Migrating from
Cross System Product Version 4.1
to
Enterprise Generation Language Version 7.1

Publication History:
Version 6.0.1.1           - Jeri Petersen
Version 7.1 Updates      - Jeri Petersen
Note: Before using this information and the product it supports, read the information in “Notices” on page 101.
## Contents

**Introduction**............................................................................................................................... 5  
Product evolution from Cross System Product................................................................. 5  
Migration path............................................................................................................................ 6  
References................................................................................................................................. 6  
Terminology............................................................................................................................... 7  
Planning your migration .......................................................................................................... 9  
Determining how to organize your source code for EGL..................................................... 15  
Differences in product capabilities for organizing your code................................................ 15  
* CSP code organization ............................................................................................................... 15  
* VAGen code organization ........................................................................................................ 16  
* EGL code organization ........................................................................................................... 16  
Considerations for structuring your EGL code....................................................................... 17  
Using the migration tools to structure your code.................................................................... 19  
**Overview of the VAGen MSL Migration Assistance Tool**....................................................... 20  
**Overview of the VAGen to EGL Migration Tool** ................................................................ 22  
Sample organization techniques .............................................................................................. 23  
Establishing naming conventions ............................................................................................. 25  
Resolving duplicate CSP member names ................................................................................ 27  
Migrating your source code from CSP to VAGen................................................................. 29  
Determining which VAGen product to use ............................................................................ 29  
Preparing to migrate your CSP/370AD source code ............................................................. 29  
Preparing to migrate your CSP/2AD source code .................................................................. 30  
Using the VAGen MSL Migration Assistance Tool............................................................... 30  
Automatic conversions during migration to VisualAge Generator .......................................... 33  
Member associations ............................................................................................................... 34  
Hints and Tips for the VAGen MSL Migration Assistance Tool ............................................. 35  
Relating your EGL naming scheme to the sandbox ................................................................. 35  
Potential problems with ESF ..................................................................................................... 35  
Using the MSL Migration Assistance Tool log file ............................................................... 36  
Improving performance ............................................................................................................ 36  
Manual steps between the tools ............................................................................................... 37  
Creating configuration maps.................................................................................................... 37  
Validating your configuration maps and programs ................................................................. 37  
Defining implicit data items...................................................................................................... 37  
Migrating your source code from VAGen to EGL............................................................... 38  
Hints and Tips for the VAGen to EGL Migration Tool ............................................................ 41  
Converting your VAGen naming scheme to EGL ................................................................. 41  
Improving performance ........................................................................................................... 42  
Finishing the migration of your code...................................................................................... 43  
Additional changes to make by hand ..................................................................................... 43  
Migrating your generation information .................................................................................. 44  
Storing control information...................................................................................................... 45  
Setting up your EGL development environment .................................................................... 49  
Information that moves during cross-part migration ............................................................. 49
Introduction

This paper is intended for customers who use Cross System Product (CSP) Version 4.1 and want to migrate to Enterprise Generation Language (EGL). The information described here is specific to EGL Version 7.1.

Product evolution from Cross System Product

Cross System Product (CSP) Version 4.1 became available in August 1992. In Version 4.1, CSP/370AD provides the development environment and COBOL generation support. CSP/370RS Version 2.1 provides the runtime support. The supported runtime environments are: MVS® Batch, MVS CICS®, IMS/VS, IMS™ BMP, MVS TSO, and CICS OS/2™. CSP/370AD Version 4.1 and CSP/370RS Version 2.1 are no longer in support.

VisualAge® Generator (VAGen) is the follow-on product to Cross System Product. VisualAge Generator Developer provides the development environment and COBOL generation support. VisualAge Generator Developer runs on either a Java™ base (VAGen on Java) or a Smalltalk base (VAGen on Smalltalk). VisualAge Generator Developer Version 4.5 is the only currently supported release. VisualAge Generator Server for MVS, VSE, and VM Version 1.2 provides support for all the MVS runtime environments that were previously supported by Cross System Product. VisualAge Generator Server for OS/2®, AIX®, Windows® NT, HP-UX, and Solaris™ Version 4.5 provides support for workstation runtime environments and originally included runtime support for CICS OS/2. However, the CICS OS/2 runtime environment is no longer in support.

