) ;</code>	If the (R option is specified, the migration tool converts the option to the EGL mathlib.round() system function.

Table 120. Statements — Assignment, MOVE, and MOVEA (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<p>target = source;</p> <p>OR</p> <p>MOVE source [T0] target;</p> <p>Note:</p> <ul style="list-style-type: none"> • 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. 	<p>target = source ;</p> <p>OR</p> <p>move source to target byName ;</p> <p>OR</p> <p>move source to target withV60Compat;</p> <p>Note:</p> <ul style="list-style-type: none"> • For assignment statements: <ul style="list-style-type: none"> – The target can be a record, 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, item, or certain system variables. – Forms cannot be used in assignment statements. – Move corresponding is never done for an assignment statement. • For move statements: <ul style="list-style-type: none"> – 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 withV60Compat is specified, the move is either an item to item move or a move corresponding depending on the part type of the source. <p>The data conversion and truncation rules are the same as in VisualAge Generator.</p> <p>EGL supports more data conversions than VAGen. For example, you can assign an INT field to a CHAR field</p>	<p>The migration tool considers the following EGL rules when migrating assignment and move statements:</p> <ul style="list-style-type: none"> • 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. • The move withV60Compat statement is tolerated and treated as an item-to-item move or a move corresponding depending on the part type of the source. <p>Therefore, the migration tool does the following things:</p> <ul style="list-style-type: none"> • Converts to an assignment statement in any of the following situations: <ul style="list-style-type: none"> – 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 parameter list or local storage. • Converts to a move byName if the source or target is the I/O object for a function or a record in the function parameter list or local storage. • Converts to a move withV60Compat in all other situations. <p>See “Assignment statements” on page 108 for details and potential problems.</p>

Table 120. Statements — Assignment, MOVE, and MOVEA (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
MOVEA source [T0] target; OR MOVEA source [T0] target FOR occurrence; Note: The <i>source</i> 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 things: <ul style="list-style-type: none"> • 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 121. Statements — SET

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
General information: <ul style="list-style-type: none"> • Commas or blanks can be used to separate multiple options on a single SET statement. 	General information: <ul style="list-style-type: none"> • 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: <i>sqlItem</i> can be an item in an SQL row record or an SQLITEM parameter for a function.	<i>sqlItem</i> = null ; Note: <i>sqlItem</i> can be an item with the isSQLNullable property set to YES in an SQL row record or a nullable parameter for a function.	No special considerations.
SET map [ALARM [CLEAR EMPTY]] ; Note: <ul style="list-style-type: none"> • CLEAR and EMPTY are mutually exclusive. • ALARM and either CLEAR or EMPTY can be combined with the PAGE option. 	set form [alarm [initial empty]] ; Note: <ul style="list-style-type: none"> • 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 121. Statements — SET (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<p>SET <i>map</i> PAGE ;</p> <p>Note: PAGE can be combined with ALARM and with either CLEAR or EMPTY.</p>	<p>converseLib.clearScreen(); // display form</p> <p>OR</p> <p>converseLib.pageEject(); // printer form</p> <p>Note: The replacement for the PAGE option cannot be combined with any other options.</p>	<p>The migration tool migrates SET <i>map</i> PAGE in the following way:</p> <ul style="list-style-type: none"> • If SET <i>map</i> 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 converseLib.clearScreen(); • If the map is a printer map, the tool converts the statement to converseLib.pageEject(); • If the map is not available to determine the map type, the migration tool converts the statement to the invalid function name converseLib.EZE_SETPAGE() as a placeholder. <p>See “SET map PAGE statement” on page 110 for details and potential problems.</p>
<p>SET <i>mapItem</i> [CURSOR FULL [NORMAL DEFINED]] ;</p> <p>Note:</p> <ul style="list-style-type: none"> • <i>mapItem</i> can be a field 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. 	<p>set formField [cursor full [normal initialAttributes]] ;</p> <p>Note:</p> <ul style="list-style-type: none"> • <i>formField</i> 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. 	<p>The migration tool migrates to the EGL equivalent of each option without regard to whether the <i>formField</i> is on a text or print form. See “SET mapItem attributes” on page 111 for details and potential problems.</p>

Table 121. Statements — SET (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<pre>SET mapItem [CURSOR FULL color extendedHighlight MODIFIED [BRIGHT DARK] [PROTECT AUTOSKIP]] ;</pre> <p>Note:</p> <ul style="list-style-type: none"> • <i>mapItem</i> can be a field 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. 	<pre>set formField [cursor full color extendedHighlight modified [bold invisible] [protect skip]] ;</pre> <p>Note:</p> <ul style="list-style-type: none"> • <i>formField</i> 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 <i>extendedHighlight</i> option of underline, EGL does not support setting these attributes for print forms. 	<p>The migration tool migrates to the EGL equivalent of each option without regard to whether the <i>formField</i> is on a text or print form. See “SET mapItem attributes” on page 111 for details and potential problems. See later rows in this table for <i>color</i> and <i>extendedHighlight</i> information.</p>
<pre>color: MONO BLUE GREEN PINK RED TURQ YELLOW WHITE</pre>	<pre>color: defaultColor blue green magenta red cyan yellow white</pre>	No special considerations.
<pre>extendedHighlight: NOHILITE BLINK RVIDEO USCORE</pre>	<pre>extendedHighlight: noHighLight blink reverse underline</pre>	No special considerations.

Table 122. Statements — RETR and FIND

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<pre>RETR DataItem1 table[.searchColumn] DataItem2 [returnColumn] ;</pre> <p>Note:</p> <ul style="list-style-type: none"> • If the <i>searchColumn</i> is not specified, the default is the first column in the table. • If the <i>returnColumn</i> is not specified, the default is the second column in the table. 	<pre>if (DataItem1 in DataTable.searchColumn) DataItem2 = DataTable.returnColumn[sysVar.arrayIndex]; end</pre> <p>Note: The <i>searchColumn</i> and <i>returnColumn</i> are required.</p>	<p>The migration tool converts the RETR statement to an if statement and an assignment statement.</p> <p>Special considerations apply if the table is not available during migration. See “RETR statement” on page 110 for details and potential problems.</p>

Table 122. Statements — RETR and FIND (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<pre> FIND DataItem table[.searchColumn] trueStatement; OR FIND DataItem table[.searchColumn] , falseStatement ; OR FIND DataItem table[.searchColumn] trueStatement [,] falseStatement ; </pre> <p>Note:</p> <ul style="list-style-type: none"> • If the <i>searchColumn</i> is not specified, the default is the first column in the table. • If FIND is used in program flow, the <i>trueStatement</i> and the <i>falseStatement</i> can be the name of a main function or EZECLOS. • If FIND is used in a function, the <i>trueStatement</i> and the <i>falseStatement</i> can be the name of any function, EZECLOS, EZEFL0, or EZERTN. 	<pre> if (DataItem in DataTable.searchColumn) EGLtrueStatement ; end OR if (DataItem in DataTable.searchColumn) else EGLfalseStatement ; end OR if (DataItem in DataTable.searchColumn) EGLtrueStatement ; else EGLfalseStatement ; end </pre> <p>Note: The <i>searchColumn</i> is required.</p>	<p>The migration tool converts the FIND statement to an if statement and the EGL equivalent of the true and false statements. See the next rows in this table for conversion of the <i>trueStatement</i> and <i>falseStatement</i> to the corresponding EGL statements.</p>
<pre> true/falseStatement in flow: • functionName() (main only) • EZECLOS </pre>	<p>Corresponding EGL replacements:</p> <ul style="list-style-type: none"> • goto <i>functionName</i>; • exit program; 	No special considerations.
<pre> true/falseStatement in a function: • functionName (any function) • EZECLOS • EZEFL0 • EZERTN </pre>	<p>Corresponding EGL replacements:</p> <ul style="list-style-type: none"> • <i>functionName</i>(); • exit program; • exit stack; • return; 	No special considerations.

Table 123. Statements — IF, WHILE, and TEST, including EZEAI0, EZESYS, and I/O error values

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<pre> IF logicalExpression ; { statement ; } [ELSE; { statement ; }] END; </pre>	<pre> if (EGLLogicalExpression) { EGLStatement ; } [else { EGLStatement ; }] end </pre>	See later rows in this table for the relationship between the VAGen logical expressions and the EGL logical expressions.
<pre> WHILE logicalExpression ; { statement ; } END; </pre>	<pre> while (EGLLogicalExpression) { EGLStatement ; } end </pre>	See later rows in this table for the relationship between the VAGen logical expressions and the EGL logical expressions.

Table 123. Statements — IF, WHILE, and TEST, including EZEID, EZESYS, and I/O error values (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<p>TEST testCondition trueStatement ;</p> <p>TEST testCondition , falseStatement ;</p> <p>TEST testCondition trueStatement [,] falseStatement ;</p> <p>Note:</p> <ul style="list-style-type: none"> The TEST statement is similar to an IF ... IS statement. The exception to this is TEST <i>mapItem</i> nnn, +nnn, and -nnn, which does not have an IF statement equivalent. If TEST is used in program flow, the <i>trueStatement</i> and the <i>falseStatement</i> can be the name of a main function or EZECLAS. If TEST is used in a function, the <i>trueStatement</i> and the <i>falseStatement</i> can be the name of any function, EZECLAS, EZEFLD, or EZERTN. 	<p>if (EGLLogicalExpression) EGLtrueStatement ; end</p> <p>if (EGLLogicalExpression) else EGLfalseStatement ; end</p> <p>if (EGLLogicalExpression) EGLtrueStatement ; else EGLfalseStatement ; end</p>	<p>With the following exceptions, the migration tool converts the TEST statement to the equivalent if ... is statement and the EGL equivalent of the true and false statements. The exceptions are:</p> <ul style="list-style-type: none"> TEST <i>mapItem</i> nnn, +nnn, and -nnn. TEST <i>sqlItem</i> NULL <p>See later rows in this table for the relationship between the VAGen logical expressions and the EGL logical expressions.</p> <p>See later rows in this table for the conversion of TEST <i>mapItem</i> nnn, +nnn, and -nnn.</p> <p>See later rows in this table for the conversion of TEST <i>sqlItem</i> NULL.</p> <p>See later rows in this table for conversion of the <i>trueStatement</i> and <i>falseStatement</i> to the corresponding EGL statements.</p>
<p>VisualAge Generator boolean operators for IF and WHILE:</p> <ul style="list-style-type: none"> AND OR 	<p>Corresponding EGL boolean operators for if and while:</p> <ul style="list-style-type: none"> && or and or or 	<p>If you clear the Use VAGen logical operators (AND and OR) migration preference, the migration tool uses && and as the logical operators. If you select the preference, the migration tool uses and and or as the logical operators.</p>
<p>VisualAge Generator relational operators for IF and WHILE:</p> <ul style="list-style-type: none"> EQ and = NE and ^= LE and <= and =< LT and < GE and >= and => GT and > 	<p>Corresponding EGL relational operators for if and while:</p> <ul style="list-style-type: none"> == != <= < >= > 	<p>No special considerations.</p>

Table 123. Statements — IF, WHILE, and TEST, including EZE Aid, EZESYS, and I/O error values (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<p>VisualAge Generator state operators for IF and WHILE:</p> <ul style="list-style-type: none"> • IS • NOT 	<p>Corresponding EGL state operators for if and while:</p> <ul style="list-style-type: none"> • is • not 	<p>In most cases, the migration tool does the following things:</p> <ul style="list-style-type: none"> • Migrates IS and NOT as shown. • Migrates a VAGen TEST statement to an EGL if ... is statement. <p>The exceptions are:</p> <ul style="list-style-type: none"> • TEST <i>mapItem</i> nnn, +nnn, and -nnn • TEST <i>mapItem</i> NULL <p>See later rows in this table for details of the exceptions.</p>
<p>VisualAge Generator array operator for IF and WHILE:</p> <ul style="list-style-type: none"> • IN <p>To search starting from a specific element of an array, use: X IN Y[Z] where Z is the index in the array from which to start searching. If no subscript is specified for the array, the search starts from the beginning of the array.</p>	<p>Corresponding EGL state operators for if and while:</p> <ul style="list-style-type: none"> • in <p>To search starting from a specific element of an array, use: X in Y from Z where Z is the index in the array from which to start searching. If no subscript is specified for the array, the search starts from the beginning of the array.</p>	<p>Special considerations apply. See “Checking for IN literal or scalar” on page 112.</p>
<p>VisualAge Generator mapItem state conditions:</p> <ul style="list-style-type: none"> • BLANK or BLANKS • CURSOR • DATA • MODIFIED • NULL or NULLS • NUMERIC 	<p>Corresponding EGL form field state conditions:</p> <ul style="list-style-type: none"> • blanks • cursor • data • modified • blanks • numeric 	<p>The migration tool converts to the equivalent EGL state conditions.</p> <p>Special considerations apply to <i>mapItem</i> NULL. See “Checking SQL and map items for NULL” on page 113 for details and potential problems.</p>
<p>Special mapItem state condition for the TEST statement: nnn +nnn -nnn</p> <p>Note: This compares the length of the data the user entered to nnn. The test is =, >, or < corresponding to nnn, +nnn, or -nnn.</p>	<p>EGL does not provide direct support for this state condition. However, you can achieve the equivalent result by following these steps:</p> <ol style="list-style-type: none"> 1. Use the system library function converseLib.fieldInputLength(), which returns the length of the data entered by the user. 2. Use an if statement to compare the resulting length for == , >, or < corresponding to nnn, +nnn, or -nnn, respectively. 	<p>When migrating any program, the migration tool always includes a declaration for:</p> <pre><custPrefix>EZE_ITEMLEN</pre> <p>The migration tool does the following things for TEST nnn, +nnn, or -nnn:</p> <ul style="list-style-type: none"> • Adds an extra statement just before the TEST statement to set <code><custPrefix>EZE_ITEMLEN</code> using the system library function converseLib.fieldInputLength(). • Changes the TEST statement to an if statement and compares <code><custPrefix>EZE_ITEMLEN</code> to == nnn, > nnn, or < nnn.

Table 123. Statements — IF, WHILE, and TEST, including EZEID, EZESYS, and I/O error values (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
VisualAge Generator map state conditions: <ul style="list-style-type: none"> MODIFIED 	Corresponding EGL form state conditions: <ul style="list-style-type: none"> modified 	No special considerations.
VisualAge Generator EZEID state conditions: <ul style="list-style-type: none"> ENTER BYPASS PAn, where <i>n</i> = 1, 2, 3 PF<i>n</i>, where <i>n</i> is 1 to 24 PA PF 	Corresponding EGL converseVar.eventKey state conditions: <ul style="list-style-type: none"> enter bypass pan, where <i>n</i> = 1, 2, 3 pf<i>n</i>, where <i>n</i> is 1 to 24 pakey pfkey 	No special considerations.
VisualAge Generator sqlItem state conditions: <ul style="list-style-type: none"> BLANK or BLANKS NULL NUMERIC TRUNC <p>Note: Checking an sqlItem for NULLS uses the IS or NOT operator, the same as for checking for any other state.</p>	Corresponding EGL SQL item state conditions: <ul style="list-style-type: none"> blanks null numeric trunc <p>Note: Checking an SQL item for null is done by using: <code>sqlItem == null</code> OR <code>sqlItem != null</code> rather than using the is or not operator.</p>	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 113 for details and potential problems.
VisualAge Generator record state conditions: <ul style="list-style-type: none"> DED DUP EOF ERR FMT FNA FNF FUL HRD LOK NRF UNQ <p>Note:</p> <ul style="list-style-type: none"> 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: <ul style="list-style-type: none"> deadLock duplicate or unique endOfFile ioError invalidFormat fileNotAvailable fileNotFound full hardIOError deadLock noRecordFound unique <p>Note:</p> <ul style="list-style-type: none"> 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 114 for details and potential problems. Special considerations also apply to migrating LOK. See “I/O error value LOK” on page 116 for details and potential problems.

Table 123. Statements — IF, WHILE, and TEST, including ZEZAID, ZEZSYS, and I/O error values (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
VisualAge Generator UI record state conditions: • MODIFIED	Corresponding EGL VGUI record state conditions: • modified	No special considerations.
VisualAge Generator DataItem state conditions: • BLANK or BLANKS • NUMERIC	Corresponding EGL data item state conditions: • blanks • numeric	No special considerations.
VisualAge Generator ZEZSYS state conditions: • AIX • AIXCICS • HP • IMSBMP • IMSVS • MVS BATCH • MVSCICS • NTCICS • OS2 • OS2CICS • OS2GUI • OS400 • SCO • SOLACICS • SOLARIS • TSO • VMCMS • VMBATCH • VSEBATCH • VSECICS • WINGUI • WINNT • ITF	Corresponding sysVar.systemType state conditions: • aix • AIXCICS • hpux • 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 ZEZSYS. See “ZEZSYS” on page 118 for details and potential problems. Note: Not all of the VAGen runtime environments are supported. However, the migration tool always converts to an equivalent value, even if it is not valid in EGL. The migration tool preserves the unsupported values to facilitate finding places where you might need to update your logic if you change from an unsupported environment such as TSO to a supported EGL environment such as ZOSCICS.
true/falseStatement in flow: • functionName() (main only) • ZECLOS	Corresponding EGL replacements: • goto functionName ; • exit program;	No special considerations.
true/falseStatement in a function: • functionName (any function) • ZECLOS • ZEFLO • ZERTN	Corresponding EGL replacements: • functionName(); • exit program; • exit stack; • return;	No special considerations.

Table 124. Statements — CALL

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<p>CALL <i>programName</i> argument [{ [,] argument }] [(options)] ;</p> <p>OR</p> <p>CALL <i>serviceRoutine</i> argument [{ [,] argument }] [(options)] ;</p> <p>Note:</p> <ul style="list-style-type: none"> Commas to separate the arguments are optional. The argument list is not enclosed in parentheses. The <i>programName</i> or <i>serviceRoutine</i> is never enclosed in quotes. 	<p>call <i>programName</i> (<i>argument</i> [{ , <i>argument</i> }]) [{ <i>options</i> }] ;</p> <p>Note:</p> <ul style="list-style-type: none"> Commas to separate the arguments are required. The argument list must be enclosed in parentheses. The <i>programName</i> cannot be a reserved word. If the program is a non-EGL program, use a linkage options element to specify the real name. The <i>programName</i> must be enclosed in quotes if the program is not in the workspace. 	<p>See later rows in this table for conversion of the options to the corresponding EGL statements or options.</p> <p>If you select the Enclose CALL and DXFR program names in quotes migration preference, the migration tool encloses the <i>programName</i> in quotes. If you clear the preference, the migration tool does not enclose the <i>programName</i> in quotes.</p> <p>Regardless of the preference setting, the migration tool does the following things:</p> <ul style="list-style-type: none"> If the (NONCSP option is specified, encloses <i>programName</i> in quotes. If the <i>programName</i> is EZCHART, encloses the <i>programName</i> in quotes and sets the isExternal property to yes. <p>See “Service routines” on page 349 for information on migrating the CALL statement for them.</p>
REPLY option	<p>If the REPLY option is specified in VisualAge Generator, the migration tool converts the option to the following EGL statements:</p> <p>try call <i>programName</i> (<i>argument</i> [{ , <i>argument</i> }]) [{ <i>otherOptions</i> }] ; end</p>	The migration tool includes the try block if the REPLY option is specified.
<p>otherOptions:</p> <ul style="list-style-type: none"> NOMAPS NONCSP 	<p>Corresponding EGL otherOptions:</p> <ul style="list-style-type: none"> isNoRefresh = YES isExternal = YES 	No special considerations.

Table 125. Statements — DXFR

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<p>DXFR target</p> <pre>[recordName] [(NONCSP)] ;</pre> <p>where target is</p> <ul style="list-style-type: none"> • programName • EZEAPP <p>Note:</p> <ul style="list-style-type: none"> • Any record can be passed. • If a working storage record is passed, any level 77 items are not included. • The <i>programName</i> is never enclosed in quotes. 	<p>transfer to program <i>target</i></p> <pre>[passing recordName] [isExternal = YES] ;</pre> <p>where <i>target</i> is:</p> <ul style="list-style-type: none"> • <i>programName</i> • sysVar.transferName <p>Note:</p> <ul style="list-style-type: none"> • Any record can be passed. • The <i>programName</i> cannot be a reserved word. If the program is a non-EGL program, use a linkage options element to specify the real name. • The <i>programName</i> must be enclosed in quotes if the program is <i>not</i> in the workspace. 	<p>If you select the migration preference Enclose CALL and DXFR program names in quotes, the migration tool encloses the <i>programName</i> in quotes. If you clear the preference, the migration tool does not enclose the <i>programName</i> in quotes.</p> <p>Regardless of the preference setting, the migration tool does the following things:</p> <ul style="list-style-type: none"> • Encloses <i>programName</i> in quotes if the (NONCSP) option is specified. • Never encloses sysVar.transferName in quotes.

Table 126. Statements — XFER

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<p>Variation 1 - Migrate to Transfer (no map or UI record)</p> <p>XFER target</p> <pre>[recordName] [(NONCSP)] ;</pre> <p>where target is</p> <ul style="list-style-type: none"> • transactionName • EZEAPP <p>Note:</p> <ul style="list-style-type: none"> • 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. 	<p>EGL syntax for transfer statement:</p> <p>transfer to transaction <i>target</i></p> <pre>[passing recordName] [isExternal = YES] ;</pre> <p>where <i>target</i> is</p> <ul style="list-style-type: none"> • <i>transactionName</i> • sysVar.transferName <p>Note: Any record can be passed.</p>	<p>If there is no comma in the statement, the migration tool converts the XFER to an EGL transfer to transaction statement.</p> <p>For more information on <i>transactionName</i>, see a later row in this table.</p>

Table 126. Statements — XFER (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<p>Variation 2 - Migrate to Show (XFER with a map or XFER with a UI record)</p> <pre>XFER target [recordName] , mapName [(NONCSP) ; OR XFER target [recordName] , UIRecordName ;</pre> <p>where <i>target</i> is</p> <ul style="list-style-type: none"> • transactionName • EZEAPP • ' ' for XFER with a UI record <p>Note:</p> <ul style="list-style-type: none"> • Any record can be passed. If a working storage record is passed, any level 77 items are not included. • (NONCSP is only supported for XFER with a map. 	<p>EGL syntax for show statement varies depending on whether a form or a VGUI record is used.</p> <pre>show formName returning to target [passing recordName] [isExternal = YES] ; OR show UIRecordName [returning to target] [passing recordName] ;</pre> <p>where <i>target</i> is</p> <ul style="list-style-type: none"> • transactionName • sysVar.transferName <p>Note:</p> <ul style="list-style-type: none"> • Any record can be passed. • For show with a VGUI record, if the target is ' ', the returning to target clause is omitted. 	<p>If there is a comma in the statement, the migration tool converts the XFER statement to an EGL show statement.</p> <p>For more information on <i>transactionName</i>, see a later row in this table.</p>
<p><i>transactionName</i></p> <p>Note:</p> <ul style="list-style-type: none"> • <i>transactionName</i> is the program name in nontransactional environments. • The <i>transactionName</i> is never enclosed in quotes. 	<p><i>transactionName</i></p> <p>Note:</p> <ul style="list-style-type: none"> • <i>transactionName</i> is the program name in nontransactional environments. The <i>transactionName</i> cannot be a reserved word. • If the program is a non-EGL program, use a linkage options element to specify the real name. • The <i>transactionName</i> must be enclosed in quotes if the program is not in the workspace. 	<p>If you select the Enclose XFER program names in quotes migration preference, the migration tool encloses the <i>transactionName</i> in quotes. If you clear the preference, the migration tool does not enclose the <i>transactionName</i> in quotes.</p> <p>Regardless of the preference setting, the migration tool does the following things:</p> <ul style="list-style-type: none"> • Encloses <i>transactionName</i> in quotes if the (NONCSP option is specified • Never encloses sysVar.transferName in quotes.

EZE words

The EZE words section is organized into the following sections:

- “Program flow EZE words” on page 340
- “SQL EZE words” on page 341
- “DL/I EZE words” on page 341
- “Date and time EZE words” on page 342
- “Other data EZE words” on page 343
- “General function EZE words” on page 345
- “String EZE words” on page 346
- “Math EZE words” on page 347

- “User interface EZE words” on page 348
- “Object scripting EZE words” on page 349

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 127. Program flow EZE words

EZE word in VisualAge Generator 4.5	EGL
EZECLOS	<p>This depends on the location:</p> <ul style="list-style-type: none"> • If used in an I/O error routine, the migration tool converts EZECLOS to the following EGL code, within the try 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 EGL code: exit program; <p>Note: The exit program statement has a default return code of sysVar.returnValue, which is the equivalent of EZERCODE. This default provides the same capability as VisualAge Generator.</p>
EZEFLO Note: EZEFLO cannot be used in flow statements.	<p>This depends on the location:</p> <ul style="list-style-type: none"> • If used in an I/O error routine, the migration tool converts EZEFLO to the following EGL code, within the try 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 EGL code: exit stack;
EZERTN or EZERTN(<i>return value</i>) Note: <ul style="list-style-type: none"> • EZERTN cannot be used in flow statements. • EZERTN(<i>return value</i>) cannot be used as an I/O error routine. 	<p>This depends on the location:</p> <ul style="list-style-type: none"> • If used in an I/O error routine, the migration tool includes the try block but omits the onException statement. • Used anywhere else, the migration tool converts EZERTN to the following EGL code: return; OR return(returnValue); <p>Note: If the <i>returnValue</i> is EZESYS, see the EZESYS information in “Other data EZE words” on page 343 for additional considerations.</p>

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 128. SQL EZE words

EZE word in VisualAge Generator 4.5	EGL definition
EZECONCT	vgLib.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. For more information, see “Differences in SQL support” on page 233.
EZESQCOD	sysVar.sqlData.sqlcode
EZESQISL Note: <ul style="list-style-type: none"> 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 compatibility. 	vgVar.sqlIsolationLevel Note: EGL supports vgVar.sqlIsolationLevel for the following uses: <ul style="list-style-type: none"> vgLib.connectionService() regardless of whether VAGen compatibility mode is selected sysLib.connect() only when VAGen compatibility mode is selected
EZESQLCA	sysVar.sqlData.sqlca Note: sysVar.sqlData.sqlca is only partially supported in EGL. For debug and Java generation, EGL does not set the fields within sysVar.sqlData.sqlca that contain the values for sysVar.sqlData.sqlerrmc and sysVar.sqlData.sqlwarn[7] .
EZESQRD3	sysVar.sqlData.sqlerrd[3] Note: The migration tool changes this to an array reference.
EZESQRRM	sysVar.sqlData.sqlerrmc Note: sysVar.sqlData.sqlerrmc is not supported for debug or Java generation.
EZESQWN1	sysVar.sqlData.sqlwarn[2] Note: The migration tool changes this to an array reference.
EZESQWN6	sysVar.sqlData.sqlwarn[7] Note: The migration tool changes this to an array reference. sysVar.sqlData.sqlwarn[7] is not supported for debug or Java generation.
N/A	sysVar.sqlData.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.

DL/I 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 129. DL/I EZE words

EZE word in VisualAge Generator 4.5	EGL
EZEDLCER	dliVar.cicsError
EZEDLCON	dliVar.cicsCondition
EZEDLDBD	dliVar.dbName
EZEDLERR	dliVar.handleHardDLIErrors
EZEDLKEY	dliVar.keyArea[1:dliVar.keyAreaLen]
EZEDLKYL	dliVar.keyAreaLen
EZEDLLEV	dliVar.segmentLevel
EZEDLPCB Note: This is an array; the default subscript is 1.	<p>The migration tool always sets the variable that declares the PSBRecord for the program to <i>psb</i>. Therefore, in statements, the migration tool converts EZEDLPCB[<i>n</i>] in the following way:</p> <ul style="list-style-type: none"> • EZEDLPCB[0] converts to <i>psb.iopcb</i>. • EZEDLPCB converts to <i>psb.pcb1</i>, because 1 is the default subscript. • EZEDLPCB[<i>n</i>], where <i>n</i> is a numeric literal converts to <i>psb.pcbn</i>. <p>In the called parameter list for a program, special considerations apply. For details, see Table 105 on page 301.</p>
EZEDLPRO	dliVar.procOptions
EZEDLPSB	<p>In statements except the CALL statement, EZEDLPSB converts to: dliLib.psbData.psbName</p> <p>In the CALL statement, EZEDLPSB converts to: dliLib.psbData</p> <p>In the called parameter list for a program, special considerations apply. For details, see Table 105 on page 301.</p>
EZEDLRST	dliVar.cicsRestart
EZEDLSEG	dliVar.segmentName
EZEDLSSG	dliVar.numSensitiveSegs
EZEDLSTC	dliVar.statusCode
EZEDLTRM Note: EZEDLTRM is equivalent to EZECNVCM. The migration tool converts both EZE words to converseVar.commitOnConverse .	converseVar.commitOnConverse

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 130. Date and time EZE words

EZE word in VisualAge Generator 4.5	EGL definition
EZEDAY	vgVar.currentShortJulianDate
EZEDAYL	vgVar.currentJulianDate
EZEDAYLC	vgVar.currentFormattedJulianDate
EZEDTE	vgVar.currentShortGregorianDate
EZEDTEL	vgVar.currentGregorianDate
EZEDTELC	vgVar.currentFormattedGregorianDate
EZETIM	vgVar.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 131. Other data EZE words

EZE word in VisualAge Generator 4.5	EGL definition
EZE Aid	converseVar.eventKey
EZEAPP	sysVar.transferName
EZECNVCM	converseVar.commitOnConverse
EZECONVT	sysVar.callConversionTable
<i>record</i> .EZEDEST	record.resourceAssociation Note: The qualification is still the record name.
EZEDESTP	converseVar.printerAssociation
EZEFEC	vgVar.handleHardIOErrors
EZELOC	sysVar.remoteSystemID
EZELTERM	sysVar.terminalID for a text program. sysVar.conversationID for a VGWebTransaction program. Note: <ul style="list-style-type: none"> In a text program, both sysVar.terminalID and sysVar.conversationID provide the terminalID information. In a VGWebTransaction program, both sysVar.terminalID and sysVar.conversationID provide the conversationID information.

Table 131. Other data EZE words (continued)

EZE word in VisualAge Generator 4.5	EGL definition
EZEMNO	<ul style="list-style-type: none"> If EZEMNO is used as the target of a MOVE or assignment, the migration tool does the following things: <ul style="list-style-type: none"> If EZEMNO is set from a numeric literal other than 9999, EZEMNO becomes: converseLib.validationFailed(numericLiteral); If EZEMNO is set from numeric literal 9999, EZEMNO becomes: converseLib.validationFailed(); If EZEMNO is set from an item, EZEMNO becomes: if (itemName == 9999) converseLib.validationFailed(); else converseLib.validationFailed(itemName); end If EZEMNO is used anywhere else, it is replaced with: converseVar.validationMsgNum
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.	<i>custPrefixEZEMSG</i> Note: <ul style="list-style-type: none"> <i>custPrefix</i> is the RenamingPrefix you specified during Stage 2 migration. There is no dot between <i>custPrefix</i> and EZEMSG Where EZEMSG is used in functions, the migration tool keeps the same qualifications for <i>custPrefixEZEMSG</i> that were used by EZEMSG in those functions. For example, xxxx.EZEMSG becomes xxxx.<i>custPrefixEZEMSG</i> Where EZEMSG is used in maps, the migration tool does the following things: <ul style="list-style-type: none"> Changes the field name to <i>custPrefixEZEMSG</i> Sets the msgField property of the form to <i>custPrefixEZEMSG</i>
EZEOVER	vgVar.handleOverflow
EZEOVERS	sysVar.overflowIndicator
EZERCODE Note: VisualAge Generator tolerates negative values and values greater than 512 for EZERCODE.	sysVar.returnValue Note: EGL does not permit negative values or values greater than 512 for sysVar.returnValue .
EZEREPLY	vgVar.handleSysLibraryErrors
EZERT2 Note: In VisualAge Generator 4.5, EZERT2 is used only as the condition code for MQ Series access.	vgVar.mqConditionCode

Table 131. Other data EZE words (continued)

EZE word in VisualAge Generator 4.5	EGL definition
EZERT8 Note: EZERT8 is set in the following situations: <ul style="list-style-type: none"> • 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 in the following situations: <ul style="list-style-type: none"> • All call statements. • All sysLib system function invocations. • Some strLib, mathLib, and vgLib 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	converseVar.segmentedMode
EZESEGTR	sysVar.transactionID
EZESYS	<p>To use the EGL values in an if or while statement, use: sysVar.systemType</p> <p>To get the old VAGen values for use in any other statement, use: <i>myItem</i> = vgLib.getVAGSysType();</p> <p>and then use <i>myItem</i> in the statement.</p> <p>If you need to use the old VAGen value in a migrated VAGen program, use: <i>custPrefixEZESYS</i></p> <p>where <i>custPrefix</i> is the Renaming Prefix you specified during Stage 2 of migration. Based on the Do not initialize old EZESYS values migration preference, the migration tool includes or omits a data declaration for <i>custPrefixEZESYS</i> and a statement to initialize it to the old VAGen value.</p> <p>See “EZESYS” on page 118 for details and potential problems.</p>
EZETST Note: Set for IF...IN, and MOVEA. EZETST is 2-byte binary.	sysVar.arrayIndex Note: arrayIndex is an 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 132. General function EZE words

EZE word in VisualAge Generator 4.5	EGL definition
result = EZEBYTES(<i>itemOrRecord</i>) 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 (<i>itemOrRecordOrForm</i>) Note: EGL supports a form 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() Note: Debug and Java generation use JDBC for SQL support. JDBC only supports single-phase commit. For more information see “Differences in SQL support” on page 233.
EZECONV(<i>target</i> , <i>direction</i> , <i>conversionTable</i>) Note: The direction must be specified as a literal; either 'L' or 'R'	sysLib.convert() Note: The direction must be specified as a value from the ConvertDirection enumeration: either ConvertDirection.local or ConvertDirection.remote
EZEC10(<i>xxx</i> , <i>yyy</i> , <i>zzz</i>)	sysLib.verifyChkDigitMod10()
EZEC11(<i>xxx</i> , <i>yyy</i> , <i>zzz</i>)	sysLib.verifyChkDigitMod11()
EZEG10(<i>xxx</i> , <i>yyy</i> , <i>zzz</i>)	sysLib.calculateChkDigitMod10()
EZEG11(<i>xxx</i> , <i>yyy</i> , <i>zzz</i>)	sysLib.calculateChkDigitMod11()
EZEPURGE(<i>queueName</i>)	sysLib.purge()
EZEROLLB()	sysLib.rollback()
EZEWAIT(<i>variableName</i>) Note: <i>variableName</i> provides the time in hundredths of a second.	sysLib.wait (<i>variableName</i>); Note: <ul style="list-style-type: none"> <i>variableName</i> provides the time in seconds. The migration tool converts the time to seconds. See “EZEWAIT” on page 120 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 133. String EZE words

EZE word in VisualAge Generator 4.5	EGL definition
EZESBLKT	strLib.setBlankTerminator()
EZESCCWS	vgLib.concatenateWithSeparator()
EZESCMR	vgLib.compareBytes()
EZESCNCT	vgLib.concatenateBytes()

Table 133. String EZE words (continued)

EZE word in VisualAge Generator 4.5	EGL definition
EZESCOPY	<code>vgLib.copyBytes()</code>
EZESFIND	<code>vgLib.findStr()</code>
EZESNULLT	<code>strLib.setNullTerminator()</code>
EZESSET	<code>vgLib.setSubStr()</code>
EZESTLEN	<code>strLib.byteLen()</code>
EZESTOKN	<code>strLib.getNextToken()</code>

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. In VisualAge Generator, the EZE math functions are written as a complete statement in the form:

```
result = EZEMathFunction(argument1, argument2, ... argumentN);
```

In EGL, many of the math functions can be used in a portion of a statement (for example, as the operand in an **if** or **while** statement). Therefore, the result must be assigned using the **mathLib.assign()** function in the following way:

```
mathLib.assign(EGLMathFunction(argument1, argument2, ... argumentN), result);
```

The following tables indicate which of the EGL math function must use the **mathLib.assign()** function to obtain the result. Otherwise, except where noted, the argument lists for the corresponding math functions are the same in VisualAge Generator as they are in EGL, so they are omitted from the table.