Enterprise Generation Language (EGL) is the replacement for VisualAge Generator. EGL is available with IBM® Rational® Business Developer Version 7.1. Rational Business Developer Version 7.1 provides the development environment. To provide support equivalent to CSP Version 4.1, you also need the following:

- COBOL generation for the z/OS® environments, which is included with Rational Business Developer Version 7.1.
  This level of COBOL generation support provides support for z/OS Batch, z/OS CICS, IMS/VS, and IMS BMP. Cross System Product Version 4.1 customers who currently generate COBOL for runtime environments not supported by EGL COBOL generation should consider migrating in the following way:
  - MVS TSO – consider migrating to COBOL generation for the z/OS CICS environment.
  - CICS OS/2 – consider migrating to Java generation for a Windows operating system.
- Build server support for the z/OS environments, which is included with IBM Rational COBOL Runtime for zSeries® Version 6.0.1. The build server replaces the preparation process from Cross System Product.
- Runtime support for the z/OS environments, which is included with IBM Rational COBOL Runtime for zSeries Version 6.0.1. IBM Rational COBOL Runtime for zSeries provides runtime support for the z/OS environments for programs generated with CSP/370AD Version 4.1, VisualAge Generator, and EGL. IBM Rational COBOL Runtime for zSeries also provides legacy support for the MVS TSO environment for programs previously generated with either CSP/370AD Version 4.1 or VisualAge Generator. The legacy support for MVS TSO enables you to migrate from CSP/370RS to the EGL equivalent and then to migrate your source code and development environment.
This paper addresses the steps you need to take to migrate from Cross System Product to EGL. This paper assumes that you use VisualAge Generator on Smalltalk for the interim step between Cross System Product and EGL. This paper does not address the additional steps that are required if you are changing your runtime environment (for example, changing from MVS TSO to z/OS CICS or from CICS OS/2 to a native workstation environment).

Migration path

The VisualAge Generator to EGL Migration Tool does not support source code migration from a CSP base. This is because the CSP language is somewhat different from the VAGen language. In addition, CSP library management through Member Specification Libraries (MSLs) is quite different from that of VisualAge Generator, which uses the Java repository or Smalltalk manager. Therefore, there is no direct migration path from CSP to EGL. You can migrate from CSP to VisualAge Generator Version 4.5 and from there to EGL. Essentially, you briefly pass through VisualAge Generator on the way to EGL. Because you have not previously migrated from CSP, there are some additional steps you must take during this migration that a customer migrating directly from VisualAge Generator Version 4.5 does not require. The VisualAge Generator customers already did these steps when they migrated to VisualAge Generator Version 4.5.

Because you only briefly pass through VisualAge Generator, there is no need to train all your developers on VisualAge Generator. For example, you can train just 1 or 2 developers on VisualAge Generator so they can migrate your code to VisualAge Generator and then on to EGL. All the other developers only need to be trained for EGL. Alternatively, you can hire a service provider to perform the code migration so that none of your developers have to be trained on VisualAge Generator.

References

The following references were used in developing this paper and provide details beyond the scope of this paper:

- *Migrating Cross System Product Applications to VisualAge Generator Version 3.1* (SH23-0244-01). This book describes the extra steps a CSP customer needs to take when migrating to VisualAge Generator. You should not need to refer to this book because this paper includes (and updates) all the relevant information from this book for CSP Version 4.1 customers who are migrating to EGL.

- *VisualAge Generator Migration Guide Version 4.0* (SH23-0267-00). This book describes the steps to migrate your source code from MSLs to either the VAGen on Java repository or the VAGen on Smalltalk manager. This paper provides updates to certain sections of the material in this book, but does not repeat the details about how to use the VAGen MSL Migration Assistance Tool. Therefore, you need to use Part 4 of the book for details on how to migrate your source code from CSP MSLs into VAGen on Smalltalk. The comparable information for VAGen on Java is in Part 2 of the book. The book is available at: [http://www-306.ibm.com/software/awdtools/visgen/library/v45docs.html](http://www-306.ibm.com/software/awdtools/visgen/library/v45docs.html)


- The following EGL reference material:
• **EGL Generation Guide Version 7.1.** This book describes the EGL generation process, including how to define build descriptor, linkage table, resource associations, bind control, and link edit parts.

• **EGL Programmer’s Guide Version 7.1.** This book provides techniques for developing and debugging EGL programs.

• **IBM Rational COBOL Runtime Guide for zSeries Version 6.0.1 (SC31-6951).** This book describes how to set up build servers and build scripts, as well as other reference material for creating your runtime environment. Be sure to use the version of this book that corresponds to Rational Business Developer Version 7.1. This paper refers to this book as the **EGL Server Guide**.

• EGL online help system.

• The following white papers are available to assist with migrating from VisualAge Generator to EGL:
  - *How to Consolidate Projects and Packages during Stage 1 of the VisualAge Generator on Smalltalk to Enterprise Generation Language Migration Tool* (hereafter referred to as the **Project Consolidation** white paper). There is a comparable white paper for VisualAge Generator on Java.

**Terminology**

EGL and Rational Business Developer can be installed with other Rational products. This paper uses the following terminology:

• **Developer product.** IBM Rational Business Developer, used as a stand alone product or installed with other products such as IBM Rational Application Developer, IBM Rational Developer for System i, or 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 zSeries. EGL COBOL generation is included with IBM Rational Business Developer.

• **EGL build server.** Any of the products or features that provide the EGL build server support for zSeries. The build server support is included with IBM Rational COBOL Runtime for zSeries.

• **EGL runtime server.** Any of the products that provide EGL runtime support for zSeries. The EGL runtime support for zSeries is included with IBM Rational COBOL Runtime for zSeries.

Cross System Product, VisualAge Generator, and EGL all use different terminology. To help you relate the CSP terminology to the EGL terminology, Table 1 shows the three sets of terminology.
<table>
<thead>
<tr>
<th>Cross System Product</th>
<th>VisualAge Generator</th>
<th>Enterprise Generation Language (EGL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Development environment</td>
<td>Image (Smalltalk)</td>
<td>Workspace</td>
</tr>
<tr>
<td></td>
<td>Workspace (Java)</td>
<td></td>
</tr>
<tr>
<td>Member specification library (MSL)</td>
<td>ENVY manager (Smalltalk)</td>
<td>EGL projects, packages, and files</td>
</tr>
<tr>
<td></td>
<td>ENVY repository (Java)</td>
<td></td>
</tr>
<tr>
<td>MSL concatenation sequence</td>
<td>Parts in the image (Smalltalk)</td>
<td>EGL Build Path and import statements</td>
</tr>
<tr>
<td></td>
<td>Parts in the workspace (Java)</td>
<td></td>
</tr>
<tr>
<td>CSP member type</td>
<td>VAGen part type or VAGen part class</td>
<td>EGL part type</td>
</tr>
<tr>
<td>CSP member</td>
<td>VAGen part</td>
<td>EGL part within a file</td>
</tr>
<tr>
<td>Data item member</td>
<td>Data item part</td>
<td>DataTable part</td>
</tr>
<tr>
<td>Global data item</td>
<td>Shared data item</td>
<td>Type definition referencing a DataItem part</td>
</tr>
<tr>
<td>Local data item</td>
<td>Non-shared data item</td>
<td>Type definition using a primitive data type</td>
</tr>
<tr>
<td>Record member</td>
<td>Record part</td>
<td>Record part</td>
</tr>
<tr>
<td></td>
<td>Note: The migration tool converts all record definitions to EGL structured records to preserve the behavior.</td>
<td></td>
</tr>
<tr>
<td>PSB member</td>
<td>PSB part</td>
<td>Record part with the PSBRecord stereotype</td>
</tr>
<tr>
<td>Table member</td>
<td>Table part</td>
<td>DataTable part</td>
</tr>
<tr>
<td>Map group member</td>
<td>Map group part</td>
<td>FormGroup part</td>
</tr>
<tr>
<td>Map member</td>
<td>Map part</td>
<td>Form part</td>
</tr>
<tr>
<td>• display map</td>
<td>• display map</td>
<td>• textForm</td>
</tr>
<tr>
<td>• printer map</td>
<td>• printer map</td>
<td>• printForm</td>
</tr>
<tr>
<td>Process member</td>
<td>Function part</td>
<td>Standalone function part</td>
</tr>
<tr>
<td></td>
<td>Note: The migration tool converts all function parts to EGL standalone function parts.</td>
<td></td>
</tr>
<tr>
<td>Statement group member</td>
<td>Function part</td>
<td>Standalone function part</td>
</tr>
<tr>
<td></td>
<td>Note: The migration tool converts all function parts to EGL standalone function parts.</td>
<td></td>
</tr>
<tr>
<td>Application member</td>
<td>Program part</td>
<td>Program part</td>
</tr>
<tr>
<td>Structure items (fields in a record)</td>
<td>Structure items (structure of fields in a record)</td>
<td>Structure fields</td>
</tr>
<tr>
<td>Array (multiply occurring item in a record or map)</td>
<td>Array (multiply occurring item in a record or map)</td>
<td>Structure field array</td>
</tr>
<tr>
<td>Process option and process object</td>
<td>I/O option and I/O object</td>
<td>EGL I/O statement</td>
</tr>
<tr>
<td>Control parts</td>
<td>Control parts</td>
<td>Build parts</td>
</tr>
<tr>
<td>Generation options (stored outside the MSL)</td>
<td>Generation options part</td>
<td>Build descriptor part</td>
</tr>
<tr>
<td>Generation option</td>
<td>Generation option</td>
<td>Build descriptor option</td>
</tr>
<tr>
<td>Cross System Product</td>
<td>VisualAge Generator</td>
<td>Enterprise Generation Language (EGL)</td>
</tr>
<tr>
<td>--------------------------------------------------</td>
<td>---------------------</td>
<td>-------------------------------------</td>
</tr>
<tr>
<td>Linkage table (stored outside the MSL)</td>
<td>Linkage table part</td>
<td>Linkage options part</td>
</tr>
<tr>
<td>Resource association (optionally stored with the application member)</td>
<td>Resource associations part</td>
<td>Resource associations part</td>
</tr>
<tr>
<td>Bind command (stored outside the MSL)</td>
<td>Bind control part</td>
<td>Bind control part</td>
</tr>
<tr>
<td>Link edit (stored outside the MSL)</td>
<td>Link edit part</td>
<td>Link edit part</td>
</tr>
</tbody>
</table>