Table 134. Math EZE words — General math functions

EZE word in VisualAge Generator 4.5	EGL definition
EZEABS	<code>mathLib.assign(mathLib.abs(...), result)</code>
EZECEIL	<code>mathLib.ceiling()</code>
EZEEXP	<code>mathLib.assign(mathLib.exp(...), result)</code>
EZEFLOOR	<code>mathLib.floor()</code>
EZEFREXP	<code>mathLib.assign(mathLib.frexp(...), result)</code>
EZELDEXP	<code>mathLib.assign(mathLib.ldexp(...), result)</code>
EZELOG	<code>mathLib.assign(mathLib.log(...), result)</code>
EZELOG10	<code>mathLib.assign(mathLib.log10(...), result)</code>
EZEMAX	<code>mathLib.assign (mathLib.max (...), result)</code>
EZEMIN	<code>mathLib.assign (mathLib.min(...), result)</code>
EZEMODF	<code>mathLib.assign (mathLib.modf(...), result)</code>
EZENCMPR	<code>vgLib.compareNum()</code>
EZEPOW	<code>mathLib.assign (mathLib.pow(...), result)</code>
EZEPRSCN	<code>mathLib.precision()</code>

Table 134. Math EZE words — General math functions (continued)

EZE word in VisualAge Generator 4.5	EGL definition
EZEROUND	mathLib.round() Note: mathLib.round() is also used to replace VAGen assignment statements that use the (R option). The migration tool converts this type of assignment statement to the following syntax: <pre>result = mathLib.round(numericExpression, -mathLib.decimals(result));</pre>
EZESQRT	mathLib.assign(mathLib.sqrt(...), result)

Table 135. Math EZE words — Trigonometric math functions

EZE word in VisualAge Generator 4.5	EGL definition
EZEACOS	mathLib.assign(mathLib.acos(...), result)
EZEASIN	mathLib.assign(mathLib.asin(...), result)
EZEATAN	mathLib.assign(mathLib.atan(...), result)
EZEATAN2	mathLib.assign(mathLib.atan2(...), result)
EZECOS	mathLib.assign(mathLib.cos(...), result)
EZECOSH	mathLib.assign(mathLib.cosh(...), result)
EZESIN	mathLib.assign(mathLib.sin(...), result)
EZESINH	mathLib.assign(mathLib.sinh(...), result)
EZETAN	mathLib.assign(mathLib.tan(...), result)
EZETANH	mathLib.assign(mathLib.tanh(...), result)

Table 136. Math EZE words — Floating point math functions

EZE word in VisualAge Generator 4.5	EGL definition
EZEFLADD	mathLib.assign(vgLib.floatingSum(...), result)
EZEFLDIV	mathLib.assign(vgLib.floatingQuotient(...), result)
EZEFLMOD	mathLib.assign(vgLib.floatingMod(...), result)
EZEFLMUL	mathLib.assign(vgLib.floatingProduct(...), result)
EZEFLSET	mathLib.assign(..., result)
EZEFLSUB	mathLib.assign(vgLib.floatingDifference(...), result)

User interface 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 137. User interface EZE words

EZE word in VisualAge Generator 4.5	EGL definition
EZEUIERR	sysLib.setError
EZEUILOC	sysLib.setLocale

Object scripting EZE words

Table 138. Object scripting EZE words

EZE word in VisualAge Generator 4.5	EGL definition
EZESCRPT(<i>targetScriptName</i>)	EZE_SCRIPT(<i>targetScriptName</i>) There is no corresponding EGL function that replaces EZESCRPT. The migration tool creates intentionally invalid syntax and issues an error message.

Service routines

The service routines section is organized into the following tables:

- Service Routines - general syntax, Table 139 on page 349
- Service Routines - VisualAge Generator and EGL equivalent routines, Table 140 on page 349

Table 139. Service Routines - general syntax

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
CALL <i>serviceRoutine</i> [<i>argumentList</i>] ;	<i>eglSystemLibrary.EGLSystemFunction</i> ([<i>argumentList</i>]); Note: EGL system functions use the same argument list as in VisualAge Generator.	No special considerations.
CALL <i>serviceRoutine</i> [<i>argumentList</i>] (REPLY ;	try <i>eglSystemLibrary.EGLSystemFunction</i> ([<i>argumentList</i>]); 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 block.

Table 140. 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	vgLib.startTransaction()	No special considerations.
CALL CSPTDLI	vgLib.VGTDLI() Note: EGL also supports EGLTDLI and AIBTDLI.	No special considerations.
CALL EZCHART	call "EZCHART" [<i>arguments</i>] { isExternal = YES ; 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 block.
CALL RESET	sysLib.rollback()	No special considerations.

PSBs

In VisualAge Generator, the PSB is a part type. The PSB contains a subset of the information in the IMS or DL/I PSB. There is no name associated with a TP PCB. The database name associated with DB and GSAM PCBs does not have to be unique. A DL/I I/O function can refer to a specific PCB within the PSB either by the database name or by the PCB number. In statements and in the called parameter list for a program, the EZEDLPCB special function word enables you to refer to a PCB by number. The I/O PCB is not explicitly included in the VAGen PSB, but is always present for the IMSVS, IMSBMP, and MVS Batch target environments. The I/O PCB is considered to be PCB number 0.

In EGL, the PSB is a subtype of the record part type. The PSBRecord is a non-structured record. The name of each PCB variable within the PSBRecord must be unique. A DL/I I/O function can refer to a specific PCB by using the name given to the PCB variable within the PSBRecord. Similarly, in statements and in the parameter list for a program, you use the name given to the PCB variable within the PSB.

The migration tool creates a variable name for each TP PCB based on its numeric position within the VAGen PSB. The tool creates a variable name for each DB or GSAM PCB using a combination of the database name from the VAGen PSB, a customer-specified suffix indicating the type of the PCB, and, if necessary, a number to create a unique variable name. The tool also creates a variable to redefine the named DB and GSAM PCBs. The redefinition variable is based on the numeric position of the PCB within the VAGen PSB. This enables the migration tool to use either variable (database name or PCB number) during migration.

Table 141. PSB

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<p>PSB information:</p> <ul style="list-style-type: none"> • Name • PCB information varies by PCB type <ul style="list-style-type: none"> – Number – Type <ul style="list-style-type: none"> - TP - DB - GSAM – Database – Segment – Parent – Index Key <p>Note:</p> <ul style="list-style-type: none"> • The PSB is a separate part type. • The I/O PCB is not explicitly specified, but must be in the DL/I PSB for the IMSVS, IMS BMP, and MVS Batch environments. 	<p>EGL syntax example:</p> <pre>Record <i>psbName</i> type PSBRecord {defaultPSBName = "originalPSBName"; iopcb IO_PCBRecord; { @PCB { pcbType=PCBKind.TP }}; pcb0 IO_PCBRecord { redefines=iopcb }; [otherPCBInformation] end // end <i>psbName</i></pre> <p>Note:</p> <ul style="list-style-type: none"> • The PSBRecord is a record stereotype. • The I/O PCB must be explicitly specified. • EGL uses the @PCB complex property to specify the PCB type. • EGL provides record definitions for the following records: <ul style="list-style-type: none"> – IO_PCBRecord – ALT_PCBRecord – DB_PCBRecord – GSAM_PCBRecord • EGL ignores (removes) the I/O PCB variable when generating for a CICS environment. 	<p>The migration tool only includes the defaultPSBName property if the PSB must be renamed due to a reserved word or because the name started with the # or @ symbol.</p> <p>The migration tool always adds the variable iopcb and the pcb0 redefinition to every PSBRecord.</p>
<p>PCB Type - TP</p> <ul style="list-style-type: none"> • Number 	<pre>ELAALT ALT_PCBRecord {@PCB { pcbType = PCBKind.TP }}; pcb1 ALT_PCBRecord { redefines = ELAALT }; ELAEXP ALT_PCBRecord {@PCB { pcbType = PCBKind.TP }}; pcb2 ALT_PCBRecord { redefines = ELAEXP }; pcb<i>n</i> ALT_PCBRecord { @PCB { pcbType = PCBKind.TP }};</pre> <p>Note:</p> <ul style="list-style-type: none"> • EGL ignores (removes) the alternate PCB variables when generating for a CICS environment. 	<p>The migration tool uses ELAALT and ELAEXP as the names for the first two TP PCBs in the VAGen PSB. The migration tool also creates redefinitions for these two TP PCBs so they can be referred to by number.</p> <p>If there are additional TP PCBs, the migration tool creates a variable name for the TP PCB based on the PCB number within the PSB. This enables the migration tool to use pcb<i>n</i> as the replacement for EZEDLPCB[<i>n</i>].</p>

Table 141. PSB (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
PCB Type - DB <ul style="list-style-type: none"> Number Database Segment Parent Index Key Note: <ul style="list-style-type: none"> The same database name can be used for multiple PCBs in the PSB. 	<pre> <i>DBName_dbSuffix</i> DB_PCBRecord { @PCB { pcbType = PCBKind.DB, pcbName = "<i>DBName</i>", secondaryIndex = "<i>indexKeyName</i>", secondaryIndexItem = "<i>renamedIndexKey</i>", hierarchy = [@Relationship { segmentRecord = segmentName, parentRecord = parentSegmentName }] } }; <i>pcbn</i> DB_PCBRecord { redefines = <i>DBName_dbSuffix</i> }; </pre> Note: <ul style="list-style-type: none"> Because the PSB is a record and each database name becomes a field within the record, each database name must be unique. 	<p>The migration tool creates the variable name for a DB PCB based on the VAGen Database name followed by a suffix. You can specify the suffix with the Stage 2 VAGen Migration Database I/O Preferences.</p> <p>If necessary, to create a unique <i>DBName</i>, the migration tool includes a number. For example: <i>DBName_n_dbSuffix</i>, where <i>n</i> is the same number as in the <i>pcbn</i> redefinition.</p> <p>The migration tool sets the pcbName property to the Database name from the corresponding VAGen PCB. If you use the VAGen Database name as the PCBNAME in your IMS or DL/I PSB, this facilitates testing using the EGL Debugger.</p> <p>The migration tool only includes the secondaryIndexItem property if the VAGen Index Key must be renamed due to a conflict with an EGL reserved word or because the name starts with the # or @ symbol.</p> <p>The migration tool also creates a redefinition of the DB PCB based on the PCB number within the VAGen PSB.</p>
PCB Type - GSAM <ul style="list-style-type: none"> Number Database 	<pre> <i>DBName_gsamSuffix</i> GSAM_PCBRecord { @PCB { pcbType = PCBKind.GSAM }}; <i>pcbn</i> GSAM_PCBRecord { redefines = <i>DBName_gsamSuffix</i> }; </pre> Note: <ul style="list-style-type: none"> Because the PSB is a record and each database name becomes a field within the record, each GSAM database name must be unique. EGL ignores (removes) the GSAM PCB variables when generating for a CICS environment. 	<p>The migration tool creates the variable name for a GSAM PCB based on the VAGen Database name followed by a suffix. You can specify the suffix with the Stage 2 VAGen Migration Database I/O Preferences.</p> <p>If necessary, to create a unique <i>DBName</i>, the migration tool includes a number. For example: <i>DBName_n_gsamSuffix</i>, where <i>n</i> is the same number as in the <i>pcbn</i> redefinition.</p> <p>The migration tool also creates a redefinition of the GSAM PCB based on the PCB number within the VAGen PSB.</p>

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 .eglbl 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:

- 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 migration tool includes the generation option /jspreldir as a comment. 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, except in the following situations:
 - 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 migration tool issues an error message if the part name conflicts with an EGL reserved word.

The control parts section is organized into the following tables:

- General control part information, Table 142 on page 354
- Generation options, Table 143 on page 354
- Generation options: conversion table values, Table 144 on page 370
- Linkage and resource associations: conversion table names, Table 145 on page 371
- Linkage table options for :callLink, Table 146 on page 372
- Linkage table options for :filelink, Table 147 on page 376
- Linkage table options for :crtxlink, Table 148 on page 377
- Linkage table options for :dxfrlink, Table 149 on page 378
- Resource association, Table 150 on page 379
- Link edit options, Table 151 on page 383
- Bind control, Table 152 on page 383
- Part-related symbolic parameters, Table 153 on page 384
- File-related symbolic parameters, Table 154 on page 385

- User-defined symbolic parameters, Table 155 on page 385

Table 142. General control part information

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<p>VAGen control part names:</p> <ul style="list-style-type: none"> • Can include the period (.) in the name. • For bind and link edit parts, any portion of the name after the first period is treated as a suffix. The suffix can be specified in the /bind and /link edit generation options. 	<p>EGL build parts:</p> <ul style="list-style-type: none"> • The period (.) is not valid in a build part name. 	<p>The migration tool changes the period (.) to an underscore (_).</p>
<p>Upper and lower case are not significant in VAGen control part tags and values.</p>	<p>Upper and lower case are significant in EGL control part tags and values.</p>	<p>The migration tool converts the control part tags and values to the correct case required for EGL.</p>

Generation options part

Table 143. Generation options

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<p>VAGen generation options part:</p> <ul style="list-style-type: none"> • 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 <i>not</i> considered to be associates of the generation options part. 	<p>EGL build descriptor part:</p> <ul style="list-style-type: none"> • Contains one or more build descriptor options. • Can be chained using the nextBuildDescriptor build descriptor option • Can only reference other build parts when one of the following criteria is satisfied: <ul style="list-style-type: none"> – The build parts are included in the same .eglbld file. – The build parts are in files that are imported by the .eglbld file. 	<p>If your VAGen control parts are all in the same VisualAge Java package or VisualAge Smalltalk application, the control parts are all placed in the same .eglbld file. In this situation, no import statements are required.</p> <p>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 must add the import statements. EGL validation displays an error message in the Problems view if EGL is unable to resolve references to other control parts.</p>
<p>VAGen generation option values are only enclosed in quotes if they contain special characters for a directory or file name.</p>	<p>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.</p>	<p>The migration tool includes the quotes automatically when it builds the XML source for the .eglbld file.</p>

Table 143. Generation options (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<p>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:</p> <ul style="list-style-type: none"> • /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. 	<p>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:</p> <ul style="list-style-type: none"> • 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. 	<p>The migration tool processes the options in the following way:</p> <ul style="list-style-type: none"> • 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.
<p>/bind=xxxx</p> <p>In VisualAge Generator, xxxx is the suffix of the bind part. The bind part for a program is named <i>pgmname.xxxx</i>, where xxxx is the suffix specified by the /bind option. You might specify a /bind=suffix for either of the following reasons:</p> <ul style="list-style-type: none"> • A special bind is needed for the program because you bind the program into multiple DB2 plans. • VisualAge Generator did not enable you to easily create a bind part with exactly the same name as the program. 	<p>bind="xxxx"</p> <p>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 have the same name, so there is no need to include the bind option.</p> <p>The bind option is only necessary if you generate the same program for multiple runtime environments and require special bind commands for each environment.</p> <p>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.</p>	<p>Because the value for the bind option has different meanings in VisualAge Generator and EGL, the migration tool cannot migrate this option. The migration tool includes /bind as a comment.</p> <p>For more information on how to set the bind build descriptor option, see the following sections:</p> <ul style="list-style-type: none"> • “Establishing a bind control part to use as a template” on page 212. • “Establishing a program-specific bind control part” on page 214.
<p>/checktype=xxxx</p> <p>xxxx is one of the following values:</p> <ul style="list-style-type: none"> • none • low • all 	<p>checkType="xxxx"</p> <p>The following EGL values correspond to the VAGen values:</p> <ul style="list-style-type: none"> • NONE • LOW • ALL 	No special considerations.

Table 143. Generation options (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool 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 values: <ul style="list-style-type: none"> • none • rdo • macro 	cicsEntries="xxxx" The following EGL values correspond to the VAGen values: <ul style="list-style-type: none"> • 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 has one of the following values: <ul style="list-style-type: none"> • 0 or minimum • 1 or info • 2 or logic • 3 or data • 4 or statements Note: <ul style="list-style-type: none"> • 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" The following EGL values correspond to the VAGen values: <ul style="list-style-type: none"> • 0 • 1 • 1 • 1 • 1 Note: <ul style="list-style-type: none"> • 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 143. Generation options (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
/configmapversion="xxx" xxx 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=xxx xxx is the name of a conversion table.	clientCodeSet = "yyy" serverCodeSet = "zzz" yyy and zzz 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 144 on page 370 for the correspondence between the VAGen and EGL values. If the value for /contable=xxx is not in Table 144 on page 370, the migration tool sets both clientCodeSet and serverCode to xxx.
/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=xxx xxx has one of the following values: <ul style="list-style-type: none"> • db2 • oracle • odbc Note: In VisualAge Generator, Oracle and ODBC are only supported for certain workstation platforms.	dbms = "xxx" The following EGL values correspond to the VAGen values: <ul style="list-style-type: none"> • DB2 • ORACLE • DB2 Note: <ul style="list-style-type: none"> • In EGL, Oracle is only supported if you use Java generation. • EGL Java generation supports JDBC instead of ODBC. • EGL also supports the following databases: <ul style="list-style-type: none"> – CLOUDSCAPE – DERBY – INFORMIX – SQLSERVER 	The migration tool changes odbc to DB2 and issues a warning message.
/dbpassword=xxx	sqlPassword = "xxx"	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. EGL validation displays an error message in the Problems view.

Table 143. Generation options (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
/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. EGL validation displays an error message 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.
/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.
/ftptranslationcmdsbc=xxxx	Not supported. EGL only supports TCP/IP for transferring files to the host.	The migration tool includes this option as a comment.
/genauthortimevalues /nogenauthortimevalues	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.

Table 143. Generation options (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
/genout=xxxx	genDirectory = "xxxx"	The migration tool converts /genout and places the result in both the original build descriptor and the new build descriptor part referenced by the secondaryTargetBuildDescriptor option. If you generate for Java, you might need to specify the genProject build descriptor option in addition to or instead of the genDirectory option. genProject is required in these cases: <ul style="list-style-type: none"> • If you generate for HP-UX or SOLARIS • If you generate VGWebTransactions or VGUI records
/genproperties /nogenproperties	genProperties = "GLOBAL" genProperties = "NO" EGL also provides genProperties = "PROGRAM".	The migration tool converts /genproperties to the EGL genProperties = "GLOBAL" option because this is the closest value in terms of what is generated.
/genresourcebundle	genResourceBundle = "YES" "NO"	The migration tool converts /genresourcebundle and places the result in both the original build descriptor and the new build descriptor part referenced by the secondaryTargetBuildDescriptor option.
/genret	genReturnImmediate = "YES" "NO"	No special considerations.
/gentables	genDataTables = "YES" "NO"	No special considerations.
/genuirecords	genVGUIRecords = "YES" "NO"	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.
/initaddws In VisualAge Generator: <ul style="list-style-type: none"> • This option applies to both main and called programs. • 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. 	initNonIODataOnCall = "YES" "NO" In EGL, this option applies only to called programs.	No special considerations.
/initrecd In VisualAge Generator, this option applies to both main and called programs.	initIORecordsOnCall = "YES" "NO" In EGL, this option applies only to called programs.	No special considerations.

Table 143. Generation options (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
/javadestdir=xxxx	destDirectory = "xxxx"	The migration tool converts /javadestdir and places the result in the new build descriptor part referenced by the secondaryTargetBuildDescriptor option.
/javadesthost=xxxx	destHost = "xxxx"	The migration tool converts /javadesthost and places the result in the new build descriptor part referenced by the secondaryTargetBuildDescriptor option.
/javadestpassword=xxxx	destPassword = "xxxx"	The migration tool converts /javadestpassword and places the result in the new build descriptor part referenced by the secondaryTargetBuildDescriptor option.
/javadestuid=xxxx	destUserID = "xxxx"	The migration tool converts /javadestuid and places the result in the new build descriptor part referenced by the secondaryTargetBuildDescriptor option.
/javasystem=xxxx xxxx has one of the following values: <ul style="list-style-type: none"> • AIX • LINUX • OS2 • OS390 • OS400 • SOLARIS • WINNT 	system = "xxxx" The following EGL values correspond to the VAGen values: <ul style="list-style-type: none"> • AIX • LINUX • not supported • USS • ISERIESJ • SOLARIS • WIN 	The migration tool converts the supported /javasystem values and places the result in the new build descriptor part referenced by the secondaryTargetBuildDescriptor option. The tool includes unsupported values as a comment in the original build descriptor part.
/jobcard=xxxx	Not supported. The equivalent function is provided in the following way: <ul style="list-style-type: none"> • The z/OS and iSeries build servers handle the jobcard. These environments ignore the JOBCARD symbolic parameter. • VSE supports the JOBCARD symbolic parameter. 	The migration tool converts this option to the JOBCARD symbolic parameter.

Table 143. Generation options (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
/jobname=xxxx	Not supported. The equivalent function is provided in the following way: <ul style="list-style-type: none"> For z/OS and iSeries, 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. These environments ignore the JOBNAM symbolic parameter. VSE supports the JOBNAM symbolic parameter. 	The migration tool converts this option to the JOBNAM symbolic parameter
/jspreldir="xxxx"	Not supported.	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 meaningful for IBM support to debug the VAGen generator. It had no effect on the generated COBOL.
/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 link edit part. The link edit part for a program is named <i>pgmname.xxxx</i> , where xxxx is the suffix specified by the /linkedit option. You might have specified a /linkedit=suffix for either of the following reasons: <ul style="list-style-type: none"> A special linkedit is needed for the program such as for static link edit to PL/I. VisualAge Generator did not enable you to easily create a link edit part with exactly the same name as the program. 	linkEdit = "xxxx" The meaning of the linkEdit option is not the same as in VisualAge Generator. In EGL, xxxx is the full name of the link edit part. The linkEdit option only needs to be specified if the link edit part name differs from that of the program. In most cases, the program and link edit part 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 and require special link edit commands for each environment.	Because the value for the linkedit option has different meanings in VisualAge Generator and EGL, the migration tool cannot migrate this option. The migration tool includes /linkedit as a comment. For more information, see "Reviewing link edit commands" on page 214.
/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.

Table 143. Generation options (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
/log=xx OR /nolog	imsLogID = "xx" OR include /nolog as a comment Note: This is for IMS.	The migration tool processes this option in the following way: <ul style="list-style-type: none"> • /log=xx is converted to imsLogID = "xx" • /nolog is converted to a comment.
/math=xxxxx xxxxx has one of the following values: <ul style="list-style-type: none"> • cobol • cspae 	math = "xxxxx" The following EGL values correspond to the VAGen values: <ul style="list-style-type: none"> • COBOL • CSPAE 	No special considerations.
/mfsdev = ('deviceName', 'MFSInfo', 'eAI') Note: <ul style="list-style-type: none"> • The VAGen <i>deviceName</i> is used. <i>MFSInfo</i> provides the corresponding MFS information to use for the VAGen device. <i>eAI</i> provides the extended attribute information. • The values for <i>eAI</i> are: EATTR, NOEATTR, and NCD. • Multiple <i>MFSInfo</i> and <i>eAI</i> values can be provided for a single <i>deviceName</i>. • Refer to the <i>VisualAge Generator Server Guide for MVS, VSE, and VM</i> for the details of this generation option. 	<mfsDevice width="nn", height="nn", devStmtParms="MFSInfo", extendedAttributes="eAI" /> Note: <ul style="list-style-type: none"> • The EGL device size (width and height) are used. <i>MFSInfo</i> and <i>eAI</i> provide the same information as in VisualAge Generator. • The values for <i>eAI</i> are: YES, NO, and NCD. • Multiple <i>MFSInfo</i> and <i>eAI</i> values can be provided for a single device size. • Refer to the <i>EGL Generation Guide</i> for the details of this build descriptor option. 	The migration tool converts the VAGen <i>deviceName</i> to the corresponding width and height . For the relationship between the device names and sizes, see Table 95 on page 286. If two <i>deviceNames</i> convert to the same width and height and have the same values specified for <i>MFSInfo</i> and <i>eAI</i> , the migration tool only includes one entry. The migration tool does not change the value of <i>MFSInfo</i> . The migration tool converts the values of <i>eAI</i> to the corresponding EGL values.
/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.

Table 143. Generation options (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool 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.	msgTablePrefix = "xxxx" In EGL, the same considerations for the msgTablePrefix apply as in VisualAge Generator.	The migration tool converts /msgtableprefix and places the result in both the original build descriptor and the new build descriptor part referenced by the secondaryTargetBuildDescriptor option. If you generate a VGUIRecord by itself without generating the program that uses it, you must include the package name with the message table prefix (for example, msgTablePrefix = "packageName.prefixID").
/msp=xxxx xxxx has one of the following values: <ul style="list-style-type: none"> • all • gsam • mfs • seq 	formServicePgmType = "xxxx" The following EGL values correspond to the VAGen values: <ul style="list-style-type: none"> • ALL • GSAM • MFS • SEQ Note: This is for IMS BMP and z/OS Batch.	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 In VisualAge Generator, the /packagename generation option is used when generating Java, Java wrappers, or the Java components for Web Transactions.	wrapperPackageName = "xxxx" In EGL, wrapperPackageName is used only when the enableJavaWrapperGen build descriptor option is set to "ONLY" or "YES"; otherwise, wrapperPackageName is ignored.	No special considerations.
/possign=x x has one of the following values: <ul style="list-style-type: none"> • f • c 	positiveSignIndicator = "x" The following EGL values correspond to the VAGen values: <ul style="list-style-type: none"> • F • C Note: This is for ISERIESC.	No special considerations.
/prep	prep = "YES" "NO"	No special considerations.
/preprofile	buildPlan = "YES" "NO"	No special considerations.

Table 143. Generation options (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<p>/printdest=xxxx</p> <p>xxxx has one of the following values:</p> <ul style="list-style-type: none"> ezeprint termid 	<p>printDestination = "xxxx"</p> <p>The following EGL values correspond to the VAGen values:</p> <ul style="list-style-type: none"> PROGRAMCONTROLLED TERMINALID 	No special considerations.
<p>/project="xxxx"["version"]</p> <p>xxxx is the name of a VisualAge for Java project, and <i>version</i> is the version name for the specified project.</p>	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.
/projectid=xxxx	projectID = "xxxx"	No special considerations.
/recovery	<p>restoreCurrentMsgOnError = "YES" "NO"</p> <p>Note: This is for IMS.</p>	No special considerations.
<p>/resource=xxxx</p> <p>xxxx is the name of a VAGen resource associations part.</p>	<p>resourceAssociations = "xxxx"</p> <p>xxxx the name of an EGL resource associations part.</p>	No special considerations.
/resourcebundlelocale=xxxx	resourceBundleLocale = "xxxx"	The migration tool converts /resourcebundlelocale and places the result in both the original build descriptor and the new build descriptor part referenced by the secondaryTargetBuildDescriptor option.
/resvword=xxxx	reservedWord = "xxxx"	No special considerations.
/rt=xxxx	returnTransaction = "xxxx"	No special considerations.
/runfile	genRunFile = "YES" "NO"	No special considerations.
/sendtranslationcmddbcs=xxxx	<p>Not supported.</p> <p>Note: EGL only supports TCP/IP for transferring files to the host.</p>	The migration tool includes this option as a comment.
/session=xxxx	<p>Not supported.</p> <p>Note: EGL only supports TCP/IP for transferring files to the host.</p>	The migration tool includes this option as a comment.
/setfull	setFormItemFull = "YES" "NO"	No special considerations.
/sp	checkToTransaction = "YES" "NO"	No special considerations.
<p>/spa=xxxx,ADF,yyyy</p> <p>Note: ADF is optional. yyyy is optional so all of the following combinations of options are valid in VisualAge Generator:</p> <p>/spa=xxxx</p> <p>/spa=xxxx,ADF,yyyy</p> <p>/spa=xxxx,,yyyy</p>	<p>In EGL, there are 3 separate options:</p> <p>spaSize = "xxxx"</p> <p>spaADF = "YES" "NO"</p> <p>spaStatusBytePosition = "yyyy"</p>	<p>The migration tool splits the /spa option into the 3 EGL options.</p> <p>The migration tool only includes spaADF if the value is YES.</p>

Table 143. Generation options (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
/spzero Note: This option is supported for COBOL generation only.	spacesZero = "YES" "NO" Note: This option is supported for COBOL generation, Java generation, and debug.	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. EGL validation displays an error message in the Problems view.
/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. EGL validation displays an error message in the Problems view.
/sqlvalid	validateSQLStatements = "YES" "NO"	No special considerations.
/symparm=pppppppp,'vvvv' <ul style="list-style-type: none"> • 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 in the following way: <ul style="list-style-type: none"> • The migration tool converts any VAGen-defined symbolic parameters to the corresponding EGL symbolic parameter. • If there is no corresponding EGL symbolic parameter, the migration tool converts the VAGen-defined symbolic parameter 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,'xxx'	transferErrorTransaction = "xxx"	No special considerations.
/SYMPARM=EZONEAS2,'xxx'	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 143. Generation options (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<p>/system=xxxx</p> <p>xxxx has one of the following values:</p> <ul style="list-style-type: none"> • MVS BATCH • MVSCICS • IMS BMP • IMSVS • AIX • JAVALINUX • JAVAOS390 • JAVAOS400 • JAVAWINNT • JAVAWRAPPER • WINNT • LINUX • OS400 • HP • SOLARIS • VSE BATCH • VSE CICS <p>The following environments can also be specified in VAGen, but are not converted by the migration tool: JAVA, JAVAGUI, WINGUI, OS2GUI, OS2, OS2CICS, AIXCICS, NTCICS, SOLACICS, TSO, VMCMS, VMBATCH</p>	<p>system = "xxxx"</p> <p>The following EGL values correspond to the VAGen values:</p> <ul style="list-style-type: none"> • ZOS BATCH • CICS for z/OS • IMS BMP • IMSVS • AIX • LINUX • USS • ISERIESJ • WIN • WIN • WIN • LINUX • ISERIESC • HPUX • SOLARIS • VSE BATCH • VSE CICS 	<p>The migration tool processes this option in the following way:</p> <ul style="list-style-type: none"> • If /system=xxxx 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 sets the following EGL build descriptor options: <ul style="list-style-type: none"> – enableJavaWrapperGen = "ONLY". This specifies that the you want to generate only the Java wrapper for a program. – wrapperCompatibility = "V4". This option specifies that the Java wrapper must be compatible with VisualAge Generator. • For the COBOL environments, the migration tool issues a warning message that you need to set the destPort build descriptor option.
<p>/targetnls=xxx</p> <p>xxx is a 3-character national language code.</p>	<p>targetNLS="xxx"</p> <p>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.</p>	<p>The migration tool converts /targetnls and places the result in both the original build descriptor and the new build descriptor part referenced by the secondaryTargetBuildDescriptor option.</p> <p>The migration tool uses the VAGen value as the targetNLS value. If the value is ENP, EGL validation displays an error message 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.</p>

Table 143. Generation options (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<p>/templates=xxxx</p> <p>In VisualAge Generator, templates are used to generate the preparation and runtime JCL, as well as to generate CICS transaction and program entries.</p>	<p>templateDir = "xxxx"</p> <p>In EGL, build scripts replace preparation templates for z/OS and iSeries. The only templates that are used are to produce runtime JCL for the ZOSBATCH, IMSBMP, and VSEBATCH runtime environments, runtime CL for the ISERIESC target environment, and preparation JCL for the VSE runtime environment.</p>	No special considerations.
<p>/trace=xxxx,yyyy</p> <p>xxxx has one of the following values:</p> <ul style="list-style-type: none"> • none • sqlerr • sqlio <p>yyyy is optional. If yyyy is present, it is set to stmt.</p> <p>Any combination of none, sqlerr, or sqlio, with or without stmt, is valid.</p>	<p>/trace splits into multiple build descriptor options:</p> <ul style="list-style-type: none"> • If sqlerr is included, sqlErrorTrace = "YES" • If sqlio is included, sqlIOTrace = "YES" • if stmt is included, statementTrace = "YES" 	No special considerations.
<p>/transfertype=xxxx</p> <p>xxxx has one of the following values:</p> <ul style="list-style-type: none"> • tcpip • sna 	<p>Not supported.</p> <p>Note: EGL only supports TCP/IP for transferring files to the host.</p>	The migration tool includes this option as a comment.
<p>/transid=primaryID,restartID</p> <p>In VisualAge Generator, /transid=,restartID is valid, with the primary transaction defaulting to the first 4 characters of the program name.</p>	<p>/transid splits into multiple build descriptor options:</p> <ul style="list-style-type: none"> • If <i>primaryID</i> is included, startTransactionID = "primaryID" • If <i>restartID</i> is included, restartTransactionID = "restartID" 	No special considerations.
<p>/twooff=nnnn</p>	twoOffset = "nnnn"	No special considerations.
<p>/unload</p> <p>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.</p>	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.

Table 143. Generation options (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
/vselib=xxxx	vseLibrary = "xxxx"	No special considerations.
/workdb=xxxx xxxx has one of the following values: <ul style="list-style-type: none"> • aux • main • dli • sql 	workDBType = "xxxx" The following EGL values correspond to the VAGen values: <ul style="list-style-type: none"> • AUX • MAIN • DLI • SQL 	No special considerations.
Not used.	vagCompatibility = "YES"	Based on the Do not set compatibility mode migration preference, the migration tool adds or omits this option to every build descriptor part.
Not used.	v60NumWithCharBehavior = "YES" v60SQLNullableBehavior = "YES" checkIndices = "NO" truncateExtraDecimals = "YES" v60DecimalBehavior = "YES"	The migration tool always adds these options to every build descriptor part. This technique is used to explicitly show the default values that must be used to preserve VAGen behavior.
For COBOL generation, in programs that do not use print forms, the decimal symbol and numeric separator symbol default to the values in the language-dependent options module used at runtime. In programs that use print forms, the decimal symbol defaults to a period and the numeric separator symbol defaults to a comma. For Java generation, the decimalSymbol is a runtime property. The numeric separator is the opposite (for example, if the decimalSymbol is a period, the numeric separator symbol is a comma).	decimalSymbol = "x" separatorSymbol = "y" x and y are one of the following symbols: <ul style="list-style-type: none"> • a period (.) • a comma (,) x and y cannot have the same value. For COBOL generation, if you specify this information at generation time, the generation information takes precedence. If you do not specify it at generation time, for programs that do not use print forms, the symbols are set in the same way as in VisualAge Generator. For programs that use print forms, the symbols default to a period as the decimal symbol and a comma as a separator symbol. For Java generation, if you specify this information at generation time it is used to set the runtime properties. If you do not specify it at generation time, you can set it in the properties file used at runtime.	The migration tool does not set the decimalSymbol or the separatorSymbol . You can add these options to your build descriptor part. Alternatively, use one of the following techniques: <ul style="list-style-type: none"> • For programs that you generate to COBOL, and that do not use print forms, you can continue to use the language-dependent options module as you did in VisualAge Generator. • For Java generation, you can set the information directly in the properties file used at runtime.

Table 143. Generation options (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
Not used.	<p>destPort = "xxxx"</p> <p>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 COBOL generation target environments.</p>	<p>The migration tool does not set destPort. The default value varies by target environment in the following ways:</p> <ul style="list-style-type: none"> • For z/OS environments, there is no default value for destPort. You must add the destPort build descriptor option and the value must match the value you use in the JCL that starts the z/OS build server. The sample JCL for starting a z/OS build server uses port 5555. • For iSeries environments, there is no default value for destPort. You must add the destPort build descriptor option. The value must match the value used by the iSeries build server. • For VSE environments, the default value for the destPort is 21. You only need to specify the destPort build descriptor option if the value is different from 21.
Not used.	<p>genProject = "xxxx"</p>	<p>If you generate for Java, you might need to specify the genProject build descriptor option in addition to or instead of the genDirectory option. genProject is required in the following cases:</p> <ul style="list-style-type: none"> • If you generate for HP-UX or SOLARIS; or • If you generate VGWebTransactions or VGUI records.
Not supported.	<p>tempDirectory = "xxxx"</p>	<p>If you generate VGUI records, the tempDirectory option enables you to specify the directory where the generated JSP is placed if there is already a JSP by the same name in the genProject directory. If tempDirectory is not specified, the JSP is generated into the genProject directory, but with a name of newxxxx.JSP, where xxxx is the VGUI record name.</p>
<p>Not supported.</p> <p>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.</p>	<p>eliminateSystemDependentCode = "YES" "NO"</p> <p>In EGL, if your program checks the value of sysVar.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.</p>	<p>The migration tool does not set eliminateSystemDependentCode. The default value is "YES".</p>

Table 143. Generation options (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
Not used.	<code>sessionBeanID = "xxxx"</code>	The migration tool does not set sessionBeanID . If you are generating Java or Java wrappers, see the EGL online help 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.	<code>sqlJDBCDriverClass = "xxxx"</code> <code>sqlValidationConnectionURL = "xx"</code> In EGL, you can specify this information at generation time or at runtime.	The migration tool does not set the sqlJDBCDriverClass and sqlValidationConnectionURL build descriptor options. If you want to specify these values at generation time, use one of the following techniques: <ul style="list-style-type: none"> 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 set the genProperties build descriptor option to "GLOBAL" or "PROGRAM" so that the properties file is generated. If you want to specify the value at runtime, you can modify the runtime properties in the properties file.

Conversion table names used in generation option parts

Table 144. Generation options: conversion table names

Language	Conversion Table VAGen <i>/contable value</i>	EBCDIC Character Set EGL <i>serverCodeSet</i>	ASCII Character Set EGL <i>clientCodeSet</i>
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

Table 144. Generation options: conversion table names (continued)

Language	Conversion Table VAGen /contable value	EBCDIC Character Set EGL serverCodeSet	ASCII Character Set EGL clientCodeSet
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 listed in the previous rows of this table)	XXXXXXXX	XXXXXXXX	XXXXXXXX

Conversion table names used in linkage table and resource associations parts

The migration tool converts the VAGen /contable option to the EGL **conversionTable** attribute in the following way:

- If the first 4 characters of the conversion table name are "CSOX" (AIX Server) or "CSOI" (OS/2 or Windows Server), the migration tool converts the first 4 characters to "CSOJ" and does not change the remaining portion of the name. The conversion table names converted for an OS/2 Server might result in invalid EGL conversion table names or in a valid (but incorrect) conversion table name.
- If the first 4 characters of the conversion table name are "CSOE" (MVS or OS/400 Server), the migration tool does not change the conversion table name.
- If the first 5 characters of the conversion table name are "ELACN" or "ELAAX", the migration tool converts the table name as shown in Table 145.
- If the name is not found in the table or follows any other naming convention, the migration tool converts the table name "as is".

Table 145. Linkage table and resource associations parts: conversion table values

Language	VAGen /contable Local: ASCII Remote: EBCDIC	EGL conversion Table	VAGen /contable Local: Windows Remote: AIX, HP-UX, or Solaris	EGL conversion Table
Arabic	ELACNARA	CSOE420	ELAAXARA	CSOJ1046
Chinese, simplified	ELACNCHS	CSOE935	BINARY	CSOJ1381

Table 145. Linkage table and resource associations parts: conversion table values (continued)

Language	VAGen /contable Local: ASCII Remote: EBCDIC	EGL conversion Table	VAGen /contable Local: Windows Remote: AIX, HP-UX, or Solaris	EGL conversion Table
Chinese, simplified	ELACNGBK	CSOE935	Not supported	CSOJ1386
Chinese, traditional	ELACNCHT	CSOE937	BINARY	CSOJ950
Danish	ELACNDKN	CSOE277	ELAAX850	CSOJ850
Eastern European	ELACN870	CSOE870	ELAAX912	CSOJ852
English (UK)	ELACN285	CSOE285	ELAAX850	CSOJ850
English (US)	ELACNENU	CSOE037	ELAAX437	CSOJ850
Finnish	ELACNFIN	CSOE298	ELAAX850	CSOJ850
French	ELACNFRA	CSOE297	ELAAX850	CSOJ850
German	ELACNDEU	CSOE273	ELAAX850	CSOJ850
Greek	ELACNGRE	CSOE875	ELAAXGRE	CSOJ813
Hebrew	ELACNHEB	CSOE424	ELAAXHEB	CSOJ856
Italian	ELACNITA	CSOE280	ELAAX850	CSOJ850
Japanese, Katakana	ELACNJPN	CSOE930	BINARY	CSOJ943
Japanese, Latin	ELACNJPL	CSOE939	BINARY	CSOJ943
Korean	ELACNKOR	CSOE933	BINARY	CSOJ1363
Norwegian	ELACNDKN	CSOE277	ELAAX850	CSOJ850
Portuguese	ELACNPTB	CSOE037	ELAAX850	CSOJ850
Russian	ELACNCYR	CSOE1025	ELAAXCYR	CSOJ866
Spanish	ELACNESP	CSOE284	ELAAX850	CSOJ850
Swedish	ELACNSWE	CSOE278	ELAAX850	CSOJ850
Swiss German	ELACNDES	CSOE500	ELAAX850	CSOJ850
Turkish	ELACNTUR	CSOE1026	ELAAXTUR	CSOJ920

Linkage table parts

The linkage table parts are Calllink, Filelink, Crtxlink, and Dxfrlink.

callLink

Table 146. Linkage table options for :callLink

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
:callLink	callLink	No special considerations.

Table 146. Linkage table options for :callLink (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<p>linktype=xxxx</p> <p>xxxx has one of the following values:</p> <ul style="list-style-type: none"> • dynamic • static • cicslink • remote • csocall • sessionejb 	<p>Type of call.</p> <p>The following EGL values correspond to the VAGen values</p> <ul style="list-style-type: none"> • localCall • localCall • localCall • remoteCall • remoteCall • ejbCall 	<p>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 entry.</p>
<p>applname=programName</p> <p>programName is the name of the program being called. Wildcards are permitted.</p>	<p>pgmName = "programName"</p>	<p>No special considerations.</p>
<p>externalname=applname</p>	<p>alias = "applname"</p>	<p>If your VAGen program had to be renamed because the name was an EGL reserved word, you can use the alias property either in the program definition or in the linkage options part 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 are called by the VAGen program.</p>
<p>package=packageName</p>	<p>package = "packageName"</p>	<p>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.</p>
<p>library=libraryName OR dllname=libraryName</p> <p>In VisualAge Generator, library and dllname are treated as synonyms.</p>	<p>library = "libraryName"</p>	<p>The migration tool merges the VAGen library or dllname into the EGL library property.</p>
<p>linktype=xxxx</p> <p>xxxx has one of the following values:</p> <ul style="list-style-type: none"> • dynamic • static • cicslink 	<p>linkType = "xxxx"</p> <p>The following EGL values correspond to the VAGen values:</p> <ul style="list-style-type: none"> • DYNAMIC • STATIC • CICSLINK 	<p>No special considerations.</p>

Table 146. Linkage table options for :callLink (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<p>parmform=xxxx</p> <p>xxxx has one of the following values:</p> <ul style="list-style-type: none"> • oslink • commptr • commdata • cicsoslink 	<p>parmForm = "xxxx"</p> <p>The following EGL values correspond to the VAGen values:</p> <ul style="list-style-type: none"> • OSLINK • COMMPTR • COMMDATA • CICSOSLINK 	<p>No special considerations.</p>
<p>contable=xxxx</p> <p>xxxx has one of the following values:</p> <ul style="list-style-type: none"> • a conversionTableName • * • EZECONVT • BINARY • NONE 	<p>conversionTable = "xxxx"</p> <p>The following EGL values correspond to the VAGen values:</p> <ul style="list-style-type: none"> • conversionTableName • * • PROGRAMCONTROLLED • not supported • not supported 	<p>The migration tool uses the information in "Conversion table names used in linkage table and resource associations parts" on page 371 to convert the conversionTable property for the EGL callLink element.</p> <p>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. EGL validation displays an error message in the Problems view. You must correct the error by editing the .eglbld file and selecting the supported value that you want to use.</p> <p>The migration tool omits the conversionTable property if the VAGen contable=NONE.</p>
<p>location=xxxx</p> <p>xxxx has one of the following values:</p> <ul style="list-style-type: none"> • systemName • EZELOC 	<p>location = "xxxx"</p> <p>The following EGL values correspond to the VAGen values:</p> <ul style="list-style-type: none"> • systemName • PROGRAMCONTROLLED 	<p>No special considerations.</p>

Table 146. Linkage table options for :callLink (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
remotecomtype=xxxx xxxx has one of the following values: <ul style="list-style-type: none"> • appcims • ca400 • cicsclient • dce • dcesecure • direct • exci • ipc • java400 • lu2 • tcpip 	remoteComType = "xxxx" The following EGL values correspond to the VAGen values: <ul style="list-style-type: none"> • not supported • not supported • CICSECI • not supported • not supported • DIRECT • not supported • DISTINCT • JAVA400 • not supported • TCPIP 	<p>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.</p> <p>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. EGL validation displays an error message in the Problems view until you correct the callLink entry.</p> <p>If you decide to use CICSSSL, you must add the ctgPort, ctgLocation, ctgKeyStore, and ctgKeyStorePassword properties to the EGL callLink entry.</p> <p>If you decide to use CICSJ2C, you must add the pgmName, conversionTable, remotePgmType, luwControl, remoteBind, location, and parmForm properties to the EGL callLink entry.</p> <p>The migration tool migrates APPCIMS "as is" because it is not supported and the values of other properties are quite different. The best replacement for APPCIMS is IMSTCP.</p>
remoteapptype=xxxx xxxx has one of the following values: <ul style="list-style-type: none"> • vg • nonvg • vgjava • itf 	remotePgmType = "xxxx" The following EGL values correspond to the VAGen values: <ul style="list-style-type: none"> • EGL • EXTERNALLYDEFINED • not applicable • not supported 	<p>If the VisualAge Generator remoteapptype=vgjava, the migration tool migrates the :callLink entry, but omits the remotePgmType property.</p> <p>If remoteapptype=itf, the migration tool turns the entire :callLink entry into a comment.</p>
serverid=serverName	serverID="serverName"	No special considerations.
luwcontrol=xxxx xxxx has one of the following values: <ul style="list-style-type: none"> • client • server 	luwControl = "xxxx" The following EGL values correspond to the VAGen values: <ul style="list-style-type: none"> • CLIENT • SERVER 	No special considerations.

Table 146. Linkage table options for :callLink (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
remotebind=xxxx xxxx has one of the following values: <ul style="list-style-type: none"> • generation • runtime 	remoteBind = "xxxx" The following EGL values correspond to the VAGen values: <ul style="list-style-type: none"> • 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.
bitmode=nn nn has one of the following values: <ul style="list-style-type: none"> • 16 • 32 	Not supported.	The migration tool includes this option as a comment.
binform=xxxx xxxx has one of the following values: <ul style="list-style-type: none"> • 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 isNoRefresh = YES property on the EGL call statement. 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 these properties. ctgKeyStore and ctgKeyStorePassword are required if you decide to use remoteComType = "CICSSL".
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 147. Linkage table options for :filelink

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
:filelink	fileLink	No special considerations.

Table 147. Linkage table options for :filelink (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<p>linktype=xxxx</p> <p>xxxx has one of the following values:</p> <ul style="list-style-type: none"> • local • remote <p>In VisualAge Generator, the default is local.</p>	<p>Type of file.</p> <p>The following EGL values correspond to the VAGen values:</p> <ul style="list-style-type: none"> • localFile • remoteFile 	<p>If the VAGen linktype is not specified, the migration tool converts to localFile.</p>
<p>filename=fileName</p> <p>fileName is the name of a file in a VAGen record definition. Wildcards are permitted.</p>	<p>fileName = "fileName"</p>	<p>No special considerations.</p>
<p>contable=xxxx</p> <p>xxxx has one of the following values:</p> <ul style="list-style-type: none"> • a conversionTableName • * • EZECONVT • BINARY 	<p>conversionTable="xxxx"</p> <p>The following EGL values correspond to the VAGen values:</p> <ul style="list-style-type: none"> • conversionTableName • * • PROGRAMCONTROLLED • not supported 	<p>The migration tool uses the information in "Conversion table names used in linkage table and resource associations parts" on page 371 to convert the conversionTable property for the EGL fileLink element.</p> <p>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. EGL validation displays an error message in the Problems view. You must correct the error by editing the .eglbld file and selecting the supported value that you want to use.</p>
<p>location=xxxx</p> <p>xxxx has one of the following values:</p> <ul style="list-style-type: none"> • CICS • EZELOC 	<p>locationSpec="xxxx"</p> <p>The following EGL values correspond to the VAGen values:</p> <ul style="list-style-type: none"> • CICS • PROGRAMCONTROLLED 	<p>No special considerations.</p>

Crtxlink

Table 148. Linkage table options for :crtxlink

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
:crtxlink	asynchLink	No special considerations.
<p>linktype=xxxx</p> <p>xxxx has one of the following values:</p> <ul style="list-style-type: none"> • local • remote <p>Note: In VisualAge Generator the default is local.</p>	<p>Type of invocation.</p> <p>The following EGL values correspond to the VAGen values:</p> <ul style="list-style-type: none"> • localAsynch • remoteAsynch 	<p>If the VAGen linktype is not specified, the migration tool converts to localAsynch.</p>

Table 148. Linkage table options for :crtxlink (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
recdname=recordName <i>recordName</i> is the name of a VAGen record definition. Wildcards are permitted.	recordName = " <i>recordName</i> "	No special considerations.
contable=xxxx xxxx has one of the following values: <ul style="list-style-type: none"> • a conversionTableName • * • EZECONVT • BINARY 	conversionTable = " <i>xxxx</i> " The following EGL values correspond to the VAGen values: <ul style="list-style-type: none"> • conversionTableName • * • PROGRAMCONTROLLED • not supported 	The migration tool uses the information in "Conversion table names used in linkage table and resource associations parts" on page 371 to convert the conversionTable property for the EGL asynchLink element. 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. EGL validation displays an error message 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 has one of the following values: <ul style="list-style-type: none"> • CICS • EZELOC 	locationSpec = " <i>xxxx</i> " The following EGL values correspond to the VAGen values: <ul style="list-style-type: none"> • CICS • PROGRAMCONTROLLED 	No special considerations.
package=packageName	package = " <i>packageName</i> "	No special considerations.

Dxfrlink

Table 149. 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 <i>programName</i> is the name of the program that is transferring with a DXFR to another program. Wildcards are not permitted.	fromPgm = " <i>programName</i> "	No special considerations.
toappl=programName2 <i>programName2</i> is the name of the program to which the transfer is occurring.	toPgm = " <i>programName2</i> "	No special considerations.

Table 149. Linkage table options for :dxfrlink (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<p>linktype=xxxx</p> <p>xxxx has one of the following values:</p> <ul style="list-style-type: none"> dynamic static noncsp 	<p>linkType = "xxxx"</p> <p>The following EGL values correspond to the VAGen values:</p> <ul style="list-style-type: none"> DYNAMIC STATIC EXTERNALLYDEFINED 	<p>If you previously specified NONCSP for a VAGen DXFR statement, you can continue to use the isExternal = "YES" property on the EGL transfer to program statement. Alternatively, you can obtain the same support by specifying linkType = "EXTERNALLYDEFINED" on the transferToProgram element for the program to which you are transferring.</p>
Not supported.	<p>alias = "applname"</p>	<p>If your VAGen program had to be renamed because the name was an EGL reserved word, you can use the alias property either in the program definition or in the linkage options part 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 you transfer to from the VAGen program.</p>

resource associations part

Table 150. Resource association

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<p>In VisualAge Generator, a resource associations 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.</p> <p>The resource associations part can also specify how print output is to be implemented for a specific target environment.</p> <p>When generating a program, the fileName for each indexed, serial, relative or print output is matched to the resource associations part. The first entry that matches based on the fileName and generation target environment is the entry that is used for that file.</p>	<p>The EGL resource associations 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.</p> <p>The resource associations part can also specify how print output is to be implemented for a specific target environment.</p> <p>When generating a program, the fileName property for each indexed, serial, relative or print output is matched to the resource associations part. The first entry that matches based on the fileName and generation target environment is the entry that is used for that file.</p>	<p>No special considerations.</p>
For VisualAge Generator, if you generate C++, 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 150. Resource association (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
file = <i>fileName</i> EZEPRINT	fileName=" <i>fileName</i> " " <i>printer</i> "	No special considerations.
<p>/system=<i>targetSystem</i></p> <p><i>targetSystem</i> has one of the following values:</p> <ul style="list-style-type: none"> • AIX * • AIXCICS * • HP-UX * • IMSBMP • IMSVS • LINUX ** • MVS BATCH • MVSCICS • NTCICS * • OS2 * • OS2CICS • OS400 • SCO * • SOLACICS * • SOLARIS * • TSO • VMCMS • VMBATCH • VSEBATCH • VSECICS • WINNT ** <p>Note:</p> <ul style="list-style-type: none"> • * — 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). 	<p>This is the EGL target environment.</p> <p>The following EGL values correspond to the VAGen values:</p> <ul style="list-style-type: none"> • aix • not supported • hpux • imsbmp • imsvs • linux • zosbatch • zoscics • not supported • not supported • not supported • iseriesc • not supported • not supported • solaris • not supported • not supported • not supported • vsebatch • vsecics • win <p>Note: Wildcards are not supported.</p>	<p>The migration tool processes the /system option in the following way:</p> <ul style="list-style-type: none"> • 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 associations 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 150. Resource association (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
<p><i>/filetype=fileType</i></p> <p><i>fileType</i> has one of the following values:</p> <ul style="list-style-type: none"> • BTREIVE • GSAM • IBMCOBOL • MFCOBOL • MMSGQ • MQ • OS2COBOL • SEQ • SEQRS • SMSGQ • SPOOL • TEMPAUX • TEMPMAIN • TRANSIENT • VSAM • VSAMRS 	<p>The EGL file type.</p> <p>The following EGL values correspond to the VAGen values:</p> <ul style="list-style-type: none"> • not supported • gsam • ibmcobol • not supported • mmsgq • mq • not supported • seq or seqws • seqrs • smsgq • spool • tempaux • tempmain • transient • vsam • vsamrs 	<p>The migration tool processes the <i>/filetype</i> option in the following way:</p> <ul style="list-style-type: none"> • If the <i>/filetype</i> option is omitted from the VAGen resource association entry, the migration tool uses default as the EGL file type. • If the <i>/system</i> option specifies a host target environment, the migration tool converts the VAGen SEQ file type to the EGL seq file type. • If the <i>/system</i> option is a workstation environment, the migration tool converts the VAGen SEQ file type to the EGL seqws file type. • For unsupported file type values, if the resource association is for a <i>/system</i> that is supported, the migration tool creates an EGL resource association entry using the VAGen file type and issues an error message. EGL validation displays an error message in the Problems view. You must fix this error before you can use the EGL resource associations part.
<i>/sysname=systemName</i>	systemName = " <i>systemName</i> "	The migration tool converts any symbolic parameters that are used within the <i>/sysname</i> option to the corresponding EGL replacement symbolic parameter.
<p><i>/replace</i></p> <p><i>/noreplace</i></p>	<p>replace = "YES"</p> <p>replace = "NO"</p>	No special considerations.
<p><i>/dup</i></p> <p><i>/nodup</i></p>	<p>duplicates = "YES"</p> <p>duplicates = "NO"</p> <p>Note: This is for ISERIESC.</p>	No special considerations.
<p><i>/commit</i></p> <p><i>/nocommit</i></p> <p>These options are only used for the OS/400 target environment.</p>	<p>commit = "YES"</p> <p>commit = "NO"</p> <p>Note: This is for ISERIESC.</p>	No special considerations.
<p><i>/blksize=xxxx,yyyy,zzzz</i></p> <p>In VisualAge Generator, this option is only used for VSE target environments.</p>	blockSize = " <i>xxxx,yyyy,zzzz</i> "	No special considerations.
<p><i>/sysnum=xxxx</i></p> <p>In VisualAge Generator, this option is only used for VSE target environments.</p>	systemNumber = " <i>xxxx</i> "	No special considerations.

Table 150. Resource association (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
/label /nolabel In VisualAge Generator, this option is only used for VSE target environments.	standardLabel = "YES" standardLabel = "NO"	No special considerations.
/pcbno= <i>n</i> This is only valid for IMSVS or IMSBMP target environments or for MVS Batch if the file type is GSAM.	pcbName = "pcb <i>n</i> "	The migration tool converts the PCB number to a name by concatenating the literal "pcb" and the number.
/noff There is no /FF option in VisualAge Generator. This option is only supported in VAGen resource association files.	FormFeedOnClose = "NO" "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" "NO"	The migration tool converts /text to text = "YES".
/contable=xxxx xxxx has one of the following values: • a conversionTableName • EZECONVT This option is only supported in VAGen resource association files.	conversionTable = "xxxx" The following EGL values correspond to the VAGen values: • a conversionTableName • PROGRAMCONTROLLED	The migration tool uses the information in "Conversion table names used in linkage table and resource associations parts" on page 371 to convert the conversionTable property for the EGL resource association element.
/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 property.
/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.

Link edit part

Table 151. 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 <i>programName.suffix</i> , 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 linkEdit build descriptor option. If you have multiple runtime environments, you might need a different link edit part for each environment. In this case, all except one of the link edit part names must differ from the program name. To refer to the different link edit part names, 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 <i>.suffix</i> to <i>_suffix</i> because periods (.) are not valid characters in EGL part names.
A VAGen link edit part contains the link edit statements needed for link editing a program during the preparation process in a host environment.	In EGL, a link edit part contains the link edit statements needed for link editing a program during the build process in a host environment.	The migration tool does the following things: <ul style="list-style-type: none"> • 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 152. 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 <i>programName.suffix</i> , 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 build descriptor option. If you have multiple runtime environments, you might need a different bind control part for each environment. In this case, all except one of the bind control part names must differ from the program name. To refer to the different bind control part names, 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 <i>.suffix</i> to <i>_suffix</i> because periods (.) are not valid characters in EGL part names.

Table 152. Bind control part (continued)

VisualAge Generator 4.5	EGL produced by the migration tool	Migration tool considerations
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 z/OS host environment.	<p>The migration tool does the following things:</p> <ul style="list-style-type: none"> • 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 153. Part-related symbolic parameters

Part-related symbolic parameters	Corresponding EGL symbolic parameter
EZECOBOLTYPE	Not supported
EZEDATA	DATA or EZEDATA in the build plan
EZEDBCS	Not supported
EZEDESTLIB	EZEDESTLIBRARY
EZEDESTNAME	Not supported
EZEDLI	EZEDLI — DL/I only
EZEENTRY	Not supported
EZEENV	SYSTEM or EZEENV in the build plan
EZEGDATE	EZEGDATE
EZEGENOUT	Not supported
EZEGMBR	EZEGMBR
EZEGTIME	EZEGTIME
EZEJOB	Not supported
EZEMBR	In a JCL script, link edit, or bind part, EZEALIAS. Otherwise, EZEMBR
EZEMBRPATH	Not supported
EZEMSG	Not supported
EZENLS	EZENLS
EZEPID	EZEPID
EZEPREPDESTACCOUNT	Not supported
EZEPREPDESTHOST	Not supported
EZEPREPDESTDIR	Not supported
EZEPREPDESTPASSWORD	Not supported
EZEPREPDESTUID	EZEDESTUSERID

Table 153. Part-related symbolic parameters (continued)

Part-related symbolic parameters	Corresponding EGL symbolic parameter
EZEPREPFTPCMDSBCS	Not supported
EZEPREPFTPCMDBCS	Not supported
EZEPRESENDCMDDBCS	Not supported
EZEPRESESSION	Not supported
EZEPREPS	Not supported
EZEPREPSQLDB	Not supported
EZEPREWORKDB	Not supported
EZEPSB	Not supported — DL/I and IMS only
EZEPTYPE	Not supported
EZESQL	EZESQL
EZETBLNAME	Not supported
EZETPROC	Not supported
EZETRAN	EZETRAN
EZETRANSFERTYPE	Not supported
EZETRO	Not supported
EZETWASIZE	Not supported
EZEUSERID	Not supported
EZEVMLoadLIB	Not applicable — VM only
EZEVSELIB	EZEVSELIB — VSE only
EZEXAPP	Not supported.

Table 154. File-related symbolic parameters

File-related symbolic parameters	Corresponding EGL symbolic parameter
EZEBLK	EZEBLK
EZEDBD	Not supported
EZEDD	EZEDD
EZEDLBL	EZEDLBL — VSE only
EZEDSN	EZEDSN
EZELRECL	EZELRECL
EZERECFM	EZERECFM

Table 155. User-defined symbolic parameters

User-defined symbolic parameters	Corresponding EGL symbolic parameter
COB2LIB	COBCICS
COBLIST	Not supported
DBDLIB	DBDLIB — DL/I only
DSNLOAD	DSNLOAD
DSYS	DSYS
ELA	ELA

Table 155. User-defined symbolic parameters (continued)

User-defined symbolic parameters	Corresponding EGL symbolic parameter
EZALTXTR	Special migration to normal build descriptor option; see Generation Options section, transferErrorTransaction = "xxx"
EZONEAS2	Special migration to normal build descriptor option; see Generation Options section, oneFormItemCopybook = "YES"
EZUAUTH	EZUAUTH
EZUINST	EZUINST
PSBLIB	PSBLIB — DL/I only
PROCLIB	PROCLIB — VSE only
PWRCLASS	PWRCLASS — VSE only
RESLIB	RESLIB — DL/I and IMS only
SQLDBNAM	SQLDBNAM — VSE only
SQLPKGNM	SQLPKGNM — VSE only
SQLPROPT	SQLPROPT — VSE only
SQLSTMDE	SQLSTMDE — VSE only
SQLSTOPT	SQLSTOPT — VSE only
SQLUSRPW	SQLUSRPW — VSE only
VMFMODE	Not applicable — VM only
VMDISKADDR	Not applicable — VM only
VUSERLIB	VUSERLIB — VSE only

Other generation information

This section is organized in the following tables:

- MVS preparation templates and procedures, Table 156 on page 387
- MVS runtime templates, Table 157 on page 388
- MVS file and database allocation templates, Table 158 on page 389
- MVS file and database allocation placeholder templates, Table 159 on page 389
- OS/400 runtime templates, Table 160 on page 390

Note: For VSE information, refer to the *Rational Business Developer V7.5 Generation for z/VSE feature Reference Manual* (SC19-2539-00)

Preparation templates and procedures

Table 156 on page 387 shows the VAGen preparation templates and procedures used for preparing the outputs of COBOL generation for the MVS runtime environments. The table only includes the MVS runtime environments that have a corresponding EGL z/OS runtime environment. The template name is shown first, followed by the procedure name. For most of the procedure names, the first three characters are ELA and the remaining characters indicate the steps included in the procedure in the following way: P (DB2 precompile), T (CICS translate), C (COBOL compile), L (link edit), and B (DB2 bind).

Note:

- For the MVS runtime environments, VisualAge Generator also uses bind control templates. See “Establishing a bind control part to use as a template” on page 212 for details of how to convert the bind control templates.
- The Generation for VSE feature continues to use preparation templates and procedures similar to VisualAge Generator. However, the template and procedure names have changed. For details, refer to the *Rational Business Developer V7.5 Generation for z/VSE feature Reference Manual* (SC19-2539-00).
- OS/400 is not included in the following table because the EGL build script FDAPREP replaces all of the VAGen OS/400 preparation templates. Comments in the FDAPREP build script indicate which VAGen preparation template formed the basis for that section of the build script.

Table 156. MVS preparation templates and procedures

Environment	Part type and database	VisualAge Generator template and procedure	EGL build script
MVS CICS	Program - without DB2	EFK2MPCB ELATCL	FDATCL
	Program - with DB2	EFK2MPCA ELAPTCLB	FDAPTCL followed by FDABIND
	Map Group - print services	EFK2MMCA ELACL	FDACL
	Map Group - format module	EFK2MMTF ELAL	FDALINK
MVS Batch	Program - without DB2	EFK2MPBA ELACL	FDABCL
	Program - with both DL/I and DB2	EFK2MPBB ELAPCLB	FDAPCL followed by FDABIND
	Program - with DB2 only	EFK2MPBC ELAPCLB	FDAPCL followed by FDABIND
	Map Group - print services	EFK2MMCA ELACL	FDACL
IMS/VS	Program - with DL/I only	EFK2MPIC ELACL	FDABCL
	Program - with DL/I and a DB2 work database	EFK2MPID ELACLB	FDABCL followed by FDABIND
	Program - with both DL/I and DB2	EFK2MPIE ELAPCLB	FDAPCL followed by FDABIND
	Map Group - print services for MFS	EFK2MMCB ELACL	FDACL
	Map Group - MFS, with /mfstest generation option	EFK2MMST MFSTEST	FDAMFS
	Map Group - MFS, with /nomfstest generation option	EFK2MMSU MFSUTL	FDAMFS

Table 156. MVS preparation templates and procedures (continued)

Environment	Part type and database	VisualAge Generator template and procedure	EGL build script
	Map Group - format module	EFK2MMTF ELAL	FDALINK
IMS BMP	Program - with DL/I only	EFK2MPIA ELACL	FDABCL
	Program - with both DL/I and DB2	EFK2MPIB ELAPCLB	FDAPCL followed by FDABIND
	Map Group - print services for SEQ and GSAM	EFK2MMCA ELACL	FDACL
	Map Group - print services for MFS	EFK2MMCB ELACL	FDACL
	Map Group - MFS, with /mfstest generation option	EFK2MMST MFSTEST	FDAMFS
	Map Group - MFS, with /nomfstest generation option	EFK2MMSU MFSUTL	FDAMFS
All MVS environments	Relink program	EFK2MPRE ELARLINK	FDALINK
	Table	EFK2MMCA ELACL	FDACL

Runtime templates

Table 157 shows the VAGen runtime templates that are used to generate the basic runtime JCL for the MVS Batch and IMS BMP environments. Table 158 shows the file and database allocation templates that are used to create DD statements within the generated runtime JCL. Table 159 shows the file and database allocation placeholder templates that are used to indicate that additional DD statements might be required for another program that is called or transferred to with an XFER or DXFR statement or for a program that uses EZEDEST or EZEDESTP. Table 160 shows the VAGen runtime templates that are used to generate the control language (CL) for the OS/400 environment. All 4 tables include the corresponding VAGen and EGL information.

Note: VSE Batch is not included in the tables because the Generation for VSE feature uses similar templates to VisualAge Generator. However, the template names have changed. For details, refer to the *Rational Business Developer V7.5 Generation for z/VSE feature Reference Manual* (SC19-2539-00).

Table 157. MVS runtime templates

Environment	Program type and database	VisualAge Generator runtime JCL template	EGL runtime JCL template
MVS Batch	Called program	EFK2MEBA	fda2meba.tpl
	Main program - No databases	EFK2MEBE	fda2mebe.tpl

Table 157. MVS runtime templates (continued)

Environment	Program type and database	VisualAge Generator runtime JCL template	EGL runtime JCL template
	Main program - DL/I only	EFK2MEBC	fda2mebc.tpl
	Main program - DB2 only	EFK2MEBD	fda2mebd.tpl
	Main program - DL/I and DB2	EKF2MEBB	fda2mebb.tpl
IMS BMP	Called program	EFK2MEBA	fda2meba.tpl
	Main program - DL/I only	EFK2MEIB	fda2meib.tpl
	Main program - DL/I and DB2	EFK2MEIA	fda2meia.tpl

Table 158. MVS file and database allocation templates

Environment	File or database type	VisualAge Generator runtime JCL template	EGL runtime JCL template
MVS Batch and IMS BMP	DL/I database in MVS Batch	EFK2MDLI	fda2mdli.tpl
	VSAM or VSAMRS input for serial, indexed, or relative files	EFK2MVSJ	fda2mvsi.tpl
	VSAM or VSAMRS output for serial, indexed, or relative files	EFK2MVSO	fda2mvso.tpl
	SEQ or SEQRS input for serial files	EFK2MSDI	fda2msdi.tpl
	SEQ or SEQRS output for serial files	EFK2MSDO	fda2msdo.tpl
	GSAM input for serial files	EFK2MGSI	fda2mgsi.tpl
	GSAM output for serial files	EFK2MGSO	fda2mgso.tpl
	GSAM file in an IMS BMP	EFK2MIMS	fda2mims.tpl

Table 159. MVS file and database allocation placeholder templates

Environment	File and database allocation placeholder type	VisualAge Generator runtime JCL template	EGL runtime JCL template
MVS Batch and IMS BMP	XFER or DXFR to EZEAPP	EFK2MEZA	fda2meza.tpl

Table 159. MVS file and database allocation placeholder templates (continued)

Environment	File and database allocation placeholder type	VisualAge Generator runtime JCL template	EGL runtime JCL template
	CALL, XFER or DXFR to a specific application or RT generation option transfers to a specific application	EFK2MCAL	fda2mcal.tpl
	Application uses EZEDEST or EZEDESTP	EFK2MEZD	fda2mezd.tpl

Table 160. OS/400 runtime templates

Environment	Purpose of template	VisualAge Generator runtime template	EGL runtime JCL template
OS/400	Provides the CL to add libraries to the client/server job and to start commitment control if this is the first server program called by a client	EFK24EBC	fda24ebc.tpl
	Provides epilogue to the runtime CL to handle errors that occur	EFK24EEC	fda24eec.tpl

Other runtime information

This section is organized in the following tables:

- Runtime environment variables, Table 161 on page 390
- vgj.properties, Table 162 on page 392

Runtime environment variables

The VAGen runtime environment variables are used for generated COBOL for CICS OS/2 and for generated C++ for the workstation environments. Most VAGen runtime environment variables do not have a corresponding EGL runtime property. Even when there is a correspondence, the values for the EGL runtime properties have different values or slightly different meanings. In addition, there are numerous new EGL runtime properties. Therefore, use the material in this section as an aid in creating your EGL runtime properties, but be sure to carefully review *all* the EGL runtime properties in the online help to determine if there are additional properties you need to set.

Table 161. Runtime environment variables

VAGen runtime environment variables	EGL runtime properties
BTRINTF	Not used -- CICS OS/2 only
CICSCOBCOPY	Not used -- CICS OS/2 only
CICSRGRP	Not used -- CICS OS/2 only

Table 161. Runtime environment variables (continued)

VAGen runtime environment variables	EGL runtime properties
CICSCOBOL	Not used -- CICS OS/2 only
CICSRD	Not used -- CICS OS/2 only
CICSWRK	Not used -- CICS OS/2 only
COBPATH	Not used -- CICS OS/2 only
CSODIR	Not used
CSO_DUMP_CONV	Not used
CSO_DUMP_DATA	Not used
CSOTIMEOUT (time in seconds)	cso.cicsj2c.timeout (time in milliseconds)
CSOTROPT	tcpiplistener.trace.flag or vgj.trace.type, depending on what you need to trace
CSOTROUT	tcpiplistener.trace.file or vgj.trace.device.spec, depending on what you need to trace
DB2INSTANCE	Not used
DLITROPT	Not supported -- remote DL/I only
DLITROUT	Not supported -- remote DL/I only
DPATH	Not used
ELAPATH	Not used
ELARTRDB_ddd where ddd is the CICS transaction code	Not used -- CICS OS/2 only
EZERGRGL_ddd where ddd is the NLS language code	vgj.datemask.gregorian.long.locale
EZERGRGS_ddd where ddd is the NLS language code	vgj.datemask.gregorian.short.locale
EZERJULL_ddd where ddd is the NLS language code	vgj.datemask.julian.long.locale
EZERJULS_ddd where ddd is the NLS language code	vgj.datemask.julian.short.locale
EZERNLS	vgj.nls.code
EZERSQLDATE	Not supported -- EGL uses JDBC
EZERSQLDB	vgj.jdbc.database.SN, where SN is the server name; format of the value differs from EZERSQLDB
EZERSQLM1	Not supported -- EGL uses JDBC
EZERSQLM2	Not used -- EGL uses JDBC
EZERSQLMF	Not used -- EGL uses JDBC
EZERSQLUS	Not supported -- EGL uses JDBC
FCEOPT	Not used -- C++ generation only
FCETROPT	Not used -- C++ generation only
FCWCOMP	Not used -- C++ generation only
FCWDB2DIR	Not used -- EGL uses JDBC

Table 161. Runtime environment variables (continued)

VAGen runtime environment variables	EGL runtime properties
FCWDBNAME_ <i>programName</i>	vgj.jdbc.default.database. <i>programName</i> ; format of the value differs from FCWDBNAME
FCWDBNOOP	Not used -- distributed CICS only
FCWDBPASSWORD	vgj.jdbc.default.password
FCWDBUSER	vgj.jdbc.default.userid
FCWDBVERSION (Oracle version)	Not used -- EGL uses JDBC
FCWDPATH (directory for tables and resource association)	Not used. Tables must be in the classpath. Resource association becomes vgj.ra.* properties
FCWFIODB	Not used -- distributed CICS only
FCWLIBPATH	Not used -- C++ generation only
FCWMAKE	Not used -- C++ generation only
FCWRSC (raf file name)	Not used. Resource association becomes vgj.ra.* properties
FCWOPT (map field with date mask)	Not supported
FCWTRDB_ <i>tttt</i> where <i>tttt</i> is the CICS transaction code	Not used -- distributed CICS only
FCWTROPT	vgj.trace.type; values and meanings differ from FCWTROPT
FCWTROUT	vgj.trace.device.spec; must also specify vg.trace.device.option=2
INFORMIXDIR	Not supported -- ODBC only
MDLROOT	Not supported -- VAGen Templates only
ORACLE_HOME	Not used -- EGL uses JDBC
RMTDLI_PARTNER_LU	Not supported -- remote DL/I on MVS only
RMTDLI_PARTNER_TP	Not supported -- remote DL/I on MVS only
RMTDLI_SERVER_ENV	Not supported -- remote DL/I on MVS only
VSEDLI_CFG	Not supported -- remote DL/I on VSE only
VSEDLI_TRACE	Not supported -- remote DL/I on VSE only

vgj.properties

The VAGen vgj.properties file is used for Java generation for the workstation environments. Most VAGen vgj.properties variables have a corresponding EGL runtime property. However, in some cases, the values for the EGL runtime properties (also called settings) have different values or slightly different meanings. In addition, there are numerous new EGL runtime properties. Therefore, use the material in this section as an aid in creating your EGL runtime properties, but be sure to carefully review *all* the EGL runtime properties in the online help to determine if there are additional properties you need to set.