### Planning your migration

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

- Changes to the CSP language, including many enhancements such as new data types, multidimensional structure field arrays, dynamic arrays, and the `case` statement.
- 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 the use of the EGL build server instead of preparation JCL templates and procedures 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 instead of using MSLs.

The differences between the CSP language and EGL are extensive. In the past when you upgraded from one version of Cross System Product to a new version, there were only minor changes to the language. The previous CSP and VAGen migration tools were able to migrate each part independently of any other parts. However, due to the differences between CSP and EGL, the migration tools need to convert 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.

*Cros-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 CSP language to EGL. This in turn means that you need to carefully consider which groups of parts you migrate together. Given the differences between Cross System Product and EGL and the need for cross-part migration, this migration is a major undertaking and needs to be carefully planned.

In addition to determining which parts you need to migrate as a group, 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 participate in the pilot project.
Select a small subset of your source code to use in the pilot project. During the pilot project, use this small subset to verify your environmental setup and your library management procedures and tools.

Review the prerequisites for the EGL developer product that you plan to use. In addition, review the prerequisites for your runtime environment. If you plan to generate COBOL for the z/OS environment, be sure to review the prerequisites for the EGL build server and EGL runtime server products. If you plan to generate Java for a workstation environment, be sure to review the prerequisites for the developer product that you plan to use.

Make key decisions about the scope of the pilot project. For example:

- Determine if you can freeze the CSP code development and maintenance during the actual migration. This technique enables you to migrate just the production level of source code. If you cannot freeze the CSP code development and maintenance, be sure to include the following tasks in your pilot project:
  - Migration of your work-in-process MSLs to VisualAge Generator and from there to EGL.
  - Developing and testing procedures for dual maintenance of common (shared) parts.
  - Choice of a backend source code repository.

- Build a task list, resource assignments, and schedule for the pilot project.

- Confirm that all workstations have Windows NT or Windows XP installed and that they have enough hard drive space and memory.