Table 162. vgj.properties

VAGen vgj.properties	EGL runtime properties
cso.application.xxx where xxx is the server group	Not used -- EGL uses a linkage properties file

Table 162. *vgj.properties* (continued)

VAGen <i>vgj.properties</i>	EGL runtime properties
cso.linkagetable.xxx where xxx is the linkage table name	cso.linkageOptions.LO, where LO is the linkage options part name and the corresponding linkage properties file is named LO.properties
cso.serverLinkage.xxx.yyy where xxx is the server group and yyy is the attribute name.	Not used -- EGL uses a linkage properties file
vgj.datemask.gregorian.long.xxx where xxx is the NLS language code	vgj.datemask.gregorian.long.locale
vgj.datemask.julian.long.xxx where xxx is the NLS language code	vgj.datemask.julian.long.locale
vgj.java.command	vgj.java.command
vgj.jdbc.database.SN where SN is the server name	vgj.jdbc.database.SN where SN is the server name; format of the value differs from EZERSQLDB
vgj.jdbc.default.database	vgj.jdbc.default.database or vgj.jdbc.default.database.programName
vgj.jdbc.default.database.user.id	vgj.jdbc.default.userid
vgj.jdbc.default.database.user.password	vgj.jdbc.default.password
vgj.jdbc.drivers	vgj.jdbc.drivers
vgj.nls.code	vgj.nls.code
vgj.nls.number.decimal	vgj.nls.number.decimal
vgj.powerserver.location	Not used
vgj.ra.FN.contable where FN is the logical file name	vgj.ra.QN.conversionTable, where QN is the MQ Series message queue name; not all of the VAGen values are valid
vgj.ra.FN.filetype	vgj.ra.FN.fileType
vgj.ra.FN.replace	vgj.ra.FN.replace
vgj.ra.FN.sysname	vgj.ra.FN.sysname
vgj.ra.FN.text	vgj.ra.FN.text
vgj.trace.device.option	vgj.trace.device.option
vgj.trace.device.spec	vgj.trace.device.spec
vgj.trace.type	vgj.trace.type; values and meanings differ from FCWTROPT

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 in EGL

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 the Table and Additional Records lists for any programs, logic statements, data item edit routines, map edit routines, and UI edit tables. 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 DataTable in EGL, 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 the main map group or help map group in any program. 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 FormGroup in EGL, 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 and references in linkage table parts. Also change the names of any bind control or link edit parts that correspond to this program. Be sure to change any non-VAGen references to the program name, including CICS program and transaction definitions. Alternatively, you can wait until you have migrated, rename the program in EGL, 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 and references in linkage table parts. Also change the names of any bind control or link edit parts that correspond to this program. Be sure to change any non-VAGen references to the program name, including CICS program and transaction definitions. Alternatively, you can wait until you have migrated, rename the program in EGL, and use the EGL **alias** property to specify the original program name. Refer to Appendix Appendix A, “Reserved words,” on page 249 that lists the EGL reserved words.

HPT.EGL.0005.w UI Record *recordName* is a reserved word or starts with the # or @ symbol. It must be renamed.

Explanation: The Stage 1 migration tool does not rename UI records for you. However, the Stage 2 tool renames the record using your Stage 2 Renaming prefix. The Stage 2 tool also includes the **alias** property for the VGUI record so that the names in the EGL

generated outputs are identical to those in VisualAge Generator. The Stage 3 tool also renames the file that contains the VGUI record. If you migrate in single file mode, the migration tool makes the same changes.

User response: None. Allow the Stage 2 migration tool to rename the record for you. This also changes all references to the record and the file name.

HPT.EGL.0006.i Migration of *preferenceFile* will produce *outputList*.

Explanation: Migration of *preferenceFile* produces *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. Also check for other error messages that might provide more details about the error.

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 EGL 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 128 for Java or “Mapping page” on page 151 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*. **Modify the renaming rules.**

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*. **Modify the renaming rules.**

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 (.). Modify the renaming rules.**

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 (.). Modify the renaming rules.**

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. For VAGen on Java, the message is repeated to provide information about the VAGen Utilities feature. For both

VAGen on Java and VAGen on Smalltalk, the message is repeated to provide the version of the reserved word list that is loaded

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.

HPT.EGL.0021.e The Externalized EGL Reserved Word List cannot be loaded. Verify *fileName*.

Explanation: The specified *fileName* cannot be found. The *fileName* is the full path name of the file. The file should have the correct name and location if you installed the Stage 1 migration tool as described in Chapter 4, "Stage 1 — Extracting from Java," on page 123 or Chapter 5, "Stage 1 — Extracting from Smalltalk," on page 147.

User response: Verify that the EGL Reserved Word file is located in the correct path and has the correct file name. If not, check your installation steps for the Stage 1 migration tool.

HPT.EGL.0022.e Part *partName* has invalid External Source Format.

Explanation: The Stage 1 migration tool has made 3 attempts to extract the External Source Format from the repository. However, the opening tag and ending tag for the part are not a valid pair (for example, :record and :erecord). The Stage 1 migration tool continues processing. However, the migration database does not contain correct External Source Format for the specified *partName*.

User response: Try looking at the External Source Format in the repository to see if there are any obvious

errors in the part. If you are using a remote repository, try copying it to a local drive and running the Stage 1 migration tool again. If you are unable to resolve the problem, contact IBM support.

HPT.EGL.0023.e Part *partName* has invalid External Source Format due to duplicate parts.

Explanation: One or more parts of the same part type have the same part name. The migration tool cannot determine which External Source Format to store in the migration database.

User response: Modify your migration set so that there are no duplicate part names in the migration set.

HPT.EGL.0024.e Derived map group name *mapGroupName* is the same as the name of another part.

Explanation: VisualAge Generator only requires a map group part if there is a floating area specification. EGL always requires a form group part. The Stage 1 migration tool attempted to create a map group part using the map group portion of the map names. However, the map group name is the same as one of the other parts in the migration set. The Stage 1 migration tool ends the processing without modifying the migration database.

User response: Change the names of the maps so that the map group portion of the name does not conflict with any of the other parts in the migration set. Also change any programs that use the map group to specify the new map group name. Run Stage 1 again.

HPT.EGL.0025.e Migration database does not have all the required tables. Rerun *setupdatabase.bat* and *setuptables.bat*.

Explanation: The migration database changed for EGL Version 7.1 and now requires some additional tables. In addition, the DB2 prerequisites have changed.

User response: Ensure that DB2 is installed at the proper level. Then run the *setupdatabase.bat* and *setuptables.bat* files to define the migration database and its tables. For details, see "Creating the migration database" on page 459.

HPT.EGL.0026.i Migration database server version is *versionNumber*.

Explanation: This informational message provides the version information for DB2.

User response: None.

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 symbol # or @ 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 A, "Reserved words," on page 249 that lists the EGL reserved words. Be sure that the resulting EGL package name does not start with the # or @ 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*.

Explanation: The database driver that is specified in the Stage 1 preferences file could not be found.

User response: One of the following situations might have occurred:

- The database information (database driver, database name, schema, userid, or password) that is specified in the Stage 1 preferences file might be incorrect. For more information on how to set these values, see "Execution page" on page 130.
- The classpath that is specified in the **Properties** for the **VAGenToEGLMigration** class does not include the db2java.zip file. For more information on how to set the **Properties**, see "Running the Stage 1 tool" on page 138.
- The PATH or CLASSPATH environment variable is too long. Try moving the directories related to SQL

to the beginning of the PATH or CLASSPATH environment variable. Alternatively, try copying the db2jdbc.dll from your *db2InstallationDirectory*\SQLLIB\BIN to your *vajavaInstallationDirectory*\ide\program directory. The default *db2InstallationDirectory* is c:\Program Files\IBM\SQLLIB. The default *vajavaInstallationDirectory* is c:\Program Files\IBM\VisualAge for Java4.0.

- The version of DB2 is not supported for Stage 1 on Java. For example, you might be trying to use DB2 Version 9.x. See Appendix G, "Migration database," on page 459 for the versions of DB2 that are supported for Stage 1 on Java.

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: Check the following things:

- Make sure that the specified database has been created.
- Review your user ID and password settings in the Stage 1 preferences file to ensure that they are correct.
- Ensure that you updated the Properties for the **VAGenToEGLMigration** class to include the location of db2java.zip on the **Class Path** page and that you clicked **Compute Now** on the **Class Path** page to complete the update.
- Check your system environment variables:
 - The classpath environment variable must include:
sqlInstallationDirectory\SQLLIB\java\db2java.zip
 - The path environment variable must include:
sqlInstallationDirectory\SQLLIB\BIN
sqlInstallationDirectory\SQLLIB\FUNCTION

This problem can occur when the classpath or path statement is too long, resulting in some of the subdirectories being omitted. If you used the default installation directory "Program Files", try to shorten the path environment variable by using one of the following techniques:

- Change "Program Files" (without the quotes) to "Progra~1" (without the quotes).
- Reinstall DB2 using shorter product directory names without blanks (for example, SQLLIB instead of the default "Program Files\SQLLIB").

Alternatively, try one of the following techniques:

- Try moving the SQL-related information to the beginning of the classpath and path environment variables.

- Try copying the db2jdbc.dll file from the \SQLLIB directory to a directory that is in your path environment variable (for example, to `%javaInstallDirectory%\ide\program`).

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(optionalErrorCode)`.

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. Also try the following techniques based on the *optionalErrorCode*:

- *optionalErrorCode* is **ToolEnv**. Expand the **IBM VisualAge Generator EGL Migration** project and then expand the **com.ibm.vgj.mig** package. Right-click the **VAGenToEGLMigration** class and click **Properties**. Click the **Class Path** tab. Next to the **Extra directories path**, click **Edit**. Select the relative path entry for:

```
..\IBM IDE Utility local implementation\
```

Then click **Add Directory** and change the entry to use an explicit path by using the following format:

```
drive:\VAJavaInstallDirectory\ide\
project_resources\
IBM IDE Utility local implementation
```

where *drive* is the drive and *VAJavaInstallDirectory* is the directory where VisualAge for Java is installed. Click **OK** twice to return to the **Class Path** page. Click **Compute Now** to update the **Complete class path** information. Then click **OK**.

- No *optionalErrorCode*. There should be additional messages in the log files or Console window. Use these messages to resolve the problem.

After resolving the problem, 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 two 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 1 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. However, before you increase the length of the SQL column, consider whether you 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.

- If `javaExceptionStackTrace` contains either of the following SQL messages, check that the DB2 user ID that is being used to run migration has sufficient authority for the migration database:
 - SQL2311N The user does not have the authority to run the Run Statistics utility on table "migrationTable"
 - SQL0551N "userid" does not have the privilege to perform operation "sqlOperation" on object "migrationTable"

This problem can occur if the user ID that created the migration database is different from the user ID that is running the Stage 1 migration tool. See "DB2 authority requirements" on page 459 for details about the authority and privileges that are required.

- If `javaExceptionStackTrace` contains the following SQL message, check your system environment variables as described for message HPT.EGL.0104.e:
 - SQL0444N Routine "routine-name" (specific name "specific-name") is implemented with code in library or path "library-or-path", function "function-code-id" which cannot be accessed. Reason code: "code".
- If the message indicates that there is an SQL problem with ADMIN_CMD or RUNSTATS, this indicates that the migration database was created before you installed the required level of DB2. See Appendix G, "Migration database," on page 459 for the correct level of DB2. Install the correct level and then run setupdatabase.bat and setuptables.bat again.
- If `javaExceptionStackTrace` indicates that there was a `java.lang.NullPointerException` the error varies depending on which tool you are running:
 - If you are running either the Preferences Tool or the Migration Tool, locate the following directory:


```
vajavaInstallDirectory\ide\features\
com-ibm-vgj-mig\projects.dat.pr\
IBM VisualAge Generator EGL Migration\
version\data
```

The directory must contain the following files:

- MigPreferences.dtd
- MigPreferences.xml
- tableNodes.xml
- VGMigrationDef.dtd

Files might be missing if you installed VisualAge Java and VisualAge Generator and used long directory names or a directory name that contains spaces. Try unzipping the migration tool to another directory and copying the missing files to the product directory. Alternatively, try reinstalling both VisualAge Java and VisualAge Generator using a short directory name (for example, vag).

- If you are running the Migration Tool, determine whether the details of the stack trace mention the following files:

- com.ibm.vgj.mig.XMLFileParser.readXML
- com.ibm.vgj.mig.ConfigPlanEntity
- com.ibm.vgj.mig.VAGenToEGLMigration.loadConfigPlans

If so, check that you have a valid migration plan file (.pln file). Also check whether you are using the -o option. This error can occur if you omit the -o option and the existing .pln file is empty or invalid. To correct the problem, either create a valid .pln file by hand or specify the -o option so that the migration tool creates a new (valid) .pln file for you. For more information about the -o option, see "Running the Stage 1 tool" on page 138. For more information about how to create the .pln file by hand, see "Creating a migration plan file manually" on page 141.

- Check the information that follows this message for additional details. Also check the Console for additional messages such as HPT.EGL.0110.e and follow the instructions for that message.

If you are not able to resolve the problem, contact IBM support for assistance.

HPT.EGL.0110.e Project *projectName* version *versionName* is not defined in the repository.

Explanation: The Stage 1 migration tool expanded the high-level PLP for the migration set, including the chain of PLP projects. The specified project version is referenced in the PLP chain, but is not available in the repository or is an empty version.

User response: Determine whether the project and version should be included in the migration set. If so, check to see whether the requested version of the project has been purged from the repository. If so, restore the project version and then migrate again. If you are not able to resolve the problem, contact IBM support for assistance.

HPT.EGL.0111.e Original VAGen project name *projectName* - results in a derived empty EGL project name. Modify the renaming rules.

Explanation: Using the renaming rules that you specified, the migration tool has created a proposed EGL project name that does not contain any characters.

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**.

HPT.EGL.0112.e Original VAGen package name *packageName* - results in a derived empty EGL package name. Modify the renaming rules.

Explanation: Using the renaming rules that you specified, the migration tool has created a proposed EGL package name that does not contain any characters.

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.0113.e Migration Set *migrationSetName* - *migrationSetVersion* contains number duplicate parts. Duplicates are not permitted.

Explanation: The Stage 1 migration tool requires unique part names so that it can associate the correct source code with the part edition. The specified migration set contains duplicate part names. *Number* specifies the number of pairs of duplicate part names. Message HPT.EGL.0115.e provides the part names and the package names in which the parts occur. There is one message for each pair of part names.

User response: The project versions from the migration set are still in the workspace. From the VAGen Parts Browser, click **Tools -> Show Duplicate Parts** to determine which parts have duplicate names. Correct the problem, version the projects, update your migration set definition, and then run the Stage 1 tool again.

HPT.EGL.0114.e Package *packageName* version *packageVersionName* in project *projectName* version *projectVersionName* is not defined in the repository.

Explanation: The Stage 1 migration tool expanded the high-level PLP for the migration set, including the chain of PLP projects. The specified package version is referenced by the specified project version in the PLP chain, but is not available in the repository. The *packageVersionName* might be in the format: Missing - (*versionDateTimeStamp*).

User response: Determine whether the package version should be included in the project. If so, check to see whether the requested version of the package has been purged from the repository. If so, restore the package version and then migrate again. If you are not able to resolve the problem, contact IBM support for assistance.

HPT.EGL.0115.e Duplicate part *partName* was found in *packageName1* with type *partType1* and in *packageName2* with type *partType2*.

Explanation: The Stage 1 migration tool requires unique part names so that it can associate the correct source code with the part edition. The specified part name occurs in one or more packages, possibly with different part types. Message HPT.EGL.0115.e is repeated for each pair of part names.

User response: The project versions from the migration set are still in the workspace. From the VAGen Parts Browser, click **Tools -> Show Duplicate Parts** to determine which parts have duplicate names. Correct the problem, version the projects, update your migration set definition, and then run the Stage 1 tool again.

HPT.EGL.0116.w Project *projectName* matches a project name filter. The project does not have a version that matches any version name filter.

Explanation: The specified PLP project was found in the repository, but does not have the specified version.

User response: Correct the version name in the repository filters of the migration preferences file. The version name is case sensitive.

HPT.EGL.0118.w Unable to determine VAGen NLS setting. Default encoding was used for output files.

Explanation: The migration tool was not able to determine the locale to use to set the encoding for the output files from Stage 1. One of the VAGen product projects might be missing from the class path. The Stage 1 tool continues processing, but you might have difficulties using the output files.

User response: Use one of the following techniques to correct the problem:

- Set the class path information using the following steps:
 1. In the Workbench window, click the **Projects** tab.
 2. Navigate to the **IBM VisualAge Generator EGL Migration** project.
 3. Expand the migration project and then expand the **com.ibm.vgj.mig** package.
 4. Within the package, right-click the **VAGenToEGLMigration** class and follow these steps:
 - a. Click **Properties** on the pop-up menu.
 - b. Click the **Class Path** tab.
 - c. Next to the **Project path** section, click **Edit**.
 - d. In the Class Path window, ensure that **IBM VisualAge Generator Runtime** is selected.
 - e. Click OK to close the Class Path window.

- f. Click OK again to close the Properties window.
5. Repeat step 4 for the **PreferencesUI** class.
6. Run Stage 1 migration again.
- Follow the steps described in “Specifying your character set information” on page 136 and then run the Stage 1 migration again.

To avoid running Stage 1 migration again, make the following changes:

- To view the miglog.xml and migdebug.xml files, change the value for the **encoding** property in the following line of the files to indicate your required encoding:

```
<?xml version="1.0" encoding="iso-8859-1"?>
```

To view the report files, change the value for the **charset** property in the following line of the .html files to indicate your required encoding:

```
<META http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
```

Alternatively, given the potentially large number of .html report files, run Stage 1 migration again, specifying that just the report is to be created by specifying the following lines in your MigPreferences.xml file:

```
<verification>
  <generateReport>true</generateReport>
  <reportName>c:\tempMigJ\JTurbo\report\
    MigrationReport.htm</reportName>
</verification>
<dbUpdate>false</dbUpdate>
```

If you use the Preferences GUI to modify the MigPreferences.xml file, select **Generate report** and clear **Update database** on the Execution page of the GUI.

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 @ 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 A, “Reserved words,” on page 249 that lists the EGL reserved words. Be sure that the resulting EGL package name does not start with the # or @ 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, incorrect level of DB2, or missing SQL tables.

User response: Review the *errorMessage*. Depending on the error, different actions might be required. For example:

- If *errorMessage* includes information about SQL error code SQL0440N, this indicates that the wrong level of DB2 is installed. See Appendix G, “Migration database,” on page 459 for the correct level of DB2. Install the correct level and then run setupdatabase.bat and setuptables.bat again.
- If *errorMessage* contains either of the following SQL messages, check that the DB2 user ID that is being used to run migration has sufficient authority for the migration database:
 - SQL2311N The user does not have the authority to run the Run Statistics utility on the table
 - SQL0551N "userid" does not have the privilege to perform operation "sqlOperation" on object "migrationTable"

This problem can occur if the user ID that created the migration database is different from the user ID that is running the Stage 1 migration tool. See “DB2 authority requirements” on page 459 for details about the authority and privileges that are required.

- Check the information in the migration preferences file.

If you are not able to resolve the problem, 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*, version *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 two 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 your 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 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.

HPT.EGL.0211.e Original VAGen configuration map name *configurationMapName* - results in a derived empty EGL project name. Modify the renaming rules.

Explanation: Using the renaming rules that you specified, the migration tool has created a proposed EGL project name that does not contain any characters.

User response: Edit the Stage 1 migration preferences file and modify your configuration map 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.0212.e Original VAGen application name *applicationName* - results in a derived empty EGL package name. Modify the renaming rules.

Explanation: Using the renaming rules that you specified, the migration tool has created a proposed

EGL package name that does not contain any characters.

User response: Edit the Stage 1 migration preferences file and modify your application 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.0213.i The message table utility for migration set *migrationSetName - versionName added count new associate rows*

Explanation: This information message indicates the number of message tables that were added as associates of programs based on the message table prefix in the program parts. The message tables are added as associates in Stage 1 so that the Stage 3 migration tool can set the EGL Build Path correctly and also include the correct **import** statements for the program parts. This message is also issued if you select the option to **Repair Message Table Associates** when running the Stage 1 migration tool.

User response: None.

Messages from the VisualAge Generator to EGL migration tool— Stage 2

The following messages are produced by 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 quotation marks, such as using a quotation mark in the 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.0004.e Output file *fileName1* **and UI record have the same name - UI record placed in file** *fileName2*.

Explanation: During single file migration, *fileName1* is the specified output file. The external source format file contains a UI record that has the same name as the specified output file. The file also contains some other parts. The migration tool placed the other parts in the specified output file before it processed the UI record. The tool then placed the UI record into an EGL file named *fileName2*. EGL validation displays an error message in the Problems view.

User response: Give *fileName1* a different name. Give *fileName2* the same name as that of the VGUI record.

IWN.MIG.0047.i Migration set *Name_version* **— 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_version* **- 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* has one of the following values: Program, Map Group, or Table. The associates for the specified *partName* are 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 quotation marks, such as using a quotation mark in the 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) or External Source 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 EGL validation displays an error message 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. If you are unable to resolve the problem, contact IBM support for assistance. Be prepared to provide a small repository (.dat file) or External Source File containing the parts that have problems.

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 459 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 connecting to a remote database, be sure that you have cataloged the database locally.

IWN.MIG.0063.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 EGL development environment before attempting to backup your database.

IWN.MIG.0070.e The user exit method *renameUserExitName(partName)* does not exist.

Explanation: The JAR file that you specified for your **Rename user exit** or the package and class within the JAR file could not be found.

User response: Check the **JAR file location**, **Package name**, and **Class name** that you specified for your **Rename user exit** in the VAGen Migration Preferences.

IWN.MIG.0071.e The user exit method *renameUserExitName(partName)* does not have the required method signature.

Explanation: The **Rename user exit** requires that you include a method with the signature *renameUserExit(String s, Connection c)*. This method did not exist in the JAR file, package, and class that you

specified for your **Rename user exit** in the VAGen Migration Preferences.

User response: Review your class definition and ensure that you included the required method signature. Also ensure that you specified the correct the **JAR file location**, **Package name**, and **Class name** for your **Rename user exit** in the VAGen Migration Preferences.

IWN.MIG.0072.e The user exit method *renameUserExitName(partName)* enforces Java language access control and the underlying method is inaccessible.

Explanation: The migration tool does not have access to the definition of the specified user exit class.

User response: Verify that **Rename user exit** class is defined as public and it is in the specified package.

IWN.MIG.0073.e The user exit method *renameUserExitName(partName)* abruptly terminated by throwing an exception.

Explanation: The method *renameUserExit(String s, Connection c)* returned a null value. Migration continues. The migration tool uses the original VAGen part name.

User response: Place a **try** block around the code in your **Rename user exit**. If there is an exception, return the original VAGen part name to avoid getting this message.

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
\org.eclipse.core.runtime\settings
\com.ibm.etools.egl.vagenmigration.prefs

User response: See “VAGen Migration preferences” on page 174 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* refers to one of the following parameters:

- **-importFile**
- **-eglFile**
- **-package**

The parameter names are case sensitive.

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* refers to one of the following parameters:

- **-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.0095.e Function *functionName* - **EZESCRPT is not supported for migration.**

Explanation: The specified function contains statements that use the EZESCRPT special function word. There is no replacement for this EZE function in EGL. The migration tool converts EZESCRPT to EZE_SCRPT to preserve the logic of your function.

However, you cannot use this function in EGL.

User response: Review the EGL function. You cannot generate or run programs that use this function.

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 **validatorFunction** and a **validatorDataTable** property for a DataItem part. The migration tool converts the map edit routine in the following way:

- EZEC10 and EZEC11 migrate to the **validatorFunction** property.
- If the part specified by *editRoutineName* is available during migration and is a function, the *editRoutineName* migrates to the **validatorFunction** property. The migration tool also migrates the edit routine to the **validatorFunction** 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 **validatorDataTable** property. The migration tool also migrates the edit routine to the **validatorDataTable** 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 **validatorFunction** 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 EGL DataItem definition and change the **validatorFunction** property to the **validatorDataTable** property. For additional considerations, see the information on edit routines in “Map edit routine for shared data items” on page 68.

IWN.MIG.0102.e 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, called parameter list, 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 is valid.

User response: If you want to use the type definition, you might need to add an **import** statement to import the package in which the DataItem part resides. 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.0103.w Data item *DataItemName* - preferences caused **evensql=y** to be ignored.

Explanation: The specified data item part is a VAGen PACK (EGL DECIMAL) item with **evensql=y**. Your VAGen Migration Preferences specified that **evensql=y** is not to be honored. Based on your preferences, the migration tool converted the PACK item to the next higher odd precision, with a maximum length of 18. The migration tool determines the EGL precision for PACK items in the following way:

- If **evensql=n** is specified for an item, the EGL precision is always calculated as $(\text{VAGen bytes} * 2) - 1$ with a maximum value of 18.
- If **evensql=y** is specified for an item, the EGL precision is calculated based on the preference:
 - If the **Do not honor evensql=y for items or variables** preference is selected, the EGL precision is calculated as:
$$(\text{VAGenBytes} * 2) - 1$$

with a maximum value of 18. Message IWN.MIG.0103.w is only issued in this situation.

- If the preference is not selected (honor **evensql=y**), the EGL precision is calculated as
$$(\text{VAGenBytes} * 2) - 2$$

with a maximum value of 18. Message IWN.MIG.0103.w is not issued.

User response: None. However, you might want to review the use of this item in any SQL WHERE clauses or EGL **prepare** statement. There might be an impact on performance if the definition of this item does not exactly match the SQL table definition. For details, see information about EVENSQL in “Eliminating the use of VisualAge Generator compatibility mode” on page 225 and “PACK data items with even length” on page 65.

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 two 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 71.

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 record parts. 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 70.

IWN.MIG.0203.w 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. EGL tolerates record parts that do not contain any fields. 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** keyword. The table names must be specified in each SQL record part 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 **tableNameVariables** properties from the VAGen alternate specification record (*altspecRecord*). 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 in “Alternate specification records” on page 72.

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** keyword. All key items must be specified in each SQL record part 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 information on “Alternate specification records” on page 72.

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 keyword 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** keyword 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 part 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 72.

IWN.MIG.0209.e Record *recordName* - alternate specification record *altspecRecord* has no items; embed keyword 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** keyword 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.0211.w Record *recordName*, data item *DataItemName* - preferences caused evensql=y to be ignored.

Explanation: The specified record contains the specified nonshared data item. The record definition specifies that this nonshared data item is a VAGen PACK (EGL DECIMAL) item with evensql=y. Your VAGen Migration Preferences specified that evensql=y is not to be honored. Based on your preferences, the migration tool converted the PACK item to the next higher odd precision, with a maximum length of 18.

The migration tool determines the EGL precision for PACK items in the following way:

- If evensql=n is specified for an item, the EGL precision is always calculated as (VAGen bytes * 2) - 1 with a maximum value of 18.
- If evensql=y is specified for an item, the EGL precision is calculated based on the preference:
 - If the **Do not honor evensql=y for items or variables** preference is selected, the EGL precision is calculated as:
$$(VAGenBytes * 2) - 1$$

with a maximum value of 18. Message IWN.MIG.0211.w is only issued in this situation.

- If the preference is not selected (honor evensql=y), the EGL precision is calculated as
$$(VAGenBytes * 2) - 2$$

with a maximum value of 18. Message IWN.MIG.0211.w is not issued.

User response: None. However, you might want to review the use of this item in any SQL WHERE clauses or EGL **prepare** statement. There might be an impact on performance if the definition of this item does not exactly match the SQL table definition. For details, see information about EVENSQ in “Eliminating the use of VisualAge Generator compatibility mode” on page 225 and “PACK data items with even length” on page 65.

IWN.MIG.0212.e Record *recordName* - alternate specification record *altspecRecord* is not available; cannot determine whether any item names require an override for the dliFieldName property.

Explanation: The record *recordName* specifies an alternate specification record named *altspecRecord*, which provides the item structure for *recordName*. In VisualAge Generator, the field names in a DL/I segment record must match the names in the DL/I PSB. In EGL, field names cannot be reserved words or start with the # or @ symbol. EGL also permits the field names in a DL/I segment record to be longer than the 8-character limitation imposed for the DL/I PSB. If the field name in a DL/I PSB does not match the EGL field name, EGL uses the **dliFieldName** property to provide the name used in the DL/I PSB for the corresponding EGL field name. Because the alternate specification record is not available, the migration tool cannot determine whether any of the field names are renamed due to the EGL naming conventions.

User response: Review the record definition for *altspecRecord* to determine if any field names in the record must be renamed due to the EGL naming conventions. If any field name was renamed, edit *recordName* and add overrides to the **embed** keyword for each renamed field to specify its corresponding **dliFieldName** property. The value for the

dliFieldName property must be the original VAGen field name.

For an example of how to code the **embed** keyword with an override for a field, see Table 82 on page 274.

IWN.MIG.0251.w UI record *recordName* is a reserved word or starts with the # or @ symbol. It was renamed to *newRecordName*.

Explanation: The UI record *recordName* conflicts with an EGL reserved word or starts with the # or @ symbol. The Stage 2 migration tool renamed the UI record to *newRecordName*, based on the **Renaming prefix** you specified in your migration preferences. The Stage 2 tool also includes the **alias** property for the VGUI record so that the names in the EGL generated outputs are identical to those in VisualAge Generator. The Stage 3 migration tool changed the file name for the VGUI record to *newRecordName.egl* because EGL requires that the VGUI record name and its file name must match. If you migrate in single file mode, the migration tool makes the same changes.

User response: None.

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 **validatorDataTable** properties
- Form field **validatorDataTable** properties
- VGUI record field **validatorDataTable** properties

If you want to keep the original table name as the name for the generated DataTable, set the **alias** property to the original DataTable name. If you do not specify the **alias** property, be sure to change any non-EGL references to the DataTable name, including CICS program definitions.

IWN.MIG.0302.w Table *tableName* - has only one row of contents. Check its use as the source of a MOVEA statement.

Explanation: In VisualAge Generator, when a table with a single row of contents is used as the source of a MOVEA statement, the source is treated as a scalar and the target array is completely initialized by the scalar source. This is contrary to the VisualAge Generator documentation, which indicates that the table should always be treated as an array, which in turn would cause only the first element of the target array to be initialized. In EGL, a **move** with a **for** modifier is

always treated as a move of one array to another, so only the first element of the target array is initialized.

User response: Review your programs to determine whether you used the specified table as the source of a MOVEA statement. If the table name is used as a qualifier for the source field in a MOVEA statement, the migration tool also issues message HPT.EGL.0710.e. To find MOVEA statements that might reference the fields in the table without using the table name as a qualifier, you might run DB2 queries against the migration database. For details, see “Queries to assist with specific error messages” on page 465. If the table is used as the source of a MOVEA in VisualAge Generator, modify the program logic to use a loop to initialize the target array from the source element of the table.

IWN.MIG.0401.e Map group (FormGroup) name *mapGroupName* is a reserved word. It must be renamed.

Explanation: The migration tool does not rename map groups (FormGroups) for you.

User response: 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.

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 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. If two or more VAGen devices convert to identical EGL device size and margin specifications, the migration tool only includes one entry for EGL. If the VAGen devices convert to identical EGL device sizes but have different margin specifications, the migration tool includes both entries. The FormGroup 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 FormGroup and delete all except one floating area specification for each group of same-size devices.

IWN.MIG.0403.e FormGroup *FormGroupName* - Requires editing to nest forms within the FormGroup.

Explanation: When you migrate in single file mode, the migration tool does not nest forms within the FormGroup. Instead, the migration tool inserts an EGL **use** statement to indicate the name of the forms that belong to the FormGroup. The migration tool includes comments at the beginning and end of each form to indicate its FormGroup.

User response: Edit the file containing the FormGroup and move the forms so that they are nested within the FormGroup. The **use** statements in the FormGroup indicate where the forms should be moved. After you have nested the form within the FormGroup, remove the **use** declaration statement.

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 both the main map group and the help map group for a program. EGL does not permit any duplicate form names in the two FormGroups for a program. This restriction applies even if the forms with duplicate names are not used by the program. The migration tool renames maps in the help map group for the program if they conflict with maps in the main map group for the program and the help maps only contain constant fields. The migration tool does not rename a map in the help map group if it contains variable fields, even if the name conflicts with a map name in the main map group. This is because the map could be used by some other program that specifies the help map group as the main map group for that program.

User response: Edit the help FormGroup 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 FormGroup. For additional considerations, see the information on map names in “Map names and help map names” on page 79.

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 **validatorFunction** function and a **validatorDataTable** property for a form field. The migration tool converts the map edit routine in the following way:

- EZEC10 and EZEC11 migrate to the **validatorFunction** property.

- If the part specified by *editRoutineName* is available during migration and is a function, the *editRoutineName* migrates to the **validatorFunction** property. The migration tool also migrates the edit routine to the **validatorFunction** 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 **validatorDataTable** property. The migration tool also migrates the edit routine to the **validatorDataTable** property if an edit message is specified for the form field 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 **validatorFunction** property. Message IWN.MIG.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 **validatorFunction** property to the **validatorDataTable** property. For additional considerations, see the information on edit routines in “Map variable fields and edit routines” on page 82.

IWN.MIG.0503.w Map group *mapGroupName*, map *mapName* - Unnamed variable field converted to constant field at position(*row,column*).

Explanation: VisualAge Generator permits 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 map variable fields” on page 85.

IWN.MIG.0504.w Map group *mapGroupName*, map *mapName* - Unnamed variable field removed from position(*row,column*).

Explanation: VisualAge Generator permits 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 map variable fields” on page 85.

IWN.MIG.0506.e Map Group *mapGroupName, map mapName* - **Unprotected constant at row, column; changed to protect=skipProtect.**

Explanation: VisualAge Generator permits unprotected constants on both display and printer maps. For constants, EGL requires that the **protect** property be set to either **skipProtect** or **protect**. The migration tool sets the **protect** property to **skipProtect** for the field.

User response: No action is required if **skipProtect** is acceptable. When the **protect** property is set to **skipProtect** for a constant field, the user can continue typing at the end of any immediately preceding variable field and the additional characters are placed in the next unprotected variable field. When the **protect** property is set to **protect** for a constant field, the user is prevented from continuing to type at the end of any immediately preceding variable field. The 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 **highlight** 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 **highlight** for the field because this information was not recorded in the External Source Format file. EGL validation displays an error message 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 compensate 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 **highlight** for the field because this information was not recorded in the External Source Format file. EGL validation displays an error message 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 help map group for the program 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 help map group for the program.
- The map is in the main map group for the program.

User response: Edit the FormGroup 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 FormGroup. For additional considerations, see the information on map names in “Map names and help map names” on page 79.

IWN.MIG.0512.e Map Group *mapGroupName, map mapName* - **duplicate cursor removed from field** *fieldName*.

Explanation: In some customizations of VisualAge Generator Templates (VAGT), multiple cursors are specified on a map. In this case, VisualAge Generator tolerates the duplicate cursor. For test facility and generation, VisualAge Generator places the cursor on the first field that specifies the cursor and ignores all the other fields. The first field in this case is the first in row and column order, not first in the edit order. EGL does not tolerate a duplicate cursor. The migration tool removes the **cursor** property from all fields except the first field that specifies the cursor. If multiple cursors are specified on array elements, the migration tool removes the **cursor** property from all elements of the array except the first element that specifies the cursor.

User response: No action is required if the cursor position on the first field that specifies the cursor is acceptable.

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 functions in a program, VisualAge Generator requires that the REPLACE function specifies the name of the corresponding UPDATE or SETUPD function. 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 assumes that the UPDATE function is for a non-SQL record and does not include the resultSetID.

User response: If EGL validation displays an error message 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 forUpdate** statement. The resultSetID must be unique within the program. To ensure this, use the function name followed by the **Result Set suffix** preference you used

during migration as the resultSetID. For additional considerations, see the information in “SQL I/O with multiple UPDATE or SETUPD functions” on page 104.

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 I/O option for maps” on page 94.

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 ensure that EGL validation displays an error message. 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 EZE_UNKNOWN_SQLTABLE in Appendix D, “Messages in the Problems view,” on page 427.

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*, or its alternate specification record, 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 in “SQL I/O and !itemColumnName” on page 103.

IWN.MIG.0605.w Function *functionName*, SQL I/O object *recordName* - SQLEXEC with model=none and no SQL clauses.

Explanation: The SQLEXEC I/O option specifies a model type of none, but does not contain any SQL clauses. The *recordName* is omitted if no I/O object is specified in VisualAge Generator. VisualAge Generator in effect generates a no op for this I/O option. 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** blocks appropriate for that error routine.

User response: Review the function to determine whether the I/O statement should be eliminated or expanded.

IWN.MIG.0607.e Function *functionName*, SQL I/O object *recordName* - Unable to determine SQL I/O clause *clauseName*.

Explanation: The specified *recordName* is not available during migration. The migration tool is unable to create the specified clause in one of the following situations:

- The function uses modified SQL, but some of the clauses are missing. 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 *clauseNames* that might be listed in this situation are: SELECT, INTO, INSERTCOLNAME, VALUES, and FORUPDATEOF.
- The function uses default SQL and specifies the **Execution time statement build** option. In this situation, VisualAge Generator creates all the clauses from the record definition that is specified as the I/O object for the function. The *clauseNames* that might be

listed in this situation are: SELECT, INTO, INSERTCOLNAME, VALUES, FORUPDATEOF, SET, WHERE, and ORDERBY.

The migration tool builds a skeleton 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 the VAGen SQL Statement Editor to view the SQL clauses. For more information and for potential problems, see “SQL I/O and missing required SQL clauses” on page 98 or “SQL I/O and Execution time statement build” on page 102. See the information on EZE_UNKNOWN_SQL_ *clauseName* in Appendix D, “Messages in the Problems view,” on page 427.

IWN.MIG.0608.e Function *functionName*, SQL I/O object *recordName* - Unable to determine SQL I/O clause *clauseName* for alternate specification *altspecRecordName*.

Explanation: The specified *recordName* is available during migration. However, the *recordName* specifies an alternate specification record *altspecRecordName* which is not available during migration. The migration tool is unable to create the specified clause in one of the following situations:

- The function uses modified SQL, but some of the clauses are missing. 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 *clauseNames* that might be listed in this situation are: SELECT, INTO, INSERTCOLNAME, VALUES, and FORUPDATEOF.
- The function uses default SQL and specifies the **Execution time statement build** option. In this situation, VisualAge Generator creates all the clauses from the record definition that is specified as the I/O object for the function. The *clauseNames* that might be listed in this situation are: SELECT, INTO, INSERTCOLNAME, VALUES, FORUPDATEOF, SET, WHERE, and ORDERBY.

The migration tool builds a skeleton 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 the VAGen SQL Statement Editor to view the SQL clauses. For more information and for potential problems, see “SQL I/O and missing required SQL clauses” on page 98 or “SQL I/O and Execution time statement build” on page 102. See the information on EZE_UNKNOWN_SQL_ *clauseName* in Appendix D, “Messages in the Problems view,” on page 427.

IWN.MIG.0609.e Function *functionName* - record *recordName* in SSAs is not available; qualification of comparison value item *itemName* cannot be determined.

Explanation: The modified DL/I statement used an unqualified comparison value item. By default, VisualAge Generator searches first for the item in the DL/I segment record associated with the current SSA. The specified DL/I segment record *recordName* associated with the current SSA is not available. Therefore, the migration tool is not able to determine the qualification for the comparison value item.

User response: Locate and review the record specified in the message. If the item is in the record, edit the function to include the missing qualification for the comparison value item. If the item is not in the record, review your program logic to determine the correct qualification to use. You can also review the generated COBOL source code from the last time you generated the program. In VisualAge Generator, at some points in time, the rules for qualification of the comparison value item varied. Therefore, due to the variations in the qualification of the comparison value item, do not regenerate the program using your current release of VisualAge Generator unless you are certain that the release has not changed since the last time you generated the program.

IWN.MIG.0610.e Function *functionName* - record *recordName* in SSAs has alternate specification record *altspecRecordName* that is not available; qualification of comparison value item *itemName* cannot be determined.

Explanation: The modified DL/I statement used an unqualified comparison value item. By default, VisualAge Generator searches first for the item in the DL/I segment record associated with the current SSA. The specified DL/I segment record *recordName* associated with the current SSA is available during migration. However, *recordName* specifies an alternate specification record *altspecRecordName* which is not available during migration. Therefore, the migration tool is not able to determine the qualification for the comparison value item.

User response: Locate and review the records specified in the message. If the item is in the alternate specification record, edit the function to include the missing qualification for the comparison value item. If the item is not in the record, review your program logic to determine the correct qualification to use. You can also review the generated COBOL source code from the last time you generated the program. In VisualAge Generator, at some points in time, the rules for qualification of the comparison value item varied. Therefore, due to the variations in the qualification of the comparison value item, do not regenerate the program using your current release of VisualAge

Generator unless you are certain that the release has not changed since the last time you generated the program.

IWN.MIG.0611.e Function *functionName* - comparison value item *itemName* is not in record *recordName*; qualification cannot be determined.

Explanation: The modified DL/I statement used an unqualified comparison value item. By default, VisualAge Generator searches first for the item in the DL/I segment record associated with the current SSA. The migration tool searched the DL/I segment record associated with the current SSA, but could not find the item. In VisualAge Generator, at some points in time, the rules for qualification of the comparison value item varied. The migration tool is not able to determine which record should be used to qualify the comparison.

User response: Review your program logic to determine the correct qualification to use. You can also review the generated COBOL source code from the last time you generated the program. Due to the variations in the qualification of the comparison value item, do not regenerate the program using your current release of VisualAge Generator unless you are certain that the release has not changed since the last time you generated the program.

IWN.MIG.0612.e Function *functionName* - invalid relational operator for SSA; correct operator cannot be determined.

Explanation: The modified DL/I statement used a relational operator that is invalid. This can occur due to a problem in VisualAge Generator that caused it to store an incorrect value for the relational operator. The migration tool is not able to determine the correct relational operator. The migration tool uses EZE_UNKNOWN_RELOP as the relational operator.

Note: The most likely operators to cause the problem are the symbols used for not equal. The symbol for not equal in an EGL SSA is !=.

User response: Use the DL/I Call Editor in VisualAge Generator to review the SSAs for the specified function. The correct operator is shown in the DL/I Call Editor even though it is stored incorrectly in the External Format File for the function. Edit the function in EGL and change EZE_UNKNOWN_RELOP to the correct value.

IWN.MIG.0613.w Function *functionName* - no I/O error routine when using Execution time statement build.

Explanation: In VisualAge Generator, the specified function for an INQUIRY, UPDATE, SETINQ, or SETUPD I/O option uses the **Execution time statement build** option, but does not specify an I/O error routine.

At generation time, this results in an SQL PREPARE statement, followed by an OPEN, and then (for INQUIRY or UPDATE) followed by a FETCH. If a soft error occurs on the SQL PREPARE statement, processing continues. The migration tool converts the I/O option to an EGL **prepare** statement followed by either a **get** (for INQUIRY or UPDATE) or **open** (for SETINQ or SETUPD). However, because there was no I/O error routine in VisualAge Generator, the migration tool cannot include a **try...onException** block around the EGL I/O statements and therefore cannot include additional logic so that processing continues after a soft error on the EGL **prepare** statement. As a result of the migration, if a soft error occurs on the EGL **prepare** statement, processing stops.

User response: If a soft error is possible for the **prepare** statement and you want processing to continue, place a **try...onException** block around the EGL **prepare** statement and include the appropriate error handling logic so that processing can continue after the soft error.

IWN.MIG.0614.w Function *functionName* - uses SQLEXEC with Execution time statement build.

Explanation: In VisualAge Generator, the specified function uses the SQLEXEC I/O option and also specifies the **Execution time statement build** option. At generation time, this results in an SQL EXECUTE IMMEDIATE which causes a PREPARE, EXECUTE, and DESTROY to be done by this single SQL statement. The migration tool converts the SQLEXEC to an EGL **prepare** statement followed by an **execute** statement, which is the closest EGL equivalent.

User response: None. However, test any programs that use this function to be sure that the behavior and performance are the same as in VisualAge Generator.

IWN.MIG.0701.e Function *functionName* - Unable to determine map type for *mapName* used in SET map PAGE statement; used converseLib.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 **converseLib.clearScreen()** function only for text forms and the **converseLib.pageEject()** function 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 converseLib.EZE_SETPAGE() placeholder to ensure that EGL validation displays an error message. The migration tool includes the original map name as a comment.

User response: Review the function and determine whether **converseLib.clearScreen()** or

converseLib.pageEject() is the correct choice. For additional considerations, see the information in “SET map PAGE statement” on page 110.

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 ensure that EGL validation displays an error message. 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 in “RETR statement” on page 110.

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 ensure that EGL validation displays an error message. 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 in “RETR statement” on page 110.

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 ensure that EGL validation displays an error message. 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 in “FIND statement” on page 109.

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 ensure that EGL validation displays an error message.

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 values:

- **unique** for an SQL record
- **duplicate** for a non-SQL record

For additional considerations, see the information in “I/O error values UNQ and DUP” on page 114.

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 form field for **blanks** and an SQL field for **null**. The item specified in *itemName* is not available during migration. The migration tool migrates NULL to EZE_NULL to ensure that EGL validation displays an error message.

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 values:

- **blanks** for a form field
- **null** for an SQL field

For additional considerations, see the information in “Checking SQL and map items for NULL” on page 113.

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 *custPrefix*EZESYS, where *custPrefix* is the **Renaming prefix** preference you set for migration. When migrating programs, if you clear the **Do not initialize old EZESYS values** migration preference, the migration tool includes a declaration for *custPrefix*EZESYS and a statement to initialize *custPrefix*EZESYS to the original VAGen values. The original VAGen values are 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 new EGL values, change *custPrefix*EZESYS to **sysVar.systemType**.

If you want to use the original VAGen values and you selected the **Do not initialize old EZESYS values** migration preference during migration, you must add a declaration and an initialization statement for *custPrefix*EZESYS to any program that uses the specified function. If you want to use the original VAGen values and you cleared the **Do not initialize old EZESYS values** migration preference, no change is necessary. The declaration and initialization statements for *custPrefix*EZESYS are already included in all the migrated programs.

IWN.MIG. 0710.e Function *functionName* - MOVEA for table *tableName*, but the table only has one row.

Explanation: The MOVEA statement specifies a table as the qualifier for the source field. In VisualAge Generator, when a table with a single row of contents is used as the source of a MOVEA statement, the source is treated as a scalar and the target array is completely initialized by the scalar source. This is contrary to the VAGen documentation, which indicates that the table should always be treated as an array, which in turn causes only the first element of the target array to be initialized. In EGL, a **move** with a **for** modifier is always treated as a move of one array to another, so only the first element of the target array is initialized. If this problem occurs in program flow statements, the program name appears in the message instead of a function name.

User response: Review the program logic to verify that you intended to initialize the entire target array. If so, modify the program logic to use a loop to initialize the target array from the source element of the table.

IWN.MIG.0711.w Function *functionName* - MOVEA for qualifier *tableName*, but the qualifier is not available.

Explanation: The MOVEA statement specifies a qualifier for the source field. In VisualAge Generator, when a table with a single row of contents is used as the source of a MOVEA statement, the source is treated as a scalar and the target array is completely initialized by the scalar source. This is contrary to the VAGen documentation, which indicates that the table should always be treated as an array, which in turn causes only the first element of the target array to be initialized. In EGL, a **move** with a **for** modifier is always treated as a move of one array to another, so only the first element of the target array is initialized. Because the qualifier is not available, the migration tool is not able to determine if it is a table that only has a single row of contents. If this problem occurs in program flow statements, the program name appears in the message instead of a function name.

User response: If the message is issued when a map group is being migrated, you can ignore this message. If the message is issued when a program is being migrated, review the definition of the specified qualifier to determine whether it is a record, a map, or a table with multiple rows. If so, no action is necessary. If the qualifier is a table with only a single row of contents, see the User response for message IWN.MIG.0710.e for information on how to resolve the problem.

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 and references in linkage options parts. Also change the names of any bind control or link edit 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 and transaction 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 EGL, edit the program definition and add the corresponding EGL primitive field declarations. You do not need to create a record to contain the fields. You can add the primitive field declarations directly to the program. For information about a white paper that can help you create the implicit items before you run Stage 1 of migration, see "References" on page 16.

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 FormGroup. 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, EGL validation displays an error message in the Problems view. Edit the program, remove the record declaration, and add the **use** declaration for the 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 sets the **segmented** property to NO 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 YES.

IWN.MIG.0806.w Program *programName*, use declaration for table *tableName* - preferences caused deleteAfterUse=yes to be omitted.

Explanation: The specified program contains a **use** declaration for the specified table. In Cross System Product and some releases of VisualAge Generator, the keep after use flag determined when memory for a table was freed by a program. The VisualAge Generator keep after use flag is normally migrated to the EGL **deleteAfterUse** property on the **use** declaration for the table. However, your VAGen Migration Preferences specified that the migration tool should not include the **deleteAfterUse** property.

User response: None. If your VAGen programs were generated using VisualAge Generator 4.5 Fix Pack 4 or later, then there should not be any difference in behavior. For details, see information about **deleteAfterUse** in "Eliminating the use of VisualAge Generator compatibility mode" on page 225.

IWN.MIG.0807.e Program *programName* - PSB *psbName* is not available; DLI segment records for the PSB cannot be determined.

Explanation: In VisualAge Generator, all the DL/I segment records specified in the PSB for the program are automatically included at test or generation time because they might be used in creating a default SSA or explicitly used in a modified SSA. In EGL, the DL/I segment records must be explicitly declared in the program if they are used in a default SSA or explicitly used in a modified SSA. The specified *psbName* is not available during migration. Therefore, the migration tool cannot determine whether any DL/I segment

records need to be added to the list of data declarations for the program.

User response: Check the EGL Problems view for a message about an ambiguous or unresolved DL/I segment records for this function or program. If there is no message, this indicates that a declaration for the DL/I segment record is already included for the program (most likely as the result of I/O directly against the DL/I segment record). If there is a message, edit the program and add the declaration for the DL/I segment record.

IWN.MIG.0808.e Called program *programName* - PSB *psbName* is not available; PCB types cannot be determined.

Explanation: In VisualAge Generator, EZEDLPCB[*n*] where *n* is a numeric literal, is used to indicate a PCB that is passed to a program as a parameter. VisualAge Generator uses the PCB without regard to whether the PCB is an I/O, alternate, database, or GSAM PCB. EZEDLPCB[0] is always the I/O PCB and is not explicitly listed in the VAGen PSB part. An error occurs if the DL/I PCB is not of the correct type when the PCB is used at runtime. In EGL, the PCB name from the PSBRecord for the program is used as the parameter name. The PCB type must be explicitly specified for each program parameter by giving the appropriate type definition using a record name (IO_PCBRecord, ALT_PCBRecord, DB_PCBRecord, or GSAM_PCBRecord). The migration tool always uses IO_PCBRecord as the type definition for EZEDLPCB[0]. The program *programName* specifies one or more EZEDLPCB[*n*] parameters where *n* is greater than 0. However, the specified *psbName* is not available during migration. Therefore, the migration tool cannot determine the type to include for the PCB in the parameter list. The migration tool uses EZE_UNKNOWN_PCB_TYPE for all the PCBs in the parameter list.

User response: Locate the specified PSBRecord. Edit the program and change the type definitions to specify the correct xxxx_PCBRecord based on the corresponding PCB type in the specified EGL PSBRecord.

IWN.MIG.0809.e Called program *programName* - PSB *psbName* is not available; PCB mapping cannot be determined.

Explanation: In VisualAge Generator, EZEDLPCB[*n*] where *n* is a numeric literal, is used to indicate a PCB that is passed to a program as a parameter. VisualAge Generator automatically associates EZEDLPCB[*n*] with the corresponding PCB in the VAGen PSB part. EZEDLPCB[0] is always the I/O PCB and is not explicitly listed in the VAGen PSB part. An error occurs if the DL/I PSB does not have the expected number of PCBs at runtime. In EGL, the PCB name from the PSBRecord for the program is used as the parameter

name. The **pcbParms** property is used to explicitly associate each PCB in the parameter list with the corresponding position within the EGL PSBRecord. The program *programName* specifies one or more EZEDLPCB[*n*] parameters, but the specified *psbName* is not available during migration. Therefore, the migration tool cannot determine the number of PCBs to include in the **pcbParms** property. The migration tool uses EZE_UNKNOWN_PCB_MAPPING for the value of the **pcbParms** property.

User response: Locate the specified PSBRecord. Edit the program and change the **pcbParms** property to provide the mapping between the PCB parameters and the PCBs in the specified EGL PSBRecord.

IWN.MIG.0810.e Called program *programName* - parameter list references higher PCB numbers than exist in PSB *psbName*; PCB types and PCB mapping are not complete.

Explanation: In VisualAge Generator, EZEDLPCB[*n*] where *n* is a numeric literal, is used to indicate a PCB that is passed to a program as a parameter. VisualAge Generator automatically associates EZEDLPCB[*n*] with the corresponding PCB in the VAGen PSB part. The association is done without regard to whether the PCB is an I/O, alternate, database, or GSAM PCB. EZEDLPCB[0] is always the I/O PCB and is not explicitly listed in the VAGen PSB part. A runtime error occurs if the DL/I PSB does not have the expected number of PCBs or if the DL/I PCB is not of the correct type. In EGL, the PCB name from the PSBRecord for the program is used as the parameter name. The PCB type must be explicitly specified for each program parameter by giving the appropriate type definition using a record name (IO_PCBRecord, ALT_PCBRecord, DB_PCBRecord, or GSAM_PCBRecord). The migration tool always uses IO_PCBRecord as the type definition for EZEDLPCB[0]. In addition, the **pcbParms** property is used to explicitly associate each PCB in the parameter list with the corresponding position within the EGL PSBRecord. The program *programName* specifies one or more EZEDLPCB[*n*] parameters, but some of the values for *n* are greater than the number of PCBs in the specified *psbName*. Therefore, the migration tool cannot determine the type definition to include for some of the PCBs in the parameter list. The migration tool uses EZE_UNKNOWN_PCB_TYPE for any PCB in the parameter list that does not correspond to a PCB in the specified *psbName*. In addition, the migration tool cannot determine the number of PCBs to include in the **pcbParms** property. The migration tool creates PCB mapping information for all EZEDLPCB[*n*] parameters, up to and including the highest value of *n*. However, this list does not match the available PCBs in the specified *psbName*.

User response: Locate the specified PSBRecord. Review the PSBRecord and the program logic to

determine which is correct. Add any additional PCBs to the PSBRecord. Edit the program and change the parameter type definitions to specify the correct xxxx_PCBRecord based on the corresponding PCB type in the specified EGL PSBRecord. Also change the **pcbParms** property to provide the correct mapping between the PCB parameters and the PCBs in the specified EGL PSBRecord.

IWN.MIG.0811.w Program *programName* - does not appear to use any maps; use statement for FormGroup *FormGroupName* commented out.

Explanation: VisualAge Generator requires that a main transaction or called transaction program always specify map group, even if the program does not use any maps. In this situation, the map group part did not have to exist. EGL does not require that a program specify a FormGroup unless the program actually uses a form. The migration tool determined that the program specifies a map group, but does not appear to use a map as an I/O object, in an XFER with a map statement, as a called parameter, or as the First Map of the program. Therefore, the migration tool commented out the **use** statement for the FormGroup in the EGL program. The migration tool also commented out the **use** statement for the help FormGroup if one was present.

User response: None. However, if there are additional functions for this program that were not included in the migration set or in the External Source Format file, these functions might use maps from the specified map group. If you need to use forms within the FormGroup, change the EGL program to uncomment the **use** statement for the FormGroup and help FormGroup. If there are multiple forms within the form group, you might need to change the **use** declaration so that it specifies the specific forms within the FormGroup that the program uses. Listing specific forms can help avoid unresolved or ambiguous references in EGL.

IWN.MIG.0901.w PSB *psbName* has multiple PCBs with the same database name *databaseName*.

Explanation: In VisualAge Generator, for a DL/I I/O function, you can specify which PCB to use by specifying either the database name or the PCB number from the VAGen PSB. The same database name can be specified for multiple PCBs in the VAGen PSB. There is no PCB name in the VAGen PSB. The actual database name is never used by test, generation, or runtime. The DL/I PSBs do not need to include the PCBNAME parameter. In EGL, for debugging, the PCBNAME parameter (or a label to provide the PCB name) is required for each database PCB in the DL/I PSB. The PCBNAME parameter must match the **pcbName** property for the corresponding PCB in the EGL PSBRecord. The migration tool uses the original

database name from the VAGen PSB concatenated with your **Database PCB suffix** migration preference to create the name of the EGL PCB record. The migration tool also sets the **pcbName** property for the EGL PCB record to the original database name from the VAGen PSB. In general, when you add the PCBNAME to your DL/I PSBs, you can use the VAGen database name as the PCBNAME. However, the PCBNAMEs must be unique in the DL/I PSB. Therefore, the migration tool issues this message whenever there are multiple PCBs in the VAGen PSB with the same database name. In this situation, the **pcbName** property set by the migration tool cannot be used for all of the PCBs.

User response: When you modify your DL/I PSB to include the PCBNAME parameters, you must use unique PCB names. After you set the PCB names in your DL/I PSB, you must update the **pcbName** property EGL PSBRecord to reflect the actual PCB names in your DL/I PSB.

IWN.MIG.1001.e Generation options part *partName* - 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.

Note: In EGL, Oracle is supported only if you use Java generation.

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. For more information, see "Differences in SQL support" on page 233.

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 a COBOL runtime environment. The EGL build process requires you to specify a destination port using the **destPort** build descriptor option.

User response: Modify the build descriptor part that corresponds to the generation options part and include the **destPort** build descriptor option. Consider the following environments when specifying the value of **destPort**:

- For z/OS environments, there is no default value for **destPort**. You must add the **destPort** build descriptor option and the value must match the value you use in the JCL that starts the z/OS build server. The sample JCL for starting a z/OS build server uses port 5555.
- For iSeries environments, there is no default value for **destPort**. You must add the **destPort** build descriptor option. The value must match the value used by the iSeries build server.
- For VSE environments, the default value for **destPort** is 21. You only need to specify the **destPort** build description option if the value is different from 21.

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, in EGL the **SymbolicParameter** is not set during generation.

User response: Modify the control part to set a default value for the **SymbolicParameter**. Alternatively, modify the control part so that it no longer uses the specified **SymbolicParameter**.

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 sets the **conversionTable** property in the EGL linkage options part to "BINARY". This value is invalid, but is not detected until generation.

User response: You must change the **conversionTable** property to a value that is supported by EGL. Refer to the information about linkage options parts in the online help for details about the EGL **conversionTable** property and the options that are available.

IWN.MIG.1103.e Linkage table part *partName* - /remotecomtype=CICSECI is not supported. Defaulted to CICSECI.

Explanation: VisualAge Generator supports /remotecomtype=CICSECI in the linkage table part. EGL does not support this value. The migration tool includes the **remoteComType**="CICSECI" in the EGL linkage options 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** properties.

User response: If you plan to use CICSECI, modify the linkage options 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 options parts in the online help for details about the EGL **remoteComType** property 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 sets the **remoteComType** property to "communicationType" in the EGL linkage options 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 options parts in the online help for details about the EGL **remoteComType** property 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 associations 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 associations 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 associations part. The value is invalid; EGL validation displays an error message 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 associations parts in the online help for details about the EGL **filetype** property 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 associations 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 mvsv* and the *fileType* is transient, the migration tool creates an EGL resource association entry and sets the EGL **system** to mvsv*. This is invalid; EGL validation displays an error message in the Problems view. You can correct the entry by specifying a valid EGL **system** (zosscics for this example).

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 link edit part, the migration tool also does not rename the link edit part.

User response: When you change the program name, be sure to change the name of the corresponding link edit 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 VisualAge Generator to EGL migration tool—Stage 3

The following messages are produced by Stage 3.

IWN.MIG.0030.i Migration set *Name_version* -- import analysis started.

Explanation: This is an information message to indicate status from the migration tool.

User response: None.

IWN.MIG.0031.i Migration set *Name_version* -- import analysis completed.

Explanation: This is an information message to indicate status from the migration tool.

User response: None.

IWN.MIG.0032.i Migration set *Name_version* -- not processed for Stage 3.

Explanation: This is an information message to indicate status from the migration tool. The specified migration set version was not processed during Stage 3. This is because another version of the migration set is imported into the workspace. For example, you requested to import the latest version of the migration sets and the specified version is not the latest version of the specified migration set.

User response: None.

IWN.MIG.0033.e Exception merging EGL Source file *fileName* - *exceptionText*.

Explanation: During Stage 3 migration, you cleared the option to **Override existing files**. New parts from the current migration set are supposed to be merged into existing files. However, the migration tool was not able to do the merge for the specified file.

User response: Contact IBM support for assistance. Be prepared to provide the workspace and the migration database that you used during Stage 3 migration.

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 EGL validation displays an error message in the Problems view or a compile error occurs. The following table lists the specific text string that caused the error. The specific EGL error message or compiler message might vary, but the text string listed in the left column appears 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 163. VAGen migration text that causes EGL syntax or validation errors

VAGen migration text in EGL syntax	Problem and Solution
###KEYS_NOT_FOUND###	<p>Problem: The current SQL record embeds the structure from another record. During migration the record named by the embed keyword 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.</p> <p>Solution: Find the EGL record named by the embed keyword. 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 key item from the current record in the order that the fields appear in the record structure of the embedded record. If the key item from the current record is also specified in the keyItems property of the embedded record, only include the field once in the EGL keyItems property.</p>
###TABLES_NOT_FOUND###	<p>Problem: The current SQL record embeds the structure from another record. During migration the record named by the embed keyword was not available.</p> <p>Solution: Find the EGL record named by the embed keyword and copy the tableNames and tableNameVariables properties into the current SQL record.</p>
EZE_DUPLICATE	<p>Problem: The record named in a VAGen IF, WHILE, or TEST statement was not available during migration.</p> <p>Solution: Find the EGL record named on the EGL if or while statement. Change EZE_DUPLICATE to one of the following values:</p> <ul style="list-style-type: none">• duplicate for a non-SQL record• unique for an SQL record
EZE_NULL	<p>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.</p> <p>Solution: Review the program and determine whether the field is in an SQL record or on a form. Replace EZE_NULL with one of the following values:</p> <ul style="list-style-type: none">• null for an SQL field• blanks for a form field

Table 163. VAGen migration text that causes EGL syntax or validation errors (continued)

VAGen migration text in EGL syntax	Problem and Solution
EZE_SCRIPT	<p>Problem: In a VisualAge Generator GUI, EZESCRPT is used to invoke a Java or Smalltalk method. EGL does not have a corresponding system function. The migration tool uses EZE_SCRIPT to indicate a statement that cannot be converted to EGL.</p> <p>Solution: Either change the function or move it to a project that contains unused functions.</p>
EZE_SETPAGE();	<p>Problem: The map named on a VAGen SET <i>map</i> PAGE statement was not available during migration.</p> <p>Solution: Find the form named on the // VAGen Info comment that accompanies the EZE_SETPAGE() statement. Change EZE_SETPAGE to one of the following functions:</p> <ul style="list-style-type: none"> • converseLib.clearScreen() for a text form • converseLib.pageEject() for a print form
EZE_UNKNOWN_PARTTYPE	<p>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.</p> <p>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.</p> <p>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 459 for some queries that might be useful in determining what is causing the problem.</p>
EZE_UNKNOWN_PCB_MAPPING	<p>Problem: The PSB part specified for the program was not available during migration. The migration tool was not able to determine the values to specify for the pcbParms property.</p> <p>Solution: Find the data declaration in the program for the variable named psb. The type definition that is specified for psb is the name of the EGL PSBRecord part for the program. Change the pcbParms property for the program to map the input PCB parameters to the corresponding PCBs within the EGL PSBRecord part. For additional details, see message IWN.MIG.0809.e (on page 421).</p>
EZE_UNKNOWN_PCB_TYPE	<p>Problem: The PSB part specified for the program was not available during migration or contained fewer PCBs than specified by the parameter list for the program. The migration tool was not able to determine the type definitions to use for the PCB parameters for the program. Each PCB in the parameter list is <i>pcb_n</i>, where <i>n</i> is a numeric literal that corresponds to a PCB in the EGL PSBRecord.</p> <p>Solution: Find the data declaration in the program for the variable named psb. The type definition that is specified for psb is the name of the EGL PSBRecord part for the program. Change the PCB parameters for the program to specify the correct type definition record (IO_PCBRecord, ALT_PCBRecord, DB_PCBRecord, or GSAM_PCBRecord) based on the type definition for the corresponding PCB in the EGL PSBRecord. For additional details, see message IWN.MIG.0808.e (on page 421).</p>

Table 163. VAGen migration text that causes EGL syntax or validation errors (continued)

VAGen migration text in EGL syntax	Problem and Solution
EZE_UNKNOWN_QUALIFIER	<p>Problem: The current Segment Search Argument (SSA), contains an EGL host variable (VAGen comparison value item) that is not qualified. The DL/I segment record or its alternate specification record for the SSA was not available during migration. Alternatively, the record was available but did not contain the comparison value item. The migration tool was not able to determine the qualifier for the EGL host variable.</p> <p>Solution: Find the DL/I segment record or its alternate specification record for the current SSA. Determine whether the host variable is an item in the record. If so, change the qualifier for the host variable to the DL/I segment record name. If not, determine the correct qualifier to use. For additional details on how to determine the correct qualifier, see message IWN.MIG.0611.e (on page 417)</p>
EZE_UNKNOWN_RELOP	<p>Problem: The modified DL/I statement used a relational operator that is invalid. This can occur due to a problem in VisualAge Generator that caused it to store an incorrect value for the relational operator. The migration tool was not able to determine the correct relational operator.</p> <p>Solution: Use the DL/I Call Editor in VisualAge Generator to review the SSAs for the specified function. The correct operator is shown in the DL/I Call Editor even though it is stored incorrectly in the External Format File for the function. Edit the function in EGL and change EZE_UNKNOWN_RELOP to the correct value.</p> <p>Note: The most likely operators to cause the problem are the symbols used for not equal. The symbol for not equal in an EGL SSA is !=.</p>
EZE_UNKNOWN_RETURN_COLUMN	<p>Problem: The VAGen table named on the VAGen RETR statement was not available during migration.</p> <p>Solution: Find the EGL DataTable named on the assignment statement and replace EZE_UNKNOWN_RETURN_COLUMN with the name of the second column in the DataTable.</p>
EZE_UNKNOWN_SEARCH_COLUMN	<p>Problem: The VAGen table named on the VAGen FIND or RETR statement was not available during migration.</p> <p>Solution: Find the EGL DataTable named on the if statement and replace EZE_UNKNOWN_SEARCH_COLUMN with the name of the first column in the DataTable.</p>
EZE_UNKNOWN_SQLTABLE	<p>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.</p> <p>Solution: Find the record named in the I/O statement and determine the correct tables clause from the tableNames or tableNameVariables record properties, or both.</p>

Table 163. VAGen migration text that causes EGL syntax or validation errors (continued)

VAGen migration text in EGL syntax	Problem and Solution
EZE_UNKNOWN_SQL_FORUPDATEOF	<p>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.</p> <p>Solution: Find the record named in the I/O statement and determine the correct FOR UPDATE OF clause from the list of fields in the record. The default FOR UPDATE OF clause in VisualAge Generator is the list of column names from the record in the same order as the fields are listed in the record, but omitting the columns for the following cases:</p> <ul style="list-style-type: none"> 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. <p>If the record named in the I/O statement embeds another SQL record, use the two records to determine the column names in the following way:</p> <ul style="list-style-type: none"> Use the record named by the embed keyword to determine the order of the columns and the isReadOnly = YES property. Use the record named in the I/O statement (the embedding record) to determine the keyItems property. <p>If the FOR UPDATE OF clause is used in an EGL prepare statement, enclose the list of column names within double-quotes.</p>
EZE_UNKNOWN_SQL_INSERTCOLNAME	<p>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.</p> <p>Solution: Find the record named in the I/O statement and determine the correct list of columns from the list of fields in the record. The default list of column names in VisualAge Generator is the list of column names from the record in the same order as the fields are listed in the record, but omitting any column name that is specified with the EGL isReadOnly = YES property. If the record named in the I/O statement embeds another record, use the record named by the embed keyword 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.</p>
EZE_UNKNOWN_SQL_INTTO	<p>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.</p> <p>Solution: Find the record named in the I/O statement and determine the correct list of fields for the INTO clause. The default list of fields in VisualAge Generator is the list of fields from the record in the same order as the fields are listed in the record. If the record named in the I/O statement embeds another record, use the record named by the embed keyword to determine the order of the fields. The INTO clause is never used in an EGL prepare statement.</p>

Table 163. VAGen migration text that causes EGL syntax or validation errors (continued)

VAGen migration text in EGL syntax	Problem and Solution
EZE_UNKNOWN_SQL_ORDERBY	<p>Problem: VisualAge Generator created a default list of column positions for the SQL ORDER BY clause for the SQL SETINQ I/O option when default SQL was used and the Execution time statement build option was specified. 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 ORDER BY clause for the EGL I/O statement.</p> <p>Solution: Find the record named in the I/O statement and determine the correct list of column positions for the ORDER BY clause. The default list of column positions in VisualAge Generator is the list of field positions that correspond to each item in the EGL keyItems property followed by the ASC option. The first field in the record is considered to be position 1. If the record named in the I/O statement embeds another record, use the EGL keyItems property for the record named in the I/O statement, not the embedded record, to determine the ORDER BY clause. Create the ORDER BY clause for the EGL prepare statement in the following format: "order by keyField1Position, keyField2Position asc"</p>
EZE_UNKNOWN_SQL_SELECT	<p>Problem: VisualAge Generator created a default list of columns 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.</p> <p>Solution: Find the record named in 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 fields are listed in the record. If the record named in the I/O statement embeds another record, use the record named by the embed keyword 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.</p>

Table 163. VAGen migration text that causes EGL syntax or validation errors (continued)

VAGen migration text in EGL syntax	Problem and Solution
EZE_UNKNOWN_SQL_SET	<p>Problem: VisualAge Generator created a default list of columns and values for the SQL SET clause when the SQL SQLEXEC I/O option was used with default SQL, the model option was set to UPDATE and the Execution time statement build option was specified. 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 SET clause for the EGL I/O statement.</p> <p>Solution: Find the record named in the I/O statement and determine the correct list of columns and values for the SET clause. The default list of columns and values in VisualAge Generator is the list of columns and their corresponding field names from the record in the same order as the columns and fields are listed in the record, but omitting the column and its corresponding field for the following cases:</p> <ul style="list-style-type: none"> 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. <p>If the record named in the I/O statement embeds another SQL record, use the two records to determine the column names in the following way:</p> <ul style="list-style-type: none"> Use the record named by the embed keyword to determine the order of the columns and the isReadOnly = YES property. Use the record named in the I/O statement (the embedding record) to determine the keyItems property. <p>Create the SET clause for the prepare statement in the following format:</p> <pre>" set columnName1 = " + fieldName1 + " , columnName2 = " + fieldName2</pre>
EZE_UNKNOWN_SQL_SQLEXEC	<p>Problem: One of the following situations occurred for an SQLEXEC I/O option in VisualAge Generator:</p> <ul style="list-style-type: none"> No record and no SQL clause were specified. A record was specified but the Model option was set to NONE. <p>The function is invalid in VisualAge Generator. Therefore, the migration tool cannot determine what I/O statement, if any, is supposed to be created.</p> <p>Solution: Review your VAGen source code to determine was is really intended for this function.</p>

Table 163. VAGen migration text that causes EGL syntax or validation errors (continued)

VAGen migration text in EGL syntax	Problem and Solution
EZE_UNKNOWN_SQL_VALUES	<p>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 field names for the VALUES clause of the EGL add statement.</p> <p>Solution: Find the record named in the I/O statement and determine the correct list of fields for the VALUES clause. The default list of field names in VisualAge Generator is the list of fields from the record in the same order as the fields are listed in the record, but omitting any field that is specified with the EGL isReadOnly = YES property. If the record named in the I/O statement embeds another record, use the record named by the embed keyword to determine the order of the fields and the isReadOnly = YES property. The VALUES clause is never used in an EGL prepare statement.</p>
<p>EZE_UNKNOWN_SQL_WHERE</p> <p>where the VAGen I/O option is one of the following options:</p> <ul style="list-style-type: none"> • INQUIRY • UPDATE • SQLEXEC with the model option set to UPDATE or DELETE 	<p>Problem: VisualAge Generator created an SQL WHERE clause for the SQL INQUIRY, UPDATE, or SQLEXEC I/O options when default SQL was used and the Execution time statement build option was specified. 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 WHERE clause for the EGL I/O statement.</p> <p>Solution: Find the record named in the I/O statement and determine the correct WHERE clause from the defaultSelectCondition and keyItems properties in the EGL record. Create the WHERE clause for the prepare statement in the following way:</p> <ul style="list-style-type: none"> • If there is a defaultSelectCondition and there are no key items, create the WHERE clause from the defaultSelectCondition. For example: " where " + <i>recordDefaultSelectCondition</i> • If there is no defaultSelectCondition, but there are one or more fields listed in the EGL keyItems property, create the WHERE clause using all key items. For example: " where <i>keyColumn1</i> = " + <i>keyField1</i> + " and <i>keyColumn2</i> = " + <i>keyField2</i> • If there is a defaultSelectCondition and there are one or more fields listed in the EGL keyItems property, create the WHERE clause using both the default select condition and all the key items. For example: " where " + <i>recordDefaultSelectCondition</i> + " and <i>keyColumn1</i> = " + <i>keyField1</i> + " and <i>keyColumn2</i> = " + <i>keyField2</i> • If the WHERE clause includes the information from the defaultSelectCondition property, convert column information and operators to text literals and omit the semicolon from the host variable names, as is shown for the keyItems property. • In all other cases, omit the WHERE clause.

Table 163. VAGen migration text that causes EGL syntax or validation errors (continued)

VAGen migration text in EGL syntax	Problem and Solution
<p>EZE_UNKNOWN_SQL_WHERE</p> <p>where the VAGen I/O option is one of the following options:</p> <ul style="list-style-type: none"> • SETINQ • SETUPD 	<p>Problem: VisualAge Generator created a WHERE clause for the SQL SETINQ or SETUPD I/O options when default SQL was used and the Execution time statement build option was specified. 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 WHERE clause for the EGL I/O statement.</p> <p>Solution: Find the record named in the I/O statement and determine the correct WHERE clause from the defaultSelectCondition and keyItems properties in the EGL record. Create the WHERE clause for the prepare statement in the following way:</p> <ul style="list-style-type: none"> • If there is a defaultSelectCondition and no key items or more than one key item, create the WHERE clause from the defaultSelectCondition. For example: <code>" where " + recordDefaultSelectCondition</code> • If there is no defaultSelectCondition, but there is exactly one field listed in the EGL keyItems property, create the WHERE clause using the key item. For example: <code>" where keyColumn >= " + keyField</code> • If there is a defaultSelectCondition and there is exactly one field listed in the EGL keyItems property, create the WHERE clause using both the default select condition and the key item. For example: <code>" where " + recordDefaultSelectCondition + " and keyColumn = " + keyField</code> • If the WHERE clause includes the information from the defaultSelectCondition property, convert column information and operators to text literals and omit the semicolon from the host variable names, as is shown for the keyItems property. • In all other cases, omit the WHERE clause.

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.

If there are numerous errors in a file, it is best to resolve the errors in the following order:

- IWN.SYN messages for invalid syntax. These messages are typically the result of one of the following things:
 - An EGL reserved word used as the name of a Program, FormGroup, Form, or DataTable that cannot be renamed by the migration tool.
 - Deliberately invalid syntax used by the migration tool. See Appendix D, “Messages in the Problems view,” on page 427 for details on resolving these errors.
- IWN.VAL messages for a part that cannot be resolved or is ambiguous, including messages such as IWN.VAL.3260.e, IWN.VAL.6619.e, and IWN.VAL.6620.e.
- IWN.VAL warning messages for a statement that indicate there is a field with the same name as another Record, Form, or DataTable, including messages such as IWN.VAL.6570.w, IWN.VAL.6571.w, and IWN.VAL.6621.w.
- Other messages.

Note: EGL produces a maximum of 40 messages per file. Therefore, you might need to resolve some messages, save the file, and rebuild the workspace before you can see additional messages.

IWN.VAL messages for the .egl files

IWN.VAL.3012.e The same name *recordName* also appears as variable, parameter, use or constant declaration in Function, Program, or Library *programName*.

Explanation: VisualAge Generator tolerated specifying the same record name in the parameter list and in the Tables and Additional Records list. In this situation, VisualAge Generator ignored the record in the Tables and Additional Records list.

User response: Edit the program and remove the record declaration.

IWN.VAL.3260.e The type *partName* cannot be resolved.

Explanation: The specified part cannot be found. The meaning for migrated VAGen code varies based on the context as described under User response.

User response: The meaning varies based on the following contexts:

- *partName* is a record used as a type definition in the record declarations list for the program. Check to see if there is a record named *partName_level77Suffix*, where *_level77Suffix* is the preference you specified during Stage 2 of migration. VisualAge Generator tolerates working storage records that contain only level 77 items on the Tables and Additional Records list for a program. However, these level 77 items cannot be referenced in the program. Only level 77 items in the primary working storage record for the program can be referenced in the program. Because the record contained only level 77 items, the migration tool created only a level 77 record. If this is the case, edit the program and delete the declaration for the original record name. Do not add the corresponding level 77 record because none of its items were referenced in the VAGen program.
- *partName* is used as the help FormGroup. The VAGen program specified a help map group, but none of the maps that the program converses specifies a help map. Edit the program. Either remove the **use**

statement for the help FormGroup or add an **import** statement for the package containing the help FormGroup.

- *partName* is used as the **segmentRecord** or **parentRecord** property in a PSBRecord. VisualAge Generator does not issue an error message for a missing DL/I segment record in the PSB unless the segment is actually used in a program for DL/I I/O or to build a default SSA for a lower level segment. In EGL, all DL/I segment records referenced in the PSB must be defined. Review the PSBRecord and the logic for all the programs that use the PSB. If the PCB that specifies the missing DL/I segment record is only used as a place holder to pass through to a non-EGL program, consider changing the PCB to remove the hierarchy information. Alternatively, create or import the missing DLIsegment record definition.

IWN.VAL.4925.e The variable declaration *recordName* for *programName* could not be resolved.

Explanation: If the program uses DL/I, the specified *recordName* might be the name of a DL/I segment specified in the PSBRecord for the program. In VisualAge Generator, all DL/I segments specified in the PSB for the program are considered to be associates of the program. This ensures that any DL/I segment that might be used in a default SSA is available to the program. However, VisualAge Generator does not issue an error message for a missing DL/I segment record unless the segment is actually used for DL/I I/O or to build a default SSA for a lower level segment. If the program specifies a PSB, the migration tool automatically includes data declarations for all DL/I segments used as I/O objects or in the hierarchical path to an I/O object.

User response: Review the program to determine whether the DL/I segment record is needed. If the record is not used and is not required to build default SSAs for lower level segments, then edit the program and remove the record declaration.

IWN.VAL.4930.e The value of use declaration *FormGroupName* in program *programName* is invalid. You must use a FormGroup, DataTable, enumeration, or a library part.

Explanation: The specified FormGroupName is either missing or a reserved word.

User response: Migrate the missing FormGroup. If the FormGroup name is a reserved word, change the FormGroup name and all programs that reference the FormGroup.

IWN.VAL.5004.e The DataTable *tableName* is defined with *n1* column(s), but the contents are defined with *n2* column(s).

Explanation: The number of validly defined columns in the DataTable does not match the list of contents. One of the following situations might have occurred:

- One or more of the fields is defined with a type definition, but the type definition cannot be resolved.
- In VisualAge Generator, you used a comma as the decimal separator for numeric fields. EGL always uses the period. The migration tool converts the comma to decimal if you select the **Change decimal comma to decimal point** migration syntax preference.

User response: If one of the fields has a type definition that cannot be resolved, correct that problem first and rebuild the project containing the DataTable. If there are still problems and you used a comma as the decimal separator in VisualAge Generator, review the preference setting. Save the **import** statements from the file containing the DataTable, then migrate the table again using Single File Mode to correct the table contents, then add the **import** statements to the file.

IWN.VAL.5052.e *programName* - Records can only be compared to null.

Explanation: In general, this error occurs when there is a field in a record, form, or DataTable with the same name as a record. In VisualAge Generator, the name resolution is context sensitive so that the name resolves to the field. In EGL, the name resolves to the record, but a record is not permitted for this type of comparison. Note that all VAGen records migrate to EGL structured Records.

User response: If the problem is caused by name resolution, qualify the field name with the name of the record, form, or DataTable that contains the field. See the User response for message IWN.VAL.6621.w for information about how to confirm that name resolution is the problem and how to determine the possible qualifiers.

IWN.VAL.5085.e *programName* - The element *operandName* is not valid for use in the expression.

Explanation: In general, this error occurs when there is a field in a record, form, or DataTable with the same name as another record, form, or DataTable. In VisualAge Generator, the name resolution is context sensitive so that the name resolves to the field for an arithmetic expression. In EGL, the name resolves to a record, form, or DataTable, but these are not permitted for an arithmetic expression.

User response: If the problem is caused by name resolution, qualify *operandName* with the name of the record, form, or DataTable that contains the field. See

the User response for message IWN.VAL.6621.w for information about how to confirm that name resolution is the problem and how to determine the possible qualifiers.

IWN.VAL.5089.e *programName - operandName1 is not valid for compare to operandName2.*

Explanation: In general, this error occurs when there is a field in a record, form, or DataTable with the same name as another record, form, or DataTable. In VisualAge Generator, the name resolution is context sensitive so that the name resolves to the field for this type of comparison. In EGL, the name resolves to a record, form, or DataTable, but these are not permitted for this type of comparison. The name resolution problem can occur due a name conflict for either *operandName1* or *operandName2*.

User response: If the problem is caused by name resolution, qualify *operandName1* or *operandName2* with the name of the record, form, or DataTable that contains the field. See the User response for message IWN.VAL.6621.w for information about how to confirm that name resolution is the problem and how to determine the possible qualifiers.

IWN.VAL.5090.e *programName - The operand operandName in the in condition must be an item or a literal.*

Explanation: In general, this error occurs when there is a field in a record, form, or DataTable with the same name as another record, form, or DataTable. In VisualAge Generator, the name resolution is context sensitive so that the name resolves to the field for the FIND, RETR, IF *x* IN *array*, or WHILE *x* IN *array* statements. In EGL, the name resolves to a record, form, or DataTable, but these are not permitted for an if *x* in or while *x* in statement. Note that both the VAGen FIND and RETR statements migrate to an EGL if *x* in *structuredFieldArray* statement.

User response: If the problem is caused by name resolution, qualify *operandName* with the name of the record, form, or DataTable that contains the field. See the User response for message IWN.VAL.6621.w for information about how to confirm that name resolution is the problem and how to determine the possible qualifiers.

IWN.VAL.5093.e *programName - The operand operandName in the is/not condition must be a text form field.*

Explanation: In general, this error occurs when there is a field in a form with the same name as another record, form, or DataTable. In VisualAge Generator, the name resolution is context sensitive so that the name resolves to the map field for statements such as IF *x* IS CURSOR or IF *x* NOT DATA. In EGL, the name resolves to a record, form, or DataTable, but these are

not permitted for this type of if statement.

User response: If the problem is caused by name resolution, qualify *operandName* with the name of the form that contains the field. See the User response for message IWN.VAL.6621.w for information about how to confirm that name resolution is the problem and how to determine the possible qualifiers.

IWN.VAL.5094.e *programName - operandName is invalid for the current is/not expression. Variable Text Form fields, CHAR, MBCHAR, DBCHAR and UNICODE are valid types for use with the mnemonic blanks.*

Explanation: In general, this error occurs when there is a field in a record, form, or DataTable with the same name as another record, form, or DataTable. In VisualAge Generator, the name resolution is context sensitive so that the name resolves to the field for the IF *x* IS BLANKS statement. In EGL, the name resolves to a record, form, or DataTable, but these are not permitted for the if *x* is **blanks** statement. Note that the VAGen mnemonics BLANK, BLANKS, and NULLS migrate to the EGL mnemonic **blanks**. The VAGen mnemonic NULL migrates to the EGL mnemonic **blanks** if the comparison is for a map field and to **null** if the comparison is for an SQL field. See “Checking SQL and map items for NULL” on page 113 for details.

User response: If the problem is caused by name resolution, qualify *operandName* with the name of the record, form, or DataTable that contains the field. See the User response for message IWN.VAL.6621.w for information about how to confirm that name resolution is the problem and how to determine the possible qualifiers.

IWN.VAL.5095.e *programName - The operand operandName in the is/not condition is not valid for use with the mnemonic null.*

Explanation: In general, this error occurs when there is a field in an SQL record with the same name as another record, form, or DataTable. In VisualAge Generator, the name resolution is context sensitive so that the name resolves to the field for the IF *x* IS NULL statement. In EGL, the name resolves to a record, form, or DataTable, but these are not permitted for the if *x* is **null** statement. Note that the VAGen mnemonic NULL migrates to the EGL mnemonic **blanks** if the comparison is for a map field and to **null** if the comparison is for an SQL field. See “Checking SQL and map items for NULL” on page 113 for details.

User response: If the problem is caused by name resolution, qualify *operandName* with the name of the record or form that contains the field. See the User response for message IWN.VAL.6621.w for information about how to confirm that name resolution is the

problem and how to determine the possible qualifiers.

IWN.VAL.5101.e *mainFunctionName* - It is invalid to use the *xxxxx* system word in this statement location.

Explanation: The program 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**, **strLib**, **vgLib**, **vgVar**, **converseLib**, **converseVar**, **dliLib**, or **dliVar**, the following causes are the most likely:

- 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. A message on the VAGen View Messages list provides the correct definition of the implicit data item. Add the definition for the data item to the declarations section of the program. For information about a white paper that can help you create the implicit items before you run Stage 1 of migration, see “References” on page 16.

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.5143.e *programName* - *operandName* is invalid for the current is/not expression. **CHAR, MBCHAR, UINCODE and STRING are the only valid types for use with the mnemonic numeric.**

Explanation: In general, this error occurs when there is a field in a record, form, or DataTable with the same name as another record, form, or DataTable. In VisualAge Generator, the name resolution is context sensitive so that the name resolves to the field for the IF *x* IS NUMERIC statement. In EGL, the name resolves to a record, form, or DataTable, but these are not permitted for the if *x* is **numeric** statement.

User response: If the problem is caused by name resolution, qualify *operandName* with the name of the record, form, or DataTable that contains the field. See the User response for message IWN.VAL.6621.w for

information about how to confirm that name resolution is the problem and how to determine the possible qualifiers.

IWN.VAL.5168.e *xxxxx* is not valid for use within an is/not expression.

Explanation: The specified value is not valid. The meaning for migrated VAGen code varies based on the context as described under User response.

User response: The meaning varies based on the following contexts:

- If the value is EZE_DUPLICATE or EZE_NULL, the migration tool was not able to determine how to migrate the statement because the associated part was not included in the migration set. See Appendix D, “Messages in the Problems view,” on page 427 for details and solutions.

IWN.VAL.5177.e *programName* - The operand *operandName* in the is/not condition must be an item in an SQLRecord.

Explanation: In general, this error occurs when there is a field in an SQL record with the same name as another record, form, or DataTable. In VisualAge Generator, the name resolution is context sensitive so that the name resolves to the field for statements such as IF *x* IS TRUNC. In EGL, the name resolves to a record, form, or DataTable, but these are not permitted for statements such as if *x* is **trunc**.

User response: If the problem is caused by name resolution, qualify *operandName* with the name of the SQL record that contains the field. See the User response for message IWN.VAL.6621.w for information about how to confirm that name resolution is the problem and how to determine the possible qualifiers.

IWN.VAL.5196.e *programName* - Invalid for count *countName*. The for count must be an integer item or literal.

Explanation: In general, this error occurs when there is a field in a record, form, or DataTable with the same name as another record, form, or DataTable. In VisualAge Generator, the name resolution is context sensitive so that the name resolves to the field for the count in a MOVEA statement. In EGL, the name resolves to a record, form, or DataTable, but these are not permitted as the **for** count in a **move** statement.

User response: If the problem is caused by name resolution, qualify *countName* with the name of the record, form, or DataTable that contains the field. See the User response for message IWN.VAL.6621.w for information about how to confirm that name resolution is the problem and how to determine the possible qualifiers.

IWN.VAL.5313.e Property: **formSize**. The value for this property in *formName* is invalid. The value must be in the format [rows, columns], where rows and columns must be positive integers. The size of the form must be such that it fits in its corresponding output device.

Explanation: In general, this error occurs if you defined different floating area specifications (including using the default specification of the full screen size) for two or more devices that have the same physical size in VisualAge Generator. VisualAge Generator tolerated, but did not recommend, using different floating area specifications for devices of the same size. EGL only permits one floating area specification for a physical device size. The meaning for the message varies based on the situations described under "User response."

User response: The meaning varies based on the following situations:

- The VAGen map group explicitly specified different size floating areas for two or more devices with the same physical device size. In this case the migration tool converts both floating areas to the EGL **screenFloatingArea** or **printFloatingArea** properties. There is a message from the migration tool, as well as an EGL validation message in the Properties view explaining that there were duplicate definitions of the floating area for a particular device size. You corrected the problem by removing all except one of the floating area definitions. However, the one remaining floating area definition has margins that are so large that the *formName* specified in the error message cannot be contained within the floating area. Correct the problem by adjusting the margins for the floating area.
- The VAGen map group includes a floating map for multiple devices of the same size. The map group explicitly specifies a floating area for one of the devices and uses the default floating area size (full screen) for another device of the same physical device size. In this case, the migration tool only migrates the explicitly specified floating area size. However, the *formName* specified in the error message is intended to be used only with the physical device that used the VAGen default full screen specification. Correct the problem in one of the following ways:
 - Remove the floating area specification for the device size so that EGL uses the default floating area. Use this technique if you know that your organization no longer uses the physical device for which the floating area was explicitly specified.
 - Change the margins for the floating area specification so that the specified *formName* fits within the margins. Note that this will change the location of the form when it is displayed at

runtime for both the device that had the default specification and for the device for which the margins were changed. Use this technique if you know that your organization continues to use both physical devices.

- Change the definition for *formName* so that it fits within the existing margins. Use this technique if you know that your organization no longer uses the physical device which had the default floating area.

IWN.VAL.5340.e Property: **position**. The value for this property for field *fieldName* in *formName* is invalid. The value must be in the format [row, column], where row and column are positive integers.

Explanation: VisualAge Generator tolerates map fields at row=0, column=0. The field cannot specify any attributes and uses the same attributes as the previous field on the map. EGL requires that every field have an attribute byte. If a constant field at row=0, column=0 begins with a blank, the migration tool adjusts the field position by removing the first character and changing the position to row=1, column=1. This field is either a constant field (*fieldName* is *) that does not contain a blank as the first character or is a variable field.

User response: Use the EGL Editor to change the form. Adjust the length, **fieldLen**, **position**, **value**, and other properties of the field as necessary so the field fits on the form. Add presentation properties to achieve the same appearance for the field as it had in VisualAge Generator. After you have made these changes, you can use the EGL Form Editor to make future changes to the form.

IWN.VAL.6501.e The *sqlStatement* SQL I/O statement requires the *sqlClause* clause, which is missing.

Explanation: See message IWN.VAL.6506.e.

User response: See message IWN.VAL.6506.e.

IWN.VAL.6503.e The *xxxxx* SQL I/O statement has clauses that are out of order. *yyyyyy* must appear before the *zzzzz* clause.

Explanation: See message IWN.VAL.6506.e.

User response: See message IWN.VAL.6506.e.

IWN.VAL.6506.e The *xxxxx* SQL I/O statement allows only one *yyyyyy* clause.

Explanation: There are several situations in which this message can occur:

- In VisualAge Generator, SQL keywords are permitted as column names. In EGL, certain SQL keywords are not permitted.

- In VisualAge Generator, certain SQL clauses do not have to be parenthesized. In EGL, these clauses must be parenthesized. For example, a subselect and its related FROM, WHERE, GROUP BY, and ORDER BY clauses must be enclosed in parentheses.

User response: If you are not using subselects, see “SQL reserved words requiring special treatment” on page 254 for the list of SQL keywords and techniques you can use to resolve the problem with the column names.

If you are using subselects, add parentheses around the subselect and its related FROM, WHERE, GROUP BY, and ORDER BY clauses. For example, consider the following SQL statement:

```
with #sql{
  SELECT EMPNO, COUNT(*) WORKDEPT
  FROM EMPLOYEE T1
  WHERE WORKDEPT LIKE '%E%'
  GROUP BY WORKDEPT
  UNION ALL
  SELECT EMPNO
  FROM EMP_ACT
  WHERE PROJNO IN('MA2100', 'MA2112')
}
```

To fix the problem, change the SQL statement to add parentheses around the second SELECT, as in the following updated code:

```
with #sql{
  SELECT EMPNO, COUNT(*) WORKDEPT
  FROM EMPLOYEE T1
  WHERE WORKDEPT LIKE '%E%'
  GROUP BY WORKDEPT
  UNION ALL
  ( SELECT EMPNO
    FROM EMP_ACT
    WHERE PROJNO IN('MA2100', 'MA2112'))
}
```

IWN.VAL.6507.e The *xxxxx* SQL I/O statement does not allow the *yyyyy* clause.

Explanation: See message IWN.VAL.6506.e.

User response: See message IWN.VAL.6506.e.

IWN.VAL.6531.e The *sqlClause* SQL clause cannot be empty.

Explanation: See message IWN.VAL.6506.e.

User response: See message IWN.VAL.6506.e.

IWN.VAL.6541.e *programName* - The passing record identifier *operandName* must be a record variable.

Explanation: In general, this error occurs when there is a field in the I/O object for the function that has the same name as another record in the program. In VisualAge Generator, the name resolution is context sensitive so that *operandName* resolves to the record for

an XFER or DXFR statement. In EGL, the fields in the I/O object have a higher precedence than records used elsewhere in the program. In addition, the **show** statement (VAGen XFER with map or UI record) is an I/O statement in EGL.

Note that the migration tool converts the VAGen statements in the following way:

- XFER without a map or a UI record migrates to an EGL **transfer to transaction** statement.
- XFER with map and XFER with a UI record migrate to an EGL **show** statement.
- DXFR migrates to an EGL **transfer to program** statement.

User response: If the problem is caused by name resolution, specify the EGL keyword **this** as the qualifier (for example, **this.operandName**) so that the name resolves to the record. See the User response for message IWN.VAL.6621.w for information about how to confirm that name resolution is the problem and how to determine the possible qualifiers.

IWN.VAL.6570.w *programName* - The item access *fieldName* resolved to a record, form, or DataTable. There is an item called *fieldName* in record, form, or DataTable *containerName*.

Explanation: In general, this warning occurs when there is a field (*fieldName*) in a record, form, or DataTable (*containerName*) with the same name as another record, form, or DataTable. In VisualAge Generator, the name resolution is context sensitive so that the name resolves to a field, record, map, or DataTable, depending on the statement. In EGL, the name resolves to a function parameter item, function local storage item, record, form, or DataTable, which is valid, but not necessarily the same resolution as in VisualAge Generator.

User response: If the problem is caused by name resolution, qualify *operandName* with the name of the record, form, or DataTable that contains the field. See the User response for message IWN.VAL.6621.w for information about how to confirm that name resolution is the problem and how to determine the possible qualifiers.

IWN.VAL.6571.w *programName* - The item access *fieldName* resolved to an item in record, form, or DataTable *containerName*. There is a record, form, or DataTable called *fieldName* that is known to the program.

Explanation: In general, this warning occurs when there is a field (*fieldName*) in the I/O object (*containerName*) with the same name as another record, form, or DataTable in the program. In VisualAge Generator, the name resolution is context sensitive so that the name resolves to the field, record, map, or

DataTable, depending on the statement. In EGL, the name resolves to the field in the I/O object, which is valid, but not necessarily the same resolution as in VisualAge Generator.

User response: The meaning varies based on the context. Be sure to consider name resolution differences first, before considering other possibilities. See the User response for message IWN.VAL.6621.w for information about how to confirm that name resolution is the problem and how to determine the possible qualifiers.

IWN.VAL.6583.e *programName* - The subscript *subscriptName* in array reference *arrayName[subscriptName]* must be an integer item or integer literal.

Explanation: In general, this error occurs when there is a field in a record, form, or DataTable with the same name as another record, form, or DataTable. In VisualAge Generator, the name resolution is context sensitive so that the name resolves to the field for the subscript. In EGL, the name resolves to a record, form, or DataTable, but these are not permitted as a subscript.

User response: If the problem is caused by name resolution, qualify *subscriptName* with the name of the record, form, or DataTable that contains the field (for example, *containerName.fieldName*). See the User response for message IWN.VAL.6621.w for information about how to confirm that name resolution is the problem and how to determine the possible qualifiers.

IWN.VAL.6619.e *programName* - *variableName* cannot be resolved.

Explanation: The *variableName* might be a record name, form name, or a field in a record, form, DataTable, or function. The meaning for migrated VAGen code varies based on the context as described under User response.

User response: The meaning varies based on the following contexts:

- Determine if the VAGen program permits implicit item definitions and *variableName* is an unqualified field name. If so, validate the program in VisualAge Generator. The VAGen validation messages provide the definition of the implicit items used in the program. Edit the EGL program to add variable definitions for the implicit items using the VAGen validation messages as a guide to the necessary primitive type definition. For information about a white paper that can help you create the implicit items before you run Stage 1 of migration, see “References” on page 16.
 - Determine if the program uses DL/I and the specified *variableName* is the name of a DL/I segment specified in the PSBRecord for the program. If the message points to a DL/I I/O statement such as an **add** or **get**, the DL/I segment record is required to build the default SSA for a lower level segment.
-
- Review the program logic to determine whether the DL/I segment record is required to build a default SSA. If so, edit the program and add a declaration for the specified record.
 - Determine if the program uses DL/I and the qualifier is EZE_UNKNOWN_QUALIFIER. Over time, VisualAge Generator varied how the qualifier was determined for DL/I comparison value items. The migration tool was not able to determine what qualifier to use. For details, see the EZE_UNKNOWN_QUALIFIER information in Appendix D, “Messages in the Problems view,” on page 427, message IWN.MIG.0611.e (on page 417), and “DL/I I/O and comparison value items” on page 105.
 - Determine if the program uses DL/I and the specified *variableName* is **dliVar.name**. VisualAge Generator permits the use of EZE DL/I status words (EZEDL* words) even if the program does not specify a PSB. EGL only permits the use of the variables in **dliVar** if the program declares a PSBRecord. Edit the program to add a declaration for the PSBRecord or to remove the use of the variables in the **dliVar** library.
 - If you did not migrate all the parts in your migration set, the migration tool cannot include the appropriate **import** statements. If the part exists in the workspace, you might need to add an **import** statement to the file containing the error.
 - If *variableName* is a qualified name (for example, X.Z), the qualifier (X) is ambiguous. In VisualAge Generator, the only permitted qualifier is a record, table, or map name because a field name can only occur once within a record, table, or map. In EGL, a qualifier can also be a field within a record or DataTable because the same field name is permitted in multiple substructures within a record or DataTable. Determine if the program has a record, DataTable, or form name that is the same as a field within one of the records or DataTables. For example, you might have a form named X that contains a field named Z and a record named Y that contains a field named X, where field X has a subfield named A. In VAGen X.Z is a valid reference because X can only mean form X. In EGL, X.Z is ambiguous because X can be either form X or the field X within record Y. X is ambiguous and until X is resolved X.Z cannot be resolved even if Z only occurs within the form named X. Try using the EGL keyword **this** as a qualifier (for example, **this.X.Z**). Alternatively, either change the name of the form or change the name of the field (X) within the record (Y). A similar situation occurs if a DataTable has the same name as a field within a record.
 - If *variableName* is not qualified, see message IWN.VAL.6621.w for additional possibilities.
-

IWN.VAL.6620.e *programName* - The variable access *xxxxx* is ambiguous.

Explanation: The *variableName* might be a field in a record, form, DataTable, or function. The meaning for migrated VAGen code varies based on the context as described under User response.

User response: The meaning varies based on the following contexts:

- If *variableName* is an unqualified item name, determine if the problem occurs for a **call** statement or system function invocation. VisualAge Generator gives precedence to Level 77 items in the primary working storage record for a program if an unqualified item name is used for a **CALL** statement or **EZE** function invocation. EGL does not provide the same precedence. If the field specified by *variableName* exists in the Level 77 record that corresponds to the **inputRecord** program property, change the **call** statement or system function invocation to qualify the item with the Level 77 record name. However, you must be sure that all programs that invoke this function use the same Level 77 record.
- Check the function that has the error to determine if there is an EGL **show** statement (VAGen XFER with a map or XFER with a UI record). The VAGen XFER with map and XFER with UI Record statements are not considered to be I/O statements. The EGL **show** statement is an I/O statement. The presence of an EGL **show** statement changes the name resolution rules for *all* the statements in the function and can result in causing the name resolution to be ambiguous. Consider the following situations:
 - A field in the original I/O object for the function is named the same as a field in the form or VGUIRecord used in the EGL **show** statement. In VAGen, the name resolves to the field in the original I/O object. In EGL, both the field in the original I/O object and the field I/O object for the **show** statement are now in the same category for name resolution. If the name should resolve to a field in the original I/O object, then qualify the field name with the name of the original I/O object (for example, *recordName.xxxxx*).
 - A field in the original I/O object for the function is named the same as the form or the VGUIRecord used in the EGL **show** statement. In VAGen, the name resolves to the field in the original I/O object. In EGL, the field name and the form or VGUIRecord used in the **show** statement are in the same category for name resolution so the name is ambiguous. If the name should resolve to a field in the original I/O object, then qualify the field name with the name of the original I/O object (for example, *recordName.xxxxx*).
- If *variableName* is a function invocation, the name might conflict with the name of a field in a record, form, or DataTable. In VisualAge Generator, name

resolution is context sensitive so that it is possible for a nonshared field name to be the same as a function name. EGL does not permit this. Rename the function and all uses of the function. Alternatively, qualify the function invocation with the name of the package that contains the function. For example:

packageName.functionName(argumentList);

IWN.VAL.6621.w *programName* - The operand *operandName1* resolved to a form, record, or DataTable, and the operand *operandName2* resolved to a primitive item. In VAGen, both operands might have resolved to an item.

Explanation: In general, this error occurs when there is a field in a record, form, or DataTable with the same name as another record, form, or DataTable. In VisualAge Generator, the name resolution is context sensitive so that *operandName1* resolves to a field. In EGL, the name resolves to the record, form, or DataTable. See “Reference information for messages - name resolution and qualification rules” on page 450 for details on the differences in the resolution rules. A variety of EGL messages can result from this difference in name resolution, depending on the statement context. The name resolution problem can occur due a name conflict for either *operandName1* or *operandName2*.

User response: Specific messages and meanings vary depending on the statement context and whether the field has the same name as a record, form, or DataTable. In general, try the following techniques to resolve the problem:

- Check the function that has the error to determine if there is an EGL **show** statement (VAGen XFER with a map or XFER with a UI record). The VAGen XFER with map and XFER with UI Record statements are not considered to be I/O statements. The EGL **show** statement is an I/O statement. The presence of an EGL **show** statement changes the name resolution rules for *all* the statements in the function and can result in changing the resolution in either of the following ways:
 - A record, form or table in VisualAge Generator now resolves to a field within the I/O object for the **show** statement in EGL
 - A field in the original I/O object in VisualAge Generator now resolves to the form or VGUIRecord used in the **show** statement.
- Confirm that the problem is due to a field in a record, form, or DataTable having the same name as another record, form, or DataTable. In VisualAge Generator, follow these steps:
 1. Generate the program to ensure that it is a valid program.
 2. Use the Associates tool to find all the associates for the program.

3. Run the References tool against the Associates list. Specify the following information for the References tool:
 - Set **Search for** to a **Text** search.
 - Set the **Text** string to *operandName1*.
 - Set the **Search scope** to **All Parts in List**.
 4. The results from the References tool can help you determine whether *operandName1* is the name of an item and also the name of a record, map, or table. The results can also help you determine the possible qualifiers for *operandName1*. Repeat the process for *operandName2*.
 5. If there are multiple possible qualifiers, review the VAGen qualification rules in “Reference information for messages - name resolution and qualification rules” on page 450 to determine the correct qualifier. If the statement in error is a **call** statement or a function invocation, you might need to check the VAGen-generated COBOL program to determine how VisualAge Generator resolved the name.
- Change the EGL source code in one of the following ways:
 - If the name should resolve to a record or item variable in the function parameter list or the local storage, consider changing the name of the variable in the function parameter list or local storage. Do not change the record or item type definition. This technique limits the change to just the function in which the error is occurring. Be sure to change all references to the variable within the function.
 - If the name should resolve to a VAGen level 77 item, qualify the EGL field name with the name of the record, including the level 77 suffix that you specified during migration (for example, *recordName_level77Suffix.operandName1*). However, you must be sure that all programs that invoke this function use the same Level 77 record.
 - If the name should resolve to any other field, qualify the field name with the name of the record, form, or DataTable that contains the field (for example, *recordName.operandName1*).
 - If the name should resolve to a record or form and the statement is in a function in which the I/O object contains a field with the same name as the record or form, specify the EGL keyword **this** as the qualifier (for example, *this.operandName1*).

IWN.VAL.6650.e *programName* - *targetName* is a record, so the assignment source must be a record, or evaluate to CHA, HEX or MBCHAR.

Explanation: In general, this error occurs when there is a field in a record, form, or DataTable that has the same name as a record. In VisualAge Generator, the name resolution is context sensitive so that *targetName*

resolves to the field when the source is a numeric literal or numeric field. In EGL, the name resolves to the record.

User response: If the problem is caused by name resolution, qualify *targetName* with the name of the record, form, or DataTable that contains the field. See the User response for message IWN.VAL.6621.w for information about how to confirm that name resolution is the problem and how to determine the possible qualifiers.

IWN.VAL.6653.e *programName* - *primitiveType1* and *primitiveType2* are not compatible types in the expression *expressionText*.

Explanation: The meaning for migrated VAGen code varies based on the context as described under User response.

User response: The meaning varies based on the following contexts:

- If *primitiveType1* is HEX and the variable is the result for the **mathLib** functions **ceiling**, **floor**, **compareNum**, **precision**, or **round**, EGL does not support HEX as the result for these functions. Change your program logic to use a result variable with the primitive type specified by *primitiveType2*.
- This error can also occur when there is a field in a record, form, or DataTable that has the same name as another form or DataTable. In VisualAge Generator, the name resolution is context sensitive so that variable represented by *primitiveType1* resolves to the field. In EGL, the name resolves to the form or DataTable. If the problem is caused by name resolution, qualify the variable name with the name of the record, form, or DataTable that contains the field. See the User response for message IWN.VAL.6621.w for information about how to confirm that name resolution is the problem and how to determine the possible qualifiers.

IWN.VAL.6665.e *programName* - Invalid move source *operandName*. The source must be a record, form, item, literal, or constant.

Explanation: In general, this error occurs when there is a field in a record, form, or DataTable that has the same name as another DataTable. In VisualAge Generator, the name resolution is context sensitive so that *operandName* resolves to the field. In EGL, the name resolves to the DataTable.

User response: If the problem is caused by name resolution, qualify *operandName* with the name of the record, form, or DataTable that contains the field. See the User response for message IWN.VAL.6621.w for information about how to confirm that name resolution is the problem and how to determine the possible qualifiers.

IWN.VAL.6676.e *programName* - Invalid move target *targetName*[*subscriptName*]. The target must be an item array of compatible type with the source scalar.

Explanation: In general, this error occurs when there is a field in a record, form, or DataTable that has the same name as another record, form, or DataTable. In VisualAge Generator, the name resolution is context sensitive so that both the source and the target (*targetName*) resolve to fields. In EGL, the source or the target resolves to the record, form, or DataTable. A name resolution problem can occur due to a name conflict for either the source or the target field.

User response: If the problem is caused by name resolution, qualify the source or the target (*targetName*) with the name of the record, form, or DataTable that contains the field. See the User response for message IWN.VAL.6621.w for information about how to confirm that name resolution is the problem and how to determine the possible qualifiers.

IWN.VAL.6695.e *functionName* - The state XXXXX is not allowed for this item reference.

Explanation: The meaning for migrated VAGen code varies based on the context as described under User response.

User response: The meaning varies based on the following contexts:

- If state is PROTECT, SKIP, INVISIBLE, BLINK, INITIALATTRIBUTES, or a color, the field is on a print form. VisualAge Generator tolerates setting these attributes for printer forms. EGL does not. Modify the function to remove the statement. Alternatively, if the same function is used for both a text form and a print form, you must create a copy of the function for use with print forms or move the statement to the text form functions that invoke the current function.
- 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 field in a record or form. Define or migrate the missing record or form.

IWN.VAL.6697.e *programName* - The state *formFieldStateName* is not allowed for a record reference.

Explanation: In general, this error occurs when there is a field in a form that has the same name as a record. In VisualAge Generator, the name resolution is context sensitive so that the name resolves to the field on the map when the SET statement is for a property such as CURSOR, a color, and so on. In EGL, the name resolves to the record.

User response: If the problem is caused by name resolution, qualify the field specified in the **set** statement with the name of the form. See the User response for message IWN.VAL.6621.w for information about how to confirm that name resolution is the problem and how to determine the possible qualifiers.

IWN.VAL.6716.e *programName* - The argument *argumentName* cannot be passed to the **inOut** parameter *parameterName* of the function *functionName*. The types *type1* and *type2* are not reference compatible.

Explanation: *functionName* is the name of the invoked function. An argument passed to an **inOut** parameter is required to be reference compatible. Reference compatible means that the argument and parameter types must match exactly. This error can occur for either of the following reasons:

- VisualAge Generator does not support, but in some cases tolerates, different types for the argument and parameter.
- There is a field in a record, form, or DataTable that has the same name as another record, form, or DataTable.

The meaning for migrated VAGen code varies based on the context as described under User response.

User response: The meaning varies based on the following contexts:

- *functionName* is a system function and EGL resolves to a record or form. This might be a situation in which VisualAge Generator tolerates a record or form even when it does not make sense. For example, VisualAge Generator tolerates passing a record as the argument for an EZE math function and passing a form as the argument to a string function. Review the program logic to determine what is intended.
- *functionName* is a system function and EGL resolves to a field, but the field is not of the correct type. VisualAge Generator tolerates some argument types that are not compatible with the parameter types. EGL requires the types to be compatible. Add a substructure (or parent field) for the argument to provide a field that has the correct definition. For example, if the problem occurs for the third argument in **sysLib.startTransaction** and the third argument is an INT field, add a CHAR parent field to use as the third argument. Specifying the CHAR field as the parent field ensures that when the record is initialized, the substructure INT field is initialized to binary zeroes.
- The invoked function is a user function and EGL resolves to a field, but the field is not of the correct type. In this situation, you might be able to change the invoked function to use a function parameter type such as **number** that permits a wider variety of argument types and lengths. Alternatively, review the invoked function and all the places where it is

invoked. You might have to split the invoked function into multiple functions, each with a different definition of the parameter.

- Also consider whether there is a field with the same name as another record, form, or DataTable. See the User response for message IWN.VAL.6621.w for information about how to confirm that name resolution is the problem and how to determine the possible qualifiers.

IWN.VAL.6731.e *programName* - The argument *argumentName* cannot be passed to the in or out parameter *parameterName* of the function *functionName*. The types *type1* and *type2* are not assignment compatible.

Explanation: *functionName* is the name of the invoked function. An argument passed to an in or out parameter is required to be assignment compatible. VisualAge Generator tolerates some types that are not assignment compatible. EGL does not permit incompatible types. The meaning for migrated VAGen code varies based on the context as described under User response.

User response: See the User response for message IWN.VAL.6716.e.

IWN.VAL.6736.e *programName* - Invalid expression *argumentName* for the convert operation. The expression must evaluate to a record or have a primitive type besides blob or clob.

Explanation: *functionName* is the name of the invoked function. The argument must be a fixed-length record or primitive type. The migration tool converts all VAGen records to structured Records. VisualAge Generator tolerates some types that are not assignment compatible. EGL does not permit incompatible types. The meaning for migrated VAGen code varies based on the context as described under User response.

User response: See the User response for message IWN.VAL.6716.e.

IWN.VAL.6741.e *programName* - The argument *argumentName* in function invocation *functionName* is invalid. The argument must be of primitive type char, mbchar, dbchar, hex, num, or unicode.

Explanation: *functionName* is the name of the invoked function. The argument is limited to the listed primitive types. VisualAge Generator tolerates some types that are not assignment compatible. For example, VisualAge Generator tolerates passing records as arguments to string functions, even though this is not valid based on the VAGen documentation. EGL does not permit incompatible types. The meaning for migrated VAGen

code varies based on the context as described under User response.

User response: See the User response for message IWN.VAL.6716.e.

IWN.VAL.6742.e *programName* - The argument *argumentName* in function invocation *functionName* is invalid. The argument must be of primitive type char, dbchar, hex, num, bin, int, smallint, bigint, pacf, money, or decimal.

Explanation: *functionName* is the name of the invoked function. The argument is limited to the listed primitive types. VisualAge Generator tolerates some types that are not assignment compatible. EGL does not permit incompatible types. The meaning for migrated VAGen code varies based on the context as described under User response.

User response: See the User response for message IWN.VAL.6716.e.

IWN.VAL.6743.e *programName* - The argument *argumentName* cannot be passed to the loose parameter *parameterName* of the function *functionName*. It must be of primitive type *type1*.

Explanation: *functionName* is the name of the invoked function. The argument is limited to the listed primitive types. VisualAge Generator tolerates some types that are not assignment compatible. EGL does not permit incompatible types. The meaning for migrated VAGen code varies based on the context as described under User response.

User response: If the *functionName* is `strLib.byteLen()`, you might be able to substitute `sysLib.bytes()`, which supports all primitive types. For additional possibilities, see the User response for message IWN.VAL.6716.e.

IWN.VAL.6744.e *programName* - The argument *argumentName* cannot be passed to the loose parameter *parameterName* of the function *functionName*. It must have a numeric primitive type.

Explanation: *functionName* is the name of the invoked function. The argument is limited to the numeric primitive types. VisualAge Generator tolerates some types that are not assignment compatible. For example, VisualAge Generator tolerates passing forms and records as arguments to math functions and to index and length arguments of string functions. EGL does not permit incompatible types. The meaning for migrated VAGen code varies based on the context as described under User response.

User response: See the User response for message IWN.VAL.6716.e.

IWN.VAL.6746.e The function reference *functionName* cannot be resolved.

Explanation: The meaning for migrated VAGen code varies based on the context as described under "User response."

User response: The meaning varies based on the following contexts:

- The message is issued as an error (suffix "e"). The specified function cannot be found. If you did not include all the parts in your migration set, the migration tool might have incorrectly "guessed" that an edit routine was a function and migrated to the **validatorFunction** property. Find the part specified by *functionName*. If the part is a DataTable, edit the file that is in error and change the **validatorFunction** property to **validatorDataTable**. If the part is a function, you might need to add an **import** statement to the file that is in error to point to the package that contains the function.
- The message is issued as a warning (suffix "w"). In VisualAge Generator, if a function is specified as the edit routine of a DataItem part, an error only occurs if the DataItem is used as a shared data item in a UI record. In EGL, the **validatorFunction** should be available, regardless of whether (or where) the DataItem part is used as a type definition. In this situation, if you want to eliminate the warning message, either remove the **validatorFunction** property from the DataItem part or add an **import** statement to the file that is in error to point to the package that contains the function.

IWN.VAL.6751.e *programName* - The target for a function invocation must be a function or a delegate.

Explanation: The name of the function conflicts with the name of a field in a record, form, or DataTable. In VisualAge Generator, name resolution is context sensitive so that it is possible for a nonshared field name to be the same as a function name. EGL does not permit this.

User response: Rename the function and all uses of the function. Alternatively, qualify the function invocation with the name of the package that contains the function. For example:

```
packageName.functionName(argumentList);
```

IWN.VAL.7553.e *functionName* - Argument *n* for *systemFunctionName* must be a string item, string constant or a string literal.

Explanation: *functionName* is the name of the invoked function. The argument is limited to the listed types. VisualAge Generator tolerates some types that are not assignment compatible. EGL does not permit incompatible types. The meaning for migrated VAGen

code varies based on the context as described under User response.

User response: See the User response for message IWN.VAL.6716.e.

IWN.VAL.7696.e The DataTable *DataTableName* cannot be used as a validatorDataTable for item *type1*. For tables of type **MatchValidTable** and **MatchInvalidTable**, the type of the first column must match the type of the item. The types *type1* and *type2* do not match.

Explanation: VisualAge Generator tolerates the use of a table as an edit routine for a data item even if the type of the data item does not match the type of the first column in the table. EGL does not permit this situation.

User response: Remove the **validatorDataTable** property from the DataItem part or change the type of the first column in the table.

IWN.VAL.7740.e *programName* - *variableName* is read-only and cannot be assigned to.

Explanation: *variableName* is one of the EGL system variables such as **sysVar.userID**. The migration tool always qualifies the EGL system variables with the EGL library name. If *variableName* is not qualified and is in uppercase, it might be an implicit item that is created automatically in VisualAge Generator. EGL does not permit implicit items.

User response: Validate the program in VisualAge Generator to determine whether it allows implicit items and if *variableName* is one of the implicit data items created for the program. If so, modify the EGL program to add a variable declaration for *variableName*. Use the information in the VAGen validation messages to determine the correct definition of the item in EGL. For information about a white paper that can help you create the implicit items before you run Stage 1 of migration, see "References" on page 16.

IWN.VAL.7755.w Unreachable code

Explanation: A statement such as **exit program**, **exit stack**, or **return** is followed by one or more statements. The statements after **exit program**, **exit stack**, or **return** can never be executed.

User response: No changes are required. However, if you want to remove the warning, comment out or remove the unreachable code.

IWN.VAL.7757.e The number of elements in the initializer array must be no greater than the number of occurs of item *fieldName*. *number1* elements found for occurs size *number2*.

Explanation: For a VGUI record, the list of values for a **submit** or **submitBypass** button array is too big for the size of the array. VisualAge Generator ignores any extra values during generation. EGL requires that the array size match the number of values in the list.

User response: Edit the VGUI record and remove the additional values.

IWN.VAL.7789.e The *propertyName* item *fieldName* cannot be a multiply occurring item.

Explanation: For UI records, VisualAge Generator tolerates the Occurrences item as an array, but treats it as though it was not an array. EGL does not permit the **numElementsItem** property to specify a multiply occurring item.

User response: Edit the VGUI record and change the specified *fieldName* so that it not an array. You might have to move the field outside a multiply occurring parent field.

IWN.VAL.7868.e *programName* - *stateName* is not a valid state for a DL/I segment record.

Explanation: For a DL/I record, the *stateName* in an **if** or **while** statement is not valid. VisualAge Generator tolerates some *stateNames* that do not make sense for DL/I (for example, FMT). EGL does not permit this *stateName*.

User response: Review your program logic to determine what you really intended the program to do. Change or comment out the statement.

IWN.VAL messages for the .eglbld file

IWN.VAL.3999.e (first of ten alternatives) XML
Validation Error - Attribute "xxxxx" was already specified for element "yyyyyy".

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. Because this error ends the processing of the .eglbld file, there might be errors for unresolved parts that are in the .eglbld file, but which are defined after the point of the error.

User response: Edit the current .eglbld file using the Text Editor. Change the control part 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.VAL.3999.e (second of ten 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 that EGL validation displays an error message in the Problems view to remind you to resolve the problem.

User response: Review the EGL online help 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.

IWN.VAL.3999.e (third of ten alternatives) XML
Validation Error - Attribute xxxxx with value yyyyyy must have a value from the list zzzzzz.

Explanation: Not all VAGen values have a corresponding replacement in EGL.

User response: For details on how migration converts the values, see the tables for linkage table and resource association in "Control parts" on page 352. For information on the choices available in EGL, refer to the EGL documentation.

IWN.VAL.3999.e (fourth of ten alternatives) XML
Validation Error - Attribute xxxxx must be declared for element type "yyyyyy"

Explanation: Not all VAGen values have a corresponding replacement in EGL. In some cases,

combinations of values that are valid in VisualAge Generator are not valid in EGL. For example:

- In a linkage options part, the **library** attribute is not valid for a **localCall** entry. In EGL, calls from generated Java code to a native C++ DLL are considered to be remote calls even when running on the same machine.
- In a linkage options part, the **remotePgmType** attribute is not valid for a **localCall** entry. In VAGen, the **remoteAppType** option is ignored for local calls. In EGL, XML validation is more restrictive—only attributes that are meaningful are permitted.

User response: For details on how migration converts the values, see the tables for linkage table and resource association in "Control parts" on page 352. For information on the choices available in EGL, refer to the EGL documentation.

IWN.VAL.3999.e (fifth of ten alternatives) XML
Validation Error - Element type "xxx" must be followed by either attribute specifications, ">" or "/>".

Explanation: In a resource associations part, the target environment information is invalid. This can occur when the original VAGen resource association entry contains /system=xxx*yyy or /system=xxx*. The wildcard * is valid in VAGen, but not in EGL. The migration tool does not attempt to convert the entry to all possible EGL target systems that might be valid.

User response: Change the entry to be a valid EGL runtime environment. Repeat the entry as many times as necessary for all of your EGL runtime environments that apply. For example, if the original VAGen entry is /system=mvs*, repeat the entry for the EGL environments ZOSBATCH and ZOSCICS. Because the XML parser stops processing at the first resource association entry that contains an * in the target environment, you might have to resolve several errors before the entire file can be parsed.

IWN.VAL.3999.e (sixth of ten alternatives) XML
Validation Error - The content of elements must consist of well-formed character data or markup.

Explanation: In a resource associations part, the target environment information is invalid. This can occur when the original VAGen resource association entry contains /system=*xxx or /system=*. The wildcard * is valid in VAGen, but not in EGL. The migration tool does not attempt to convert the entry to all possible EGL target systems that might be valid.

User response: Change the entry to be a valid EGL runtime environment. Repeat the entry as many times as necessary for all of your EGL runtime environments

that apply. For example, if the original VAGen entry is *cics, the only valid EGL runtime environment is zoscics. Because the XML parser stops processing at the first resource association entry that contains an * in the target environment, you might have to resolve several errors before the entire file can be parsed.

IWN.VAL.3999.e (seventh of ten alternatives) XML Validation Error - Attribute "xxxxx" is required and must be specified for element type "yyyyy".

Explanation: In VisualAge Generator, some attributes were not required. In EGL, the attribute specified by xxxxx is required when the element specified by yyyyy is defined

User response: Use the Text Editor to add the required attribute. For information on the choices available in EGL, refer to the EGL documentation.

IWN.VAL.3999.e (eighth of ten alternatives) XML Validation Error - Element type elementType must be declared.

Explanation: Not all VAGen values have a corresponding replacement in EGL. In some cases, combinations of values that are valid in VisualAge Generator are not valid in EGL. For example, in a resource association file, the value BTRIEVE was valid as a file type for the AIX environment. In EGL, the value BTRIEVE is not valid..

User response: Use the Text Editor to correct the part. For details on how migration converts the values, see the tables for linkage table and resource association in "Control parts" on page 352. For information on the choices available in EGL, refer to the EGL documentation.

IWN.VAL.3999.e (ninth of ten alternatives) XML Validation Error - The string "--" is not permitted within comments.

Explanation: XML does not permit a string of hyphens in a comment. This problem generally occurs when the VAGen control part contained comments and used lines of hyphens to separate sections of a block comment.

User response: Use the Text Editor to correct the part. Replace the string of "--" with a string of "==".

IWN.VAL.3999.e (tenth of ten alternatives) XML Validation Error - Invalid byte 2 of 3-byte encodingSequence sequence.

Explanation: One or more of the build parts in the file contains national language characters that are not valid for the encoding sequence specified for the file. The default encoding sequence is UTF-8.

User response: To change the default encoding

sequence before single file migration or Stage 2 migration, from the EGL developer environment, click **Windows -> Preferences -> EGL**. Use the drop-down list to select a value for the **Encoding** preference that supports your national language characters. To change the default encoding sequence after migration, use the system editor to change the encoding information in the file. You cannot open the file with the EGL Build Parts Editor until the encoding information is corrected.

IWN.VAL.4300.e The part named partName could not be resolved or did not resolve to one of the following types: partTypeList

Explanation: The meaning for migrated VAGen code varies based on the context as described under User response.

User response: The meaning varies based on the following contexts:

- The specified control part does not exist. Create the part or remove the reference to it.
- 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. If the specified control part is in a different project, update the EGL Build Path in the properties of the current project 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.
- 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.VAL.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. Edit the current .eglbld file so that there is only one specification for xxxxx in the control part. Assuming that you have **Build Automatically** selected, when you save the .eglbld file, the message 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.VAL.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.

Java messages for JSPs

Invalid character constant

Explanation: This can occur in the following situations:

- For the `xxxxxBean.java` generated for a VGUI record, the `setFillCharacter` method uses a value of `'nullFill'`. In VisualAge Generator, a shared data item has two sets of properties: one for maps and one for UI records. In EGL, a `DataItem` part has one set of properties. The migration tool merges the two sets of properties, giving precedence to the UI properties. However, the migration tool cannot predict whether the data item part is used in a map, a UI record, or both. The data item did not specify a UI default fill character so the migration tool used the map default

character, which was null. When the shared data item was used in a UI record, VisualAge Generator used blank as the default fill character because no UI fill character was specified. When the data item part is used as a type definition in a VGUI record, EGL generates using the null fill character, which is not valid.

User response: If the problem is due to the `setFillCharacter` method with a value of `"nullFill"`, edit the VGUI record and add an override property for the item to set the `fillCharacter` property to `""`. This preserves `nullFill` as the default fill character for use on EGL forms.

Reference information for messages - name resolution and qualification rules

The VisualAge Generator and EGL name resolution and qualification rules differ due to enhancements in the EGL. In most cases, the differences in the VAGen and EGL name qualification and name resolution rules do not pose a problem. However, in the case where a field in a record, form, or `DataTable` has the same name as another record, form, or `DataTable`, VisualAge Generator and EGL might resolve the names differently. The following sections review the rules for the two products and then describe the messages that EGL validation might display in the Problems view when this situation occurs.

VisualAge Generator name resolution and qualification rules

VisualAge Generator does not permit two parts to have the same name. However, nonshared items defined in the following places might have the same name as a record, map, or table:

- A nonshared item in a record or table.
- A field on a map (which is always treated as a nonshared item).
- A nonshared field in function local storage or parameter list.

In addition, multiple items might have the same name. For example, the same item name can be used in the following places:

- Function local storage or function parameter list.
- Level 77 item in the primary working storage record for the program.
- Item in the called parameter list for the program.
- Item in an I/O object for a function.
- Item in any other record, map, or table for the program.

In the situations in which the same name is defined in multiple places within a program, the VAGen name resolution is context sensitive. For example:

- VAGen name resolution uses an item in the following situations:
 - An assignment statement or a MOVE statement in which one of the following occurs:
 - The source is a literal, another item, or, in the case of an assignment statement, an arithmetic expression.

- The target is another item.
- A subscript in any statement.
- The source, target, or for count in a MOVEA statement.
- An IF, WHILE, or TEST statement in which the right-hand side of the comparison can only be used with an item (such as BLANKS, NUMERIC, NULLS, TRUNC) or in which the other side of the comparison is a literal or another item.
- A SET statement in which the value being set can only be used with an item (such as NULL or CURSOR).
- A FIND or RETR statement.
- If the item name is not qualified and there are multiple items with the same name, VisualAge Generator resolves the name based on categories VG1 through VG3 as described later in this section.
- VAGen name resolution uses a record in the following situations:
 - A MOVE statement in which the source or target is another record or map.
 - An IF, WHILE, or TEST statement in which the right-hand side of the comparison can only be used with a record (such as DUP or ERR).
 - A SET statement in which the value being set can only be used with a record (such as EMPTY or SCAN).
 - A DXFR statement
 - An XFER or XFER with UI record statement.
- VAGen name resolution uses a map in the following situations:
 - A MOVE statement in which the source or target is another record or map.
 - A SET statement in which the value being set can only be used with a map (such as EMPTY, CLEAR, or PAGE).
 - An XFER with map statement.
- In cases in which the statement context does not determine the part type that is expected, the precedence varies based on the statement, whether the statement is in an I/O function, and whether an item and a record, map, or table have the same name. For example:
 - For a CALL statement or a system function invocation in which the argument names are all unique, there is no problem. In the case where a field has the same name as another record, map, or table, the observed VAGen name resolution varies based on the use of level 77 items, called parameters, records in the tables and additional records list, the use of records or forms as I/O objects, and so on.

Note: Based on tests using VisualAge Generator 4.5 Fix Pack 5, no consistent pattern was determined for CALL statements or function invocations. In the event of a name resolution conflict for a CALL statement or system function invocation, generating the COBOL program might be the quickest method of determining how VisualAge Generator resolved the name.

- For the statement IF *x* IS MODIFIED, *x* might be a map or a field on another map. In this case, the map is used. To reference the field in the other map, you must qualify the name as *mapName.x*. Similarly, if *x* is a UI record or a field in another UI record, the UI record is used. To reference the field in the other UI record, you must qualify the name as *UIRecord.x*.

When VisualAge Generator expects to find an item, the following name qualification rules are used to determine where the item is located.

- If the item name is qualified, the qualifier can be a record, map, or table. Multiple levels of qualification are not necessary (and are not supported) because an item name must be unique within a record, map, or table.
- If an item name is not qualified, data items are checked to determine the qualifier in the following order:
 - Category VG1 - Item names in the local storage and parameter list for this function.
 - Category VG2 - The following records and maps that are specific to this function:
 - I/O object and its items.
 - Records in the function parameter list and their items.
 - Records in the function local storage list and their items.
 - If the name is not unique in this category, it must be qualified.
 - Category VG3 - The following records, maps, and tables in the program:
 - Program working storage record and its items.
 - Table and Additional Records list and their items.
 - Records and maps in the called parameter list and their items.
 - I/O objects of other functions in the program and their items.
 - If the name is not unique in this category, it must be qualified.
 - If the program allows implicit data items, VisualAge Generator creates an implicit definition based on usage.

For a CALL statement or function invocation, records tend to take precedence over fields (because a record name cannot be qualified) and level 77 items tend to take precedence over other fields (because long ago, only level 77 items could be used in a CALL statement). However, based on tests using VisualAge Generator 4.5 Fix Pack 5, no consistent pattern was determined for CALL statements or function invocations.

EGL name resolution and qualification rules

In EGL, the name resolution rules differ from the VisualAge Generator rules due to the following EGL enhancements:

- The same field name can be included multiple times in a record under different substructures. Multiple levels of qualification are supported and in some cases are required. The qualifier can be a field within the record (for example, MYRECORD.SHIPPING.ADDRESS or just SHIPPING.ADDRESS).
- A function can have multiple I/O statements and therefore, multiple I/O objects.
- The **show** statement (VAGen XFER with map or UI record) is considered to be an I/O statement.
- Fields (item variables) can be declared at the program level.
- EGL does not have level 77 items. The migration tool includes the level 77 items in a separate basic record that is declared in the program.
- New EGL parts such as libraries.
- The EGL keyword **this**, which can be used to indicate that you want to use a record variable declared at the program level or a form specified in the **use** forms statement for the program rather than a name in Categories EGL1 and EGL2 as described later in this section.

When EGL finds a name, the following name resolution and qualification rules are used to determine where the field is located.

- If the program does not set the **allowUnqualifiedItemReferences** property to yes, all field references must be qualified. If the field is in a substructure, the field must be fully qualified. For example, if **allowUnqualifiedItemReferences** is not set to yes and CUSTOMER_RECORD contains the field NAME which is the parent of LASTNAME, then CUSTOMER_RECORD.NAME.LASTNAME is valid, but CUSTOMER_RECORD.LASTNAME is not valid.
- The qualifier can be a record, form, DataTable, or another field within a structured record or DataTable. Multiple levels of qualification are permitted and in some cases are required.
- If a field is not qualified or is partially qualified, EGL checks the fields to determine the qualifier in the following order:
 - Category EGL1 - The following variables declared at the function level:
 - Item variable names declared in the local storage and parameter list for this function.
 - Record variable names declared in the local storage and parameter list for this function.
 - Names must be unique in this category.
 - Category EGL2 - The following I/O objects and other fields that are specific to this function:
 - I/O objects and their fields. In EGL, a function can have multiple I/O statements. In addition, in EGL the **show** statement (VAGen XFER with map or XFER with UI record) is considered to be an I/O statement.
 - The fields in record variables in the function parameter list.
 - The fields in record variables in the function local storage list.
 - If the name is not unique in this category, it must be qualified.
 - Category EGL3 - The following variables declared at the program level:
 - Record variable names from the record declarations for the program. The record declarations list includes records specified by the **inputRecord** property or used in I/O statements in any function of the program.
 - Item variable names declared at the program level.
 - Item variable and record variable names listed in the parameter list for the program.
 - If the name is not unique in this category, it must be qualified.
 - Category EGL4 - A form name from the **use** forms statement for the program. This list includes forms specified by the **inputForm** property, program parameter list, or used in I/O statements in any function of the program. If the **use** form statement only specifies a FormGroup, then all forms within the FormGroup are considered.
 - Category EGL5 - A DataTable name that is specified in the **use** declarations for the program.
 - Category EGL6 - The following fields used anywhere in the program:
 - Fields from record variables, forms, and DataTables in categories EGL3 through EGL5.
 - If the **use** form statement only specifies a FormGroup, then all fields on all forms within the FormGroup are considered.
 - If the name is not unique in this category, it must be qualified
 - Category EGL7 - Fields in a user library specified in the **use** declarations for the program. (This category does not affect migrated VAGen programs because nothing migrates to an EGL library.)

- Category EGL8 - Fields in a system library. (The migration tool always qualifies fields migrated from the VAGen EZE words with the EGL system library name.)
- Implicit definitions are not permitted in EGL.

The EGL name resolution and qualification rules are consistent -- they are not affected by the type of statement or function being invoked.

Validation messages due to differences in name resolution and qualification rules

In most cases, the differences in the VisualAge Generator and EGL name resolution and qualification rules do not pose a problem. However, in the case where a field in a record, form, or DataTable has the same name as another record, form, or DataTable, VisualAge Generator and EGL might resolve the names differently. This can result in the following situations:

- Invalid EGL statements which are detected by normal EGL validation (for example, trying to set the color for a record rather than a field on a form). In these cases, EGL validation displays an error message in the Problems view.
- Special EGL warning messages that are produced by validation if you are using VisualAge Generator compatibility mode. These warning messages indicate that the name resolution *might* be different between VisualAge Generator and EGL.
- No EGL message is issued for situations in which there is a level 77 item or called parameter that received precedence for a CALL statement or function invocation in VisualAge Generator, and which results in a different resolution in EGL.

The following are examples of situations in which name resolution might change between VAGen and EGL if there are things that have the same name.

Example 1

Resolution changes from field in one record to another record:

ProgramA:

```
Tables and Additional Records List
RECORDA - a record that contains field RECORDZ defined as CHAR
RECORDZ - a record that contains other fields
FunctionA:
  MOVE "abc" to RECORDZ;
/* VAGen-resolves to field RECORDA.RECORDZ, */
/* based on statement context; */
/* record is invalid if source is a literal */
/* EGL -resolves to record RECORDZ, */
/* based on Category EGL3; a record is valid */
/* -message IWN.VAL.6621.w is issued */
```

Example 2

Resolution changes from a field in the original I/O object to being ambiguous due to the name of a form used in **show** statement:

ProgramB:

```
FORMF is a form in the program FormGroup for the program
FunctionB:
  INQUIRY RECORDB /* RECORDB contains field FORMF defined as BIN */
  XFER PGMBX , FORMF; /* converts to EGL show statement */
  FORMF = 123; /* VAGen-resolves to field RECORDB.FORMF */
/* based on statement context; */
/* form is invalid if source is a literal */
/* EGL -cannot resolve between field or form FORMF; */
/* based on Category EGL2; form is not valid */
```

```

/*      -FORMF is now an I/O object for show      */
/*      -a form is invalid;                        */
/*      -message IWN.VAL.6620.e is issued          */

```

Example 3

Resolution changes from a record to a field in a form used in **show** statement

ProgramC:

```

FORMF is a form in the FormGroup for the program and contains
field RECORDZ Tables and Additional Records List
RECORDZ - a record that contains some fields
FunctionC:
  XFER PGMCY , FORMF; /* converts to EGL show statement      */
  CALL PGMCX RECORDZ; /* VAGen-resolves to record RECORDZ    */
                        /* based on statement context;        */
                        /* record receives precedence on a CALL */
                        /* EGL -resolves to field FORMF.RECORDZ; */
                        /* based on Category EGL2              */
                        /* -FORMF is now an I/O object for show */
                        /* -message IWN.VAL.6571.w is issued    */

```

Example 4

Resolution changes from a field in the original I/O object to being ambiguous due to a field in a form used in **show** statement.

ProgramD:

```

FORMF is a form in the FormGroup for the program and
contains a field named ITEMDD defined as BIN
FunctionD:
  INQUIRY RECORDD /* RECORDD contains field ITEMDD defined as BIN */
  XFER PGMDX , FORMF; /* converts to EGL show statement          */
  ITEMDD = 123; /* VAGen-resolves to field RECORDD.ITEMDD          */
                /* based on Category VG2                          */
                /* - RECORDD is I/O object                        */
                /* - FORMF is not an I/O object                    */
                /* EGL -cannot resolve between 2 fields ITEMDD    */
                /* based on Category EGL2                          */
                /* -FORMF is now an I/O object for show           */
                /* -message IWN.VAL.6620.e is issued              */

```

Appendix F. APARs required for VisualAge Generator

There are currently no known APARs that are required beyond what is included in VisualAge Generator Developer V4.5 Fix Pack 5.

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.

The migration database requires DB2 Version 8.1 with Fix Pack 15 or DB2 Version 8.2 with Fix Pack 8. These two fix pack levels are equivalent. DB2 Version 9.x is supported for Stage 1 on Smalltalk and Stages 2 and 3. DB2 Version 9.x is not supported for Stage 1 on Java.

Using DB2 on Windows XP

The migration tool has the following requirements:

- 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.

DB2 authority requirements

The db2 user ID that is used when running the migration tool must be granted select, insert, update, delete, and control authority for all the migration tables, views and indexes. Control authority is required by the RUNSTATs commands issued during Stage 1 to improve the performance for the remaining migration steps.

Creating the migration database

After you install or upgrade DB2 to the appropriate level, create the migration database by following these steps:

1. Make sure that DB2 and any other applications that use it are shut down. For example, shut down VisualAge Generator and the EGL development environment.
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 created 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 also 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 124. If you are migrating from Smalltalk, see “Setting Stage 1 preferences” on page 148.

Resetting the migration database for Stage 1

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, follow these steps:

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, follow these steps:
 1. Determine the migration set ID that you need to delete from the migration database by following these steps:

- 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 value in the CONFIGPLANID column corresponding to 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

The migration tool provides better performance if you use a local DB2 database. However, if you decide to use a remote database, this section provides information that can be helpful. 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, follow these steps:

1. Bring up a DB2 Command Prompt window on Windows .
2. To catalog the node, enter the following command all on one line:

```
db2 catalog tcpip node nodeName remote [ hostName | ipAddress ]
server [ svcname | 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 is 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, follow these steps:

1. Bring up a DB2 Command Prompt window on Windows.
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 a database alias for the SAMPLE database, 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 command:
db2 connect to VGMIG

To assist in determining if the Stage 1 migration tool ran correctly, run the following .bat file that is located in your *VAGen-installation\ide\vgmigration* directory for Java or your *VAGen-installation* directory for Smalltalk:

checkStage1.bat

The .bat file runs several queries:

- List of the migration plan names in the database
- Total number of parts in the database. This number includes any dummy map group parts that were created by the Stage 1 migration tool.
- Other queries to check the validity of the External Source Format and parts placement into EGL files.

The results of the first two queries should be greater than 0. The results of the other queries should be 0. If your results differ, then there was a problem during Stage 1 migration and you should contact IBM Support.

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, substr(vgesfsource,1,n) from migschema.vgpart
```

n specifies 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, substr(vgesfsource,1,n) from migschema.vgpart
      where vgesfsource not like ':%'
```

n specifies the number of characters you want to display.

Determining the number of parts in the migration database

To determine the total number of parts in the migration database:

```
db2 select count(*) from migschema.vgpart
```

To determine the number of parts of a specific part type in the migration database:

```
db2 select count(*) from migschema.vgpart where vgparttype = nnnnnnn
```

where *nnnnnnnn* varies based on the part type as shown in the following table.

Table 164. Part type values

Part type	vgparttype values
Function	65536
Link Edit	131072
Bind Control	262144
Resource Associations	524288

Table 164. Part type values (continued)

Part type	vgparttype values
Linkage Table	1048576
Generation Options	2097152
Table	8388608
Record	33554432
PSB	67108864
Map	268435456
Map Group	536870912
Data Item	1073741824
Program	2147483648

Determining the number of parts migrated during Stage 2

To determine how many new parts were done by the Stage 2 migration, run this query before and after the migration:

```
select count(*) from migschema.vgpart where is_migrated = 'Y'
```

Reviewing the EGL file names

As an alternative or supplement to reviewing the report from Stage 1, you might want to list all the EGL directory and file names that are produced by the Stage 1 migration tool. The easiest way to do this is to create a .bat file containing DB2 commands similar to those in the following lines:

```
db2 connect to VGMIG
db2 set schema migschema
db2 select distinct('ListOfEGLFileNames ') from configplan
>>%1\DBResults%2.txt
db2 select distinct(substr(eglfilename, 1, 130)) from eglfile order by 1
>>%1\DBResults%2.txt
db2 select distinct('ListOfVAGenPartsAndEGLFileNames') from configplan
>>%1\DBResults%2.txt
db2 select substr(vgpartname, 1, 32), substr(eglfilename,1,130)
from vgpart_lineage order by 1, 2
>>%1\DBResults%2.txt
db2 disconnect VGMIG
```

Notes:

1. The first and last commands connect and disconnect from the migration database.
2. The db2 **set** command specifies the schema name so that the table names do not have to be qualified in the rest of the .bat file.
3. The db2 **select distinct('ListOfEGLFileNames ')** and **db2 select distinct('ListOfVAGenPartsAndEGLFileNames')** commands simply provide separators in the output report.
4. The **db2 select distinct(substr(eglfilename, 1, 130))** command produces a list of all the EGL file names, sorted by file name. The eglfilename includes the EGL project, package, and file name, so this query provides a quick look at the organization of your EGL workspace. You can change the value **130** to be as long as necessary to contain your complete file names. The maximum length for the default eglfile table is 512 characters.
5. The **db2 select substr(vgpartname, 1, 32), substr(eglfilename,1,130)** command produces a list of all the VAGen parts and their corresponding EGL file sorted

by part name and file name. This query includes any dummy map groups that are created by the Stage 1 migration tool. Depending on your naming conventions, this query can be helpful in determining whether parts are placed conveniently for future EGL development. Similar to the previous query, you can change the value 130 to be as long as necessary to contain your complete file names. You might also want to modify this query to include (and sort by) the part type. See Table 164 on page 463 for the correspondence between the VAGen part types and the numeric part type that is stored in the migration database.

6. The results of the queries are piped to a file using:

```
>>%1\DBResults%2.txt
```

where

- %1 is the first parameter for the .bat file and provides the name of the drive and directory where you want the output file placed.
- %2 is the second parameter for the .bat file and provides a meaningful suffix for the file name so you can distinguish the output from different runs of the Stage 1 migration tool.

To run the .bat file, follow these steps:

1. Open a DB2 Command Prompt window.
2. Run the .bat file and specify the two parameters. For example:

```
reportStage1Results.bat c:\myTestMigration\stage1 _TrialA
```

The results are placed in:

```
c:\myTestMigration\stage1\DBResults_TrialA.txt
```

Queries to assist with specific error messages

The queries in the following sections might be useful to resolving specific error or warning messages.

IWN.MIG.0302.w

This message occurs when a table has only a single row of contents. These tables can cause problems if they are used as the source for a MOVEA statement. To locate MOVEA statements in which fields in the table are used as the source for a MOVEA statement without using the table name as a qualifier, use the following queries:

```
db2 connect to VGMIG
db2 set schema migschema
db2 select vpartname, eglpartname from vpart
   where vparttype in (65536, 2147483648)
   and vgesfsorce like '%MOVEA fieldName%'
>>message302Results.txt
```

fieldName is a field in the table. You should repeat the query for each field in the table. You might also need to vary the number of blanks following MOVEA.

Resetting the migration database for Stage 2

To reset all parts in the migration database if you want to rerun Stage 2 and 3 of migration without rerunning Stage 1, use the following DB2 commands:

```
db2 update migschema.vpart set is_migrated = 'N', eglsource = NULL,
   eglpartname = NULL
db2 delete from migschema.translation_msgs
```

Backing up and restoring the migration database

To back up 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 are 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.

Appendix H. Migration tool performance

The following factors can affect the performance of the migration tool:

- General performance for Stages 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 the distribution by part type.
 - Number of migration sets.
 - 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 the 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:
 - Migration options.

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
- Number of migration sets and other migration options
- Processor speed
- Number of lines in function parts
- Clean Java workspace for Stage 1

In addition, there is a section that provides some information that you can use to help plan your disk space requirements.

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 XP with a 2.0 GB of memory and a 2.1 gigaHertz processor speed. The measurements are for EGL 7.1, and all times are in minutes.

Table 165. 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	Stage 2 time	Stage 3 time to write files	Stage 3 time to refresh or build
1	4	93	10,614	25	14	1	1	2
2	478	516	11,350	163	21	3	2	16
3	3	44	12,083	249	24	3	1	6
4	6	118	15,281	71	40	5	2	3
5	7	8	16,800	107	224	5	1	5
6	11	226	19,453	225	87	9	4	8
7	1	99	20,486	1,246	21	3	3	10
8	52	1,592	48,323	2,191	155	9	9	37

Here are some further details about the test cases in Table 165:

- Stage 1 for test case 5 used Stage 1 on Smalltalk; all other test cases used Stage 1 on Java.
- Test case 6 used the white paper technique to consolidate packages. It also used the built-in customization to split the common files by part type and part name.
- Test case 8 used the built-in customization to split the common files by part type.

Here are some general observations based on Table 165:

- Test cases 6 and 7 have similar numbers of parts, but test case 6 takes more than 4 times as long to run Stage 1 and 3 times as long to run Stage 2. Test case 7 has many more programs, but the programs are very small with very few associated parts. Conversely, test case 6 has relatively few programs, but each program is quite complex with many associated parts. The number of programs and their associated parts impacts performance in the following ways:
 - In Stage 1, all associates must be analyzed to determine whether to place the parts with the program or in a common parts file.
 - In Stage 2, even though a part is only converted to EGL once, the part must still be reviewed for each program in which it is used as an associate. For example, the I/O object of a function must be added to the record declarations or **use** form statement of each program in which the function is used.
- Test case 8 has about 3 times the number of parts and about 20 times the number of programs as test case 5. The programs in test case 5 are extremely complex, so test case 5 has about 30% more associates than test case 8. This contributes to the longer Stage 1 runtime for test case 5. In addition, Stage 1 on Smalltalk seems to be a bit slower than Stage 1 on Java, but, because there is only one large test case for Smalltalk, a performance comparison between Smalltalk and Java is unrealistic.
- Test case 8 has more than twice the number of parts as test case 7, but takes nearly 4 times as long to build the workspace. Test case 7 only has one project, so there are no cycle dependencies. Conversely, test case 8 has many cycle dependencies between the 52 projects. Even though the Stage 3 migration tool automatically invokes the tool that optimizes the EGL project build order, the build takes longer to resolve the cycle dependencies.

Based on the information in Table 165 on page 468, 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.

Number of migration sets and other migration options

Table 166 shows the impact of consolidating into a smaller number of migration sets. It also shows the impact of the Stage 2 **Migrate remaining VAGen parts** option and the Stage 3 **Override existing files** option. The table shows the same set of VAGen projects migrated using three different techniques. There are 10 projects, 226 packages, 19453 parts, and 225 programs. The two common projects contain 7734 parts and 41 programs. This represents 40% of the parts. Each of the eight migration sets represents one subsystem and includes the two common projects. There are no duplicate parts in the eight subsystems so it is possible to migrate all the subsystems as a single migration set as shown in TestCase 6C. The measurements for EGL 7.1, and all times are in minutes.

Table 166. Effect of number of migration sets and migration options

Test case	Number of migration sets	Migrate remaining VAGen parts	Override existing files	Stage 1 time	Stage 2 time	Stage 3 time to write files	Stage 3 time to refresh or build
6A	8	No	No	234	11	5	9
6B	8	Yes	Yes	234	10	5	7
6C	1	Yes	Yes	87	9	4	8

Here are some general observations based on Table 166:

- Test cases 6A and 6B use the same Stage 1 database as the starting point for Stages 2 and 3. These two test cases differ only in the options chosen for Stage 2 and Stage 3.
 - In Stage 2, there is little impact due to selecting the **Migrate remaining VAGen parts** option. This is probably due to the relatively small number (250) of unused parts.
 - In Stage 3, the time to merge newly migrated files with existing files in 6A is no different than the time to completely rewrite the files. The advantage of clearing the **Override existing files** option is that the import statements from the various migration sets are all included in the final EGL files.
- Test case 6C added one project that contains a high-level PLP part that points to the 10 other projects. This technique enables all eight subsystems to migrate as a single migration set.
 - In Stage 1, there is a significant savings in time because the common parts do not need to be loaded and analyzed for each migration set.
 - In Stage 2, there is some savings because the common parts do not have to be analyzed for cross part migration in multiple migration sets. However, because the migration tool does convert all the parts, there is not as big a percentage savings as in Stages 1 and 3.
 - In Stage 3, there is a small savings in time due to a combination of several factors:
 - The common parts are only analyzed once to determine the **import** statements.

- The EGL files are only written one time.
- There is no need for any merge logic for the files because all the parts are migrated at the same time.

Processor speed

Table 167 shows the impact of changing the machine memory and processor speed. Test case 8A used a machine with 1.0 GB of memory and 1.1 gigaHertz processor speed. Test case 8B used a machine with 2.0 GB of memory and 2.1 gigaHertz processor speed. The measurements are for EGL 7.1, and all times are in minutes.

Table 167. Effect of processor speed on migration times

Test case	Number of projects	Number of packages	Number of parts	Number of programs	Stage 1 time	Stage 2 time	Stage 3 time to write files	Stage 3 time to refresh or build
8A	52	1,592	48,323	2,191	290	not available	not available	not available
8B	52	1,592	48,323	2,191	155	9	9	37

Based on Table 167, 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 168 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). The measurements are for EGL 5.1.2, using a machine with 1.0 GB of memory and 1.1 gigaHertz processor speed. All times are in minutes.

Table 168. Effect of in-function lines on migration times

Test case	Before removing blank lines in VAGen				After removing blank lines in VAGen			
	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
9	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 1

The next table shows the impact of having a clean workspace at the start of Stage 1 migration. Test case 13 contains 3 projects, 29 packages, 2660 parts, and 33 programs. Test case 14 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. The measurements are for EGL 5.1.2, using a machine with 1.0 GB of memory and 1.1 gigaHertz processor speed. All times are in minutes.

Table 169. Effect of clean Java workspace on migration times

Test case	Stage 1 Time without Clean Workspace	Stage 1 Time with Clean Workspace
10	17	12
11	11	1

Based on Table 169, 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 137.

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 161.

Disk space requirements

There is no direct relationship between the disk space requirements of a VisualAge Generator application and the corresponding EGL application.

The next table provides disk space requirements for the same test cases shown in Table 165 on page 468. The EGL measurements were all done immediately after migration, before correcting any messages in the Problems view and before doing any generation. The measurements are for EGL 7.1, and all sizes are in megabytes (MB).

Table 170. Disk space requirements

Test case	VAGen .dat file size	DB2 backup file size for Stage 1	EGL project interchange file	EGL workspace size	EGL .metadata directory size	EGL total workspace size
1	11.4	114.7	8.2	40.6	40.0	80.6
2	33.4	163.9	6.3	157.5	41.5	199.0
3	15.3	163.9	9.5	155.7	59.3	215.0
4	13.6	180.3	1.5	86.4	26.6	113.0
5	15.8	426.1	3.6	190.6	38.4	229.0
6	24.2	262.2	3.0	169.3	31.7	201.0
7	26.1	196.7	6.7	147.0	46.0	193.0
8	64.1	393.3	32.5	453.0	156.0	609.0

Here are some general observations based on Table 170 on page 471:

- The VAGen .dat file size is the size of an exported VisualAge Java repository file that contains only the VAGen projects and packages from the migration set.
- The DB2 backup file size is the size of the backup file for the migration database at the end of Stage 1 after running the DB2 runstats.bat command file. The Stage 2 size is larger due to the addition of EGL source code to the database.
- Test cases 1, 3, and 8 all have VAGen Web Transaction programs and UI records. The migration tool creates EGL Web projects for each project within the test case that contains either a VAGen Web Transaction or UI record. EGL Web projects contain the following files:
 - standard .properties files such as the csogw.properties and gw.properties files
 - standard .jar files such as the fda7.jar, hpt.jar and hptGateway.jar files
 - standard .jsp files such as the Vagen1LogonPage.jsp and CSOERRORUIR.jsp files
 - .jsp files that are generated for the UI records in the project

These files result in a much larger project interchange file size than for projects that do not contain VAGen Web Transaction programs or UI records.

Appendix I. VisualAge Generator and EGL interoperability

When you migrate the following types of programs, you must generate the EGL programs or relink the VAGen programs to provide interoperability for the VAGen and EGL programs:

- CICS programs
- iSeries programs
- VAGen Web Transactions
- Cross System Product

VisualAge Generator and EGL interoperability on z/OS CICS

All programs in the same z/OS CICS run unit must be linked with the same runtime server product. A run unit for z/OS CICS consists of all programs that run under the same CICS transaction, including any program transferred to using a **call** or **transfer to program** (VAGen DXFR) statement. After any program in the run unit is linked with the IBM Rational COBOL Runtime for zSeries, you do not necessarily need to migrate the remaining VAGen programs to EGL or regenerate other previously migrated programs with EGL, but, as a minimum, you must relink all the programs in the run unit to use the IBM Rational COBOL Runtime for zSeries. If you do not relink all the programs in the run unit ASRA abends can occur.

VisualAge Generator and EGL interoperability on iSeries

The runtime libraries for VisualAge Generator and EGL do not readily facilitate coexistence of programs generated with VisualAge Generator and programs generated with EGL. This section outlines modifications that you can make to achieve the coexistence.

This section uses the following terminology:

- *QVGEN* is the runtime library shipped with the VAGen product
- *QEGL* is the runtime library shipped with EGL
- *PREP step* refers to the task of deploying the generated objects from the generator machine to the target host machine. This step involves compiling and creating a program. Other objects that must be deployed include message tables, DataTables, and format modules.

For VisualAge Generator, you can customize the PREP step by modifying the templates that reside on the VAGen Developer client workstations. The following templates are used in the PREP step:

- efk24pcl.tpl
- efk24pmn.tpl
- efk24psc.tpl
- efk24psm.tpl
- efk24wcl.tpl
- efk24wsc.tpl
- efk24ppm.tpl

For EGL, the PREP step is carried out by the build server (shipped with QEGL) using a REXX build script named FDAPREP. You can customize the build script to change the compiler or CRTPGM options. In addition, you can control the logic of FDAPREP by setting up user symbolic parameters in the build descriptors and having FDAPREP act on the value of these symbolic parameters.

The following problems can occur when calling programs in a mixed environment:

- **Problem 1:** The main problem occurs when invoking called programs (generated either with VisualAge Generator or EGL). When main programs are initialized their heap memory is allocated by a service routine in QVGNMEM which is a server program in QVGEN or in QEGL, depending on which product is in the library list. For example, if QEGL is in the library list (alone or ahead of QVGEN) then QEGL/QVGNMEM is used. However, any called programs are bound during the PREP step to either QVGEN/QVGNMEM or to QEGL/QVGNMEM, depending whether the PREP step specifies the VisualAge Generator or EGL library. For a main program to invoke a called program correctly, *both* must be bound to the same QVGNMEM. Otherwise heap memory is corrupted and an abend occurs.

Solution: Use one of the following techniques to ensure that the same QVGNMEM is used for both the calling and the called program:

- **Technique 1:** Use QEGL consistently by following these steps:
 1. Change the PREP step for VAGen called programs so that they use BNDDIR(QEGL/QEGLBND) instead of BNDDIR(QVGEN/QVGN). You can make this change by modifying the VAGen template efk24pcl.tpl.
 2. At runtime, set LIBL so that the QEGL library is listed before the QVGEN library.
- **Technique 2:** Use QVGEN consistently by following these steps:
 1. Change the PREP step for EGL called programs so that they use BNDDIR(QVGEN/QVGN) instead of BNDDIR(QEGL/QEGLBND). You can make this change by modifying the procedure PCL in the EGL build script FDAPREP.
 2. At runtime, set LIBL so that the QVGEN library is listed before the QEGL library.

Technique 1, using QEGL, requires that you run the PREP step again for all VAGen called programs, whether they are called by EGL-generated programs now or in the future, to bind the VAGen called programs with the correct (QEGL) library. Technique 2, using QVGEN, requires that you run the PREP step again for all the EGL programs when QVGEN goes out of service. In either case, when all your VAGen programs are migrated, you can remove the QVGEN library. To determine which technique to use, you should consider the number of programs for which you must run the PREP step. For example, if you expect to create a number of new systems using EGL and these systems might call programs previously written in VisualAge Generator, then you might want to use Technique 1 so that the new systems start with the new QEGL library.

- **Problem 2:** A VAGen-generated Java client program can only do a remote call using the VAGen catcher program, which in turn can only call programs that are bound with the VAGen QVGEN module. Similarly an EGL-generated Java client program can only do a remote call using the EGL catcher program, which in turn can only call programs that are bound with the EGL QEGL module. This restriction affects programs called remotely from Java, including programs called from Java wrappers.

Solution: Use one of the following techniques to ensure that the QVGNMEM module for the called program matches that for the catcher program:

- **Technique 1:** If the client program is generated with EGL, then bind the called program and any programs it calls with QEGL regardless of whether the programs are generated with VisualAge Generator or EGL. This is the preferred technique.
- **Technique 2:** If the client program is generated with VisualAge Generator, then bind the called program and any programs it calls with QVGEN, regardless of whether the called program is generated with VisualAge Generator or EGL.
- **Problem 3:** VAGen-generated Web Transaction programs are not interoperable with EGL-generated Web Transactions, including any programs that are called or transferred to using the **call**, **transfer**, or **show** statements.

Solution: You must migrate all the VisualAge Generator Web Transactions to EGL at the same time. Otherwise, the results might not be predicable, and crashes in the called or transferred-to program might occur.

The following table summarizes the interoperability of VisualAge Generator and EGL programs for iSeries:

Program kind	Problem
Main programs	None. VisualAge Generator main programs can run in the QEGL library and EGL main programs can run in the QVGEN library.
Called programs	See Problem 1, described previously
Remotely called programs	See Problem 2, described previously
Web transaction programs	See Problem 3, described previously
Print services programs	None
Format modules	None
Message tables	None
Data tables	None
DDS files	None

VisualAge Generator and EGL interoperability for Web Transactions

For Web Transaction programs and UI records (EGL VGWebTransaction programs and VGUIRecords), the following rules apply:

- Before you debug your VGWebTransaction programs, you must generate all the VGWebTransaction programs and VGUIRecords.
- When you deploy or use an EGL-generated bean, you must perform the following tasks:
 - Regenerate all the VGUIRecords that you plan to include in the new WAR file.
 - Regenerate all of the corresponding VGWebTransaction programs.
 - Migrate and generate any programs that the VGWebTransaction program calls or transfers to using the **call**, **transfer**, or **show** statements (VAGen CALL, DXFR, or XFER statements).

You do not need to migrate and generate programs that are referenced in a program link or hyperlink.

Cross System Product interoperability

You must generate the following programs and their related DataTables and FormGroups:

- Any programs and associated DataTables and Form Groups that were generated with Cross System Product Version 3.3 or earlier, including programs generated with the Cross System Product Version 3.3 COBOL generation facility.
- Any programs and associated DataTables and FormGroups that were generated with Cross System Product Version 4.1. These programs must be generated with EGL in accordance with the following rules:
 - If you generate a program with EGL, you must also generate the following parts with EGL:
 - All FormGroups used by the program.
 - All DataTables that are specified with the **validatorDataTable** property for any form field in the FormGroup.
 - If you generate a FormGroup with EGL, you must also generate the following parts with EGL:
 - All programs that use the FormGroup.
 - All DataTables that are specified with the **validatorDataTable** property for any form field in the FormGroup
 - If you generate a DataTable with EGL, you do not need to generate any programs or FormGroups that use the DataTable.
 - As soon as you generate any program, DataTable or FormGroup with EGL, then you must use the EGL server product for your runtime environment.

Notices

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 documentation 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 grant 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.

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.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation 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:

Intellectual Property Dept. for Rational Software
3600 Steeles Avenue East
Markham, ON
Canada L3R 9Z7

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 document 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.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

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:

- IBM
- AIX
- CICS
- DB2
- IMS
- iSeries
- MVS
- OS/2
- OS/400
- Rational
- VisualAge
- WebSphere
- 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

A

alternate specification record 411, 412
ambiguous situations 25, 27, 30, 34, 65, 205, 257
 data items 65
 EZE words 117
 functions 94
 map groups and maps 77
 other statements 106
 programs 87
 records 70
 tables 76
appendix index entry 249, 259, 265, 281, 283, 286, 307, 325, 339, 349, 352, 395, 427, 457
array 35, 277, 293, 327, 332, 341
 dynamic 4
 map 52
 multidimensional 4
associated parts 4, 30, 31, 35, 39, 52, 65, 140, 165, 406
 migrating with 46
 migrating without 46
associated program parts 89
AUDIT 240

B

batch mode 19, 25, 26, 171, 180, 184, 186, 197, 199
bind control 208
bind control part 209, 213
 program-specific 214
 using as template 212
build descriptor 206, 208
 debug 209
 default 216
 default 207
 EGL 213
build descriptor option 213
 bind 214
 genproject 209
 link edit 214
build descriptor options 206, 209, 222, 223
 COBOL generation
 reviewing 208
 Java generation
 reviewing 209
 reviewing
 general 206
build descriptor parts 203
 reviewing 206
build parts 39, 52, 354
build path 15, 31, 34, 35, 38, 39, 40, 192, 205

C

CALL AUDIT 349
CALL COMMIT 349
CALL CREATX 349
CALL CSPTDLI 349
CALL EZCHART 349
CALL RESET 349
CICS 7, 9, 33, 41, 76, 77, 88, 231, 354, 372, 376, 377, 379, 385, 395, 396, 412, 420
 CALL CREATX differences 241
 commit differences 241
 EZE special data word differences
 EZEAPP 242
 EZEDEST 242
 EZEDESTP 242
 EZELTERM 242
 EZERCODE 242
 EZERT8 242
 EZESEGTR 242
 EZEUSR 242
 EZEUSRID 242
 EZECONCT differences 241
 features not supported
 native environments 241
 function words
 not supported, native environments 240
 resource associations
 not supported, native environments 241
 rollback differences 241
 service routines
 not supported, native environments 240
 XFER, DXFR 241
COBOL generation
 generating and testing 222
common code 8, 22, 29, 31, 32, 33, 34, 35, 45, 109, 154
common parts 128, 130, 151, 152
comparison value items
 DL/I I/O 105
configuration map 12, 15, 19, 20, 21, 22, 23, 25, 28, 32, 42, 43, 44, 45, 89, 149, 150, 151, 152, 153, 154, 161, 163, 165, 166, 167, 190, 354, 460
containerContextDependent property 15, 31, 34, 35, 40, 52
control part 23, 27, 42, 181, 352, 353, 354, 396, 423
 bind control 26, 41, 42, 383, 425
 generation option 26, 41
 generation options 39, 42, 52, 353, 354
 link edit 26, 41, 42, 383, 423, 424
 linkage option 26
 linkage table 372
 Calllink 372
 Crtxlink 377
 Dxfrlink 378
 Filelink 376

control part (*continued*)

 linkage table options 353
 resource association 26, 42, 379
 resource associations 353
converse 30, 33, 53, 113, 311
cross-part migration 4, 20, 30, 31, 34, 46, 65

D

data item 12, 33, 36, 38, 39, 42, 52, 65, 66, 70, 112, 117, 259, 325, 326, 395, 405, 408, 409, 411, 412, 427
 assignment statements 108
 implicit 88, 106, 420
 preferences 178
 renaming 41, 177, 249, 253
 shared 31, 33, 34, 40, 44, 45, 65, 67, 68, 70, 74
database 9, 19, 20, 21, 23, 33, 95, 96, 131, 140, 142, 180, 181, 184, 189, 354, 396, 403, 404, 419, 423
DB2
 performance information 171
debug
 EZESQLCA 237
 EZESQRRM 237
 EZESQWN6 237
 runtime differences 237
 maps 237
 SQL 237
deleteAfterUse 421
destPort 423
display 30, 33, 35, 45, 46, 53, 94, 110, 284, 286, 288, 291, 311, 414, 415, 418, 427
DL/I I/O
 comparison value items 105

E

edit function 30
edit routine 30, 44, 46, 68, 82, 264, 298, 300, 395, 408, 413
edit table 30, 47, 261, 264, 278, 279
error messages
 HPT 395
 IWN.MIG 405
 IWN.VAL 435
 IWN.XML 448
evensql 409, 411
export 27, 195, 405, 406
External Source Format 20, 23, 25, 27, 28, 34, 46, 51, 78, 97, 131, 139, 163, 164, 171, 184, 195, 197, 293, 405, 406, 408, 409, 414, 421, 427, 457, 463
EZE words 51, 65, 257, 339
 date and time
 EZEDAY 342
 EZEDAYL 342
 EZEDAYLC 342

EZE words (*continued*)

date and time (*continued*)

EZEDTE 342
EZEDTEL 342
EZEDTELC 342
EZETIM 342

DL/I

EZEDLCER 341
EZEDLCON 341
EZEDLDBD 341
EZEDLERR 341
EZEDLKEY 341
EZEDLKYL 341
EZEDLLEV 341
EZEDLPCB 341

EZELTERM

ambiguous situations 117

EZESYS

ambiguous situations 118

EZEWAIT

ambiguous situations 120

floating point math functions

EZEFLADD 348
EZEFLDIV 348
EZEFLMOD 348
EZEFLMUL 348
EZEFLSET 348
EZEFLSUB 348

general function

EZEBYTES 345
EZEC10 345
EZEC11 345
EZECOMIT 345
EZECONV 345
EZEG10 345
EZEG11 345
EZEPURGE 345
EZEROLLB 345
EZEWAIT 345

general math functions

EZEABS 347
EZECEIL 347
EZEEXP 347
EZEFLFLOOR 347
EZEFREXP 347
EZELDEXP 347
EZELOG 347
EZELOG10 347
EZEMAX 347
EZEMIN 347
EZEMODF 347
EZENCMPR 347
EZEPOW 347
EZEPRSCN 347
EZEROUND 347
EZESQRT 347

math 347

object scripting

EZESCRPT 349

other data

EZE Aid 343
EZEAPP 343
EZECNVCM 343
EZECONVT 343
EZEDEST 343
EZEDESTP 343
EZEFECD 343

EZE words (*continued*)

other data (*continued*)

EZELOC 343
EZELTERM 343
EZEMNO 343
EZEMSG 343
EZEORDER 343
EZEORDER 343
EZERCOD 343
EZEREPLY 343
EZERT2 343
EZERT8 343
EZESEGM 343
EZESEGTR 343
EZESYS 343
EZETST 343
EZEUSR 343
EZEUSRID 343

program flow

EZECLOS 340
EZEFLD 340
EZERTN 340

SQL

EZECONCT 341
EZESQCOD 341
EZESQISL 341
EZESQLCA 341
EZESQRD3 341
EZESQRRM 341
EZESQWN1 341
EZESQWN6 341

string

EZESBLKT 346
EZESCCWS 346
EZESCMR 346
EZESCNCT 346
EZESCOPY 346
EZESFIND 346
EZESNULT 346
EZESSET 346
EZESTLEN 346
EZESTOKN 346

trigonometric math functions

EZEACOS 348
EZEASIN 348
EZEATAN 348
EZEATAN2 348
EZECOS 348
EZECOSH 348
EZESIN 348
EZESINH 348
EZETAN 348
EZETANH 348

user interface

EZEUIERR 348
EZEUILOC 348

EZEDLPCB 92

EZELOC 241

EZEPURGE 240

F

fill character 70, 261, 278, 298

filter 22, 23, 179, 396

configuration map 150

configuration maps 151

packages 129

filter (*continued*)

projects 127, 129

repository 21, 125, 126, 127, 131, 139, 141, 150, 165, 167

version 127

version depth 127, 128, 150, 151

version name 127, 128, 150, 151

FormGroup 77, 205

function 15, 29, 30, 31, 36, 39, 40, 42, 43, 44, 45, 46, 51, 65, 68, 69, 71, 72, 76, 82, 83, 85, 94, 95, 96, 99, 103, 104, 105, 106, 107, 109, 110, 111, 113, 115, 117, 118, 120, 176, 300, 301, 307, 308, 310, 311, 325, 327, 406, 408, 413, 415, 416, 418, 419, 438

common 33, 45, 52

DL/I I/O 321

DL/I statements 322

I/O 312

renaming 41, 177, 249, 253

SQL 51, 176

SQL I/O 313, 315, 318

functions

handling ambiguous situations 94

SQL I/O 97

G

general function EZE words 345

generate 31, 33, 34, 40, 46, 47, 416

program 8, 20, 30, 408, 423

programs 26

report 130, 131, 133, 163

tables 26

VisualAge Generator 141

generation option 12, 353, 354, 383, 422, 423

conversion table values 370

VisualAge Generator 36

generation option part 12

generation options 206

H

help map 80, 288, 290, 301

help map group 53, 80, 178, 396, 413, 414

help map names 79

high-level PLP project 21, 22, 127, 139, 140, 141

creating 140

I

I/O options for default (unmodified)

DL/I statements 322

implicit item 52, 106, 420

in programs 88

import 36, 181, 185, 189, 190, 195, 405, 406

External Source Format 27

Stage 3 tool 20

import into workspace 181, 184, 186

import statement 26, 28, 31, 34, 35, 36, 38, 39, 40, 51, 53, 90, 192, 205, 283, 286, 354

isDecimalDigit 83, 296

IWN.MIG 405
IWN.VAL 435
IWN.XML 448

J

Java and C++ differences
 EZE special data words
 EZCONVT 244
 EZERCODE 244
 general 244
 maps 244
 SQL
 EZESQLCA 244
 EZESQRRM 244
 EZESQWN6 244
Java generation
 generating and testing 223
JSP
 invalid character constant 450

L

library 14, 149, 151, 161, 372
 management 4, 5, 6, 8, 15, 44
 Smalltalk 20, 24, 148, 150, 162
link edit 208
linkage options 208
linkage options parts 203
 reviewing 210
log file 21, 23, 25, 27, 139, 163, 187, 188,
 196, 198, 199
 name 132, 156, 181, 189
 name preference 159
 Stage 2 migration 185

M

map 28, 31, 36, 44, 45, 52, 65, 67, 68, 80,
 94, 109, 142, 205, 249, 327, 415, 418, 419,
 427
 assignment statements 108
 constant field 291, 293, 296
 display 30, 33, 45
 general syntax, map type, and
 properties 288
 EZEMSG 343
 numeric hardware attribute 83
 print 420
 printer 30, 33, 45
 general syntax, map type, and
 properties 290
 renaming 41, 177, 253
 spanning 152
 unnamed variable fields 85
 unprotected constants 86
 variable field 30, 46, 82, 291, 293,
 296, 298, 299
 error messages 300
 XFER with 338
map edits 70
map group 28, 30, 32, 33, 41, 42, 43, 65,
 77, 142, 395, 406, 412, 413, 414
 general syntax and floating areas 284
 renaming 249
 spanning 128, 152

map group part 12
map groups 283
 ambiguous situations 77
 device names, types, sizes 286
 general information 283
map item
 checking for NULL 113
 edit routine 68
 implicit 88
map names 79
map part 12
map properties
 error messages 264
 general edits 261
 general information 261
 numeric edits 263
maps 286
 ambiguous situations 77
 functions and I/O options 311
 general information 286
messages 23, 67, 73, 78, 83, 87, 99, 103,
 104, 139, 148, 163, 188, 196, 199, 261,
 279, 288, 290, 291, 293
 debug 132, 156
 fatal 156
 from migration tools 395
 HPT 395
 informational 132, 156
 IWN.MIG 405
 IWN.VAL 435
 IWN.XML 448
 Problems view 26, 173, 179, 199, 427
 Stage 1 common 395
 Stage 1 on VisualAge for Java 399
 Stage 1 on VisualAge for
 Smalltalk 403
 Stage 2 185, 405
 Stage 3 181, 189
 warning 132, 156
MigPreferences.xml 123, 124, 125, 127,
 130, 138, 147, 148
 sample 133, 157
migration database 21, 22, 23, 24, 25, 34,
 123, 126, 131, 132, 139, 142, 147, 148,
 150, 155, 163, 164, 187, 189, 192, 406,
 427, 457, 462, 463
 creating 459
 resetting
 tables 460
 tables 460
 views 460
migration feature 148
 adding 124
 loading 148
migration plan 21, 22, 23, 125, 126, 131,
 132, 139, 140, 142, 149, 150, 155, 156,
 157, 162, 163, 164, 167, 396, 460
 creating manually 141
 high-level configuration maps 165
 multiple 23
migration set 21, 22, 23, 25, 30, 31, 34,
 35, 42, 43, 45, 46, 51, 52, 78, 90, 126, 127,
 128, 129, 138, 139, 140, 142, 150, 152,
 153, 155, 162, 165, 181, 184, 185, 187,
 192, 283, 397, 405, 406, 460
migration sets
 processing 35

migration tool performance 466

O

output files 28, 29
Overwrite PLN 162, 167

P

package 12, 13, 15, 19, 22, 23, 26, 27, 28,
 34, 36, 38, 39, 40, 42, 43, 51, 53, 74, 89,
 90, 125, 128, 129, 130, 131, 138, 142, 151,
 152, 163, 185, 187, 188, 192, 195, 196,
 198, 199, 205, 255, 283, 354, 372, 377,
 399, 403, 406
 naming 152
 renaming 129, 154
part name 15, 35, 38, 41, 205
 conflicting 51, 93, 176, 178, 353
 duplicate 31, 41
 invalid 41, 76, 77, 87, 249, 253, 383,
 423, 424
 renaming 177, 253
 resolution 31, 40, 41
 VisualAge Generator 405
part name restrictions 35
parts 22
 large numbers 19
 placement 42
 single file mode 27
 Stages 1, 2, 3 27, 42
 placing 35
 single file mode 195, 196
 Stages 1 to 3 42
 Project List Parts (PLP) 20
 Stages 1, 2, 3 26
 Stages 1, 2, 3 20
performance
 migration tool 466
planning your migration 3, 4
preference file 404
 migration 162
 Java 22
preferences 22, 27, 189, 190, 396, 403,
 404
 build descriptor 35
 deriving file names 159
 editor 35
 renaming 177
 repository filters 165
 required EGL 172
 sample file 124
 setting 195
 Single File Mode 174
 SQL 155, 176
 Stage 1 21, 23, 42, 167
 Java 21
 setting 141, 143
 setting on Java 124
 setting on Smalltalk 148
 Stage 2 24, 25, 184
 setting 180
 Stage 3 25, 26
 suggested 173
 VAGen Migration Preferences 41,
 174, 198, 318

- preferences (*continued*)
 - VAGen Migration Syntax
 - Preferences 177
 - VAGen Syntax Migration
 - Preferences 313, 315
 - workbench
 - setting 171
- Problems view 13, 30, 40, 42, 47, 51, 52, 53, 69, 73, 77, 78, 87, 88, 96, 97, 109, 110, 111, 112, 113, 173, 197, 204, 206, 211, 354, 372, 376, 377, 406, 414, 420, 424, 427, 435, 448
- program 30, 31, 33, 35, 41, 42, 43, 44, 46, 53, 65, 70, 80, 90, 96, 113, 141, 205, 301, 303, 307, 396, 406, 420, 421
 - behavior 19, 29
 - implicit data items 88
 - migrating with 45
 - properties 35, 36
 - renaming 249, 253
 - sample
 - Stage 1 tool 21, 24
- programs 300
 - sample 19
 - single file migration 27
- project list part 15
- project list part (PLP) 22, 140
- project name 22, 43, 127, 128, 129, 130, 139, 140, 142, 152, 185, 186
- PSB 303, 350

R

- record 31, 36, 52
 - renaming 41
- records 31, 65, 70, 74, 75, 90, 109, 265, 266, 409, 410, 411, 419, 420
 - alternate specification 72, 73, 268
 - assignment statements 108
 - common 52
 - DL/I 274
 - I/O 311
 - indexed 312
 - level 77 items 71, 72
 - message queue 312
 - redefined 70, 71
 - relative 312
 - renaming 249, 253
 - serial 312
 - SQL 270, 427
 - User Interface (UI) 23, 42, 276, 277, 278, 279, 280, 311, 396
 - working storage 24, 177
- Rename
 - User Exit Information 174
- renaming 22, 23, 42, 80, 125, 130, 131, 149, 405, 407
- Renaming page 129, 154
- Renaming Prefix 41, 177, 343, 419
- renaming rules 131, 152, 153, 399, 403, 460
- report 21, 23, 33, 125, 139, 396
 - Stage 1 migration 142, 155, 157, 159, 164, 181, 190
- repository 14, 15, 19, 125, 137, 142, 181
 - Java 20, 22, 24, 138
 - source code 4, 6, 8, 20, 25, 26, 354

- Repository explorer 13
- Repository Filter 151
- repository management 15
- reserved word list 41
- reserved words 23, 25, 27, 42, 249, 276, 281, 288, 290, 301, 396
 - EGL
 - list 249
 - FormGroup names 77
 - Java
 - list 255
 - program names 87
 - SQL 177
 - list 253
 - table names 76
 - UI record names 75
- resource association 208, 354, 379, 424, 448
- resource associations parts 203
 - EGL
 - reviewing 211
- results
 - intermediate 19
 - migration 407
 - pilot project 8
 - reviewing 130
 - Stage 1 139
 - migration database 164
- running the tool
 - Stage 1 24, 162, 457
 - Java 138
 - Smalltalk 162
 - Stage 2 24, 184
 - batch mode 25, 185
 - user interface 184
 - Stage 3 25, 189
 - batch mode 26
- runtime differences
 - COBOL
 - CALL 238
 - DXFR 238
 - maps 238
 - XFER 238
 - Java
 - CALL 239
 - DXFR 239
 - XFER 239

S

- service routine 93, 257, 349
 - general syntax 349
 - VisualAge Generator and EGL
 - equivalent routines 349
- SET map PAGE 110
- single file migration
 - batch mode 197
 - user interface 195
- single file mode 27, 28, 29, 41, 42, 46, 78, 196, 198, 283, 286, 406, 409, 413, 427
 - migration 195
 - parts placement 195
 - set up 195
- source code 3, 8, 19, 20, 23, 24, 25, 26, 27, 29, 33, 50, 140, 165, 203, 205, 231, 349, 435
 - extracting from Java 123, 138

- source code (*continued*)
 - extracting from Smalltalk 162
 - pilot project 5
 - reviewing 205
- SQL 9, 29, 259, 266, 354, 410, 416
 - checking item for NULL 419
 - checking items for NULL 113
 - hard errors 114
 - statements 65
 - WHERE clause 35, 65
- SQL clauses
 - FOR UPDATE OF 315, 318
 - GROUP BY 315, 318
 - HAVING 315, 318
 - INTO 315, 318
 - ORDER BY 315, 318
 - SELECT 315, 318
 - WHERE 315, 318
- SQL EZE words 341
- SQL I/O 415, 416
 - Execution time statement build 176
- SQL I/O and 'itemColumnName' 102, 103
- SQL I/O and missing SQL clauses 98
- SQL I/O options
 - ADD 313, 315, 318
 - CLOSE 313, 315, 318
 - DELETE 313, 315, 318
 - INQUIRY 313, 315, 318
 - REPLACE 313, 315, 318
 - SCAN 313, 315, 318
 - SETINQ 313, 315, 318
 - SETUPD 313, 315, 318
 - SQLEXEC 313, 315, 318
 - UPDATE 313, 315, 318
- SQL I/O statements 96
- SQL I/O with multiple updates 104
- SQL query 461, 462
- SQL record 51, 410
- SQL record definition 32
- SQL records
 - alternate specification 72
- SQL row record 66
- SQL statements
 - modified
 - with Execution time statement
 - build 318
 - without Execution time statement
 - build 315
 - unmodified
 - without Execution time statement
 - build 313
- SQL table 409, 410
- SQL tables 24, 97
- Stage 1 20
 - Java 123
 - preferences 42, 43, 124
 - running 138
 - Smalltalk 147
 - preferences 42, 43, 148
 - running 162
- Stage 2 20, 171
 - preferences
 - setting 180
 - running 184
 - batch mode 185
 - user interface 184

- Stage 3 20, 189
 - preferences 189
 - running 189
- statements 51, 118, 257, 301, 311, 325
 - use declaration 76
 - ambiguity in I/O 94
 - assignment, MOVE, MOVEA 327
 - CALL 337, 343
 - CALL, DXFR, XFER 396
 - call, transfer, show 88, 420
 - display 94
 - DXFR 338
 - FIND 109
 - flow 300, 307, 340
 - function invocation 327
 - general rules
 - data item qualification and numeric literals 326
 - I/O 65, 176, 343, 420
 - IF, WHILE, TEST 332
 - level 77 items 106
 - link edit 383
 - print 94
 - produced in ambiguous situations 65
 - RETRIEVE, FIND 331
 - SET 329
 - SETUPD, UPDATE 415
 - SQL 102, 103, 104, 307
 - use declaration 77, 303, 412
 - XFER 338
- subsystem 19, 31, 32, 33, 34, 35, 38, 40, 41, 46, 51, 52, 69, 72, 73, 74, 83, 97, 99, 103, 104, 109, 110, 141, 166, 204
- symbolic parameters 354, 379, 383, 384
 - file-related 385
 - part-related 384
 - user-defined 385
- syntax 27, 95
 - assignment, MOVE, MOVEA
 - examples 327
 - data item examples 259
 - EGL 4, 20, 29, 30, 35, 45, 177, 189, 205, 257, 395, 427
 - conversion (Stage 2) 171
 - errors 53
 - invalid 99
 - precise 29
 - general conventions
 - differences between VisualAge Generator and EGL 258
 - general display map examples 288
 - general function examples 308
 - general printer map examples 290
 - general program examples 301
 - general record examples 266
 - general table examples 281
 - map group examples 283
 - program main function example 307
 - service routine general examples 349
 - SET examples 329
 - statement examples
 - function invocation 327
 - tables 257
 - VAGen 31, 52, 177, 405, 406
 - XFER examples 338
- system library function 30, 45, 69, 92, 118, 327, 332

T

- table 31, 36, 41, 52, 141
- tables 65, 66, 73, 96, 281, 412
 - database 132, 155
 - FIND statement 109
 - renaming 249, 253
 - RETR statement 110
- Tables and Additional Records list 53, 90, 303, 354
- terminology 3
- trace 354
 - level 132, 156
 - messages 395

U

- UI record 44
 - renaming 41
- unused parts 22, 128, 130, 151, 152, 153
- update database 131, 163, 164
- use declaration 421

V

- VAGen Migration Preferences 187

W

- Windows XP
 - using DB2 459
- wizard
 - import 27, 199
- workbench
 - preferences
 - setting 171
- workspace 4, 7, 12, 13, 20, 25, 26, 33, 34, 35, 38, 74, 124, 135, 140, 141, 142, 173, 179, 185, 187, 188, 190, 195, 198, 354, 397, 398, 407
 - clean 137
 - duplicate parts 28, 31
 - restoring 138, 162
 - saving 138, 159

Readers' Comments — We'd Like to Hear from You

**Rational Business Developer
VisualAge Generator to EGL Migration Guide
Version 8 Release 0**

Publication No. SC31-6830-08

We appreciate your comments about this publication. Please comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. The comments you send should pertain to only the information in this manual or product 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.

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. IBM or any other organizations will only use the personal information that you supply to contact you about the issues that you state on this form.

Comments:

Thank you for your support.

Submit your comments using one of these channels:

- 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.

Email address



Cut or Fold
Along Line

Fold and Tape

Please do not staple

Fold and Tape



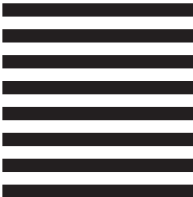
NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
Information Development
Department G71A / Bldg. 503
P.O. Box 12195
Research Triangle Park, NC
27709-2195



Fold and Tape

Please do not staple

Fold and Tape

Cut or Fold
Along Line



Program Number: 5724-J19

Printed in USA

SC31-6830-08