Obtain education for the team that will run the pilot project:

- EGL development environment
- EGL language
- CSP to VisualAge Generator migration tools
- VisualAge Generator to EGL migration tools
- Your new source code repository

Run the pilot project plan to perform the following tasks:

- Install Windows NT or Windows XP.
  
  **Note:** Although EGL runs on other platforms, VisualAge Generator Developer and the VAGen to EGL Migration Tool only run on Windows operating systems.

- Install the EGL developer product for the pilot team, and be sure to install on a machine that has regional settings that match the language you use when you develop your CSP programs. For example:
  - If you develop your CSP programs using CSP/370AD in German, you should install your EGL developer product on a German machine. This ensures that the comma is used as a decimal point and that German umlaut characters are migrated correctly.
  - If you develop 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.

- Install either VisualAge for Java or VisualAge for Smalltalk on the same workstation as EGL. If only one workstation will be used for migration, install EMSRV and the Java repository or Smalltalk manager on the same workstation as EGL. If you use VisualAge for Smalltalk on a Windows XP operating system, you must install EMSRV as a service. See “Determining which VAGen product to use” on page 29 for recommendations on whether you should use Java or Smalltalk. See “Appendix F. Installing EMSRV” on page 94 for details about installing EMSRV. This paper assumes that you use VisualAge for Smalltalk.

- Install VisualAge Generator Version 4.5 with Fix Pack 5 on the same workstation as EGL. Review the *VisualAge Generator to EGL Migration Guide*, Appendix F “Situations where
incorrect External Source Format causes problems in creation of EGL.” for additional VisualAge Generator APARs that might be necessary for your specific situation.

- Install DB2® on the same workstation as EGL. DB2 is required for the VAGen to EGL migration database. DB2 Version 8.1.15 (or the equivalent Version 8.2.8) is required for migrating from VAGen on Java. DB2 Version 8.1.15, Version 8.2.8, or Version 9 can be used for migrating from VAGen on Smalltalk.

- Determine how to organize your source code in EGL. Map this organization to the equivalent VAGen organization. See “Determining how to organize your source code for EGL” on page 15 for considerations.

- Run the VAGen MSL Migration Assistance Tool to migrate your pilot set of code from CSP to VAGen Version 4.5. See “Migrating your source code from CSP to VAGen” on page 29 for information related to running this tool.

- Complete the migration into VisualAge Generator so that your code is ready for migration into EGL. See “Manual steps between the tools” on page 37 for details.

- Run the VAGen to EGL Migration Tool for the pilot set of code. See “Migrating your source code from VAGen to EGL” 38 for information related to running this tool.

- Correct any errors in the Problems view that result from the migration. See “Additional changes to make by hand” on page 43 for details.

- Test the source code in the EGL development environment.

- Plan and install the connectivity required to use the EGL debug facility. If you use any of the following capabilities when you test your CSP applications using the CSP Interactive Test Facility (ITF), you need to plan how you will achieve comparable testing capabilities:
  - Non-CSP programs that you need to call from ITF
  - Access to DB2 database
  - Access to DL/I databases
  - Access to VSAM files
  - Access to special facilities that are only available if you use CSP/370AD in the CICS environment. These facilities include features such as writing to temporary storage queues, purging the queue, and using the CREATX service routine.

See the sections “Special techniques for debugging in EGL” on page 50 and “Considerations for the EGL debugger and z/OS host connectivity” on page 53 for details.

- Create your build parts for debug. This includes the build descriptor options, linkage options, and resource associations parts that you need for debug. Some of the information might be available in your CSP generation options, linkage table files, or batch generation command files or stored in the MSL program members. See “Appendix A. Converting CSP control information” on page 61 for details.

- Test your source code using the EGL debug facility. Be sure to test each type of connectivity to the z/OS host. In addition, you need to be aware of certain changes inherent in moving from host-based development to workstation-based development. For example, the collating sequence is different. See the sections “Special techniques for debugging in EGL” on page 50 and “Considerations for the EGL debugger differences from the z/OS host” on page 57 for details.

- z/OS COBOL target environments:
  - Install and enable TCP/IP. TCP/IP is the only method for transferring outputs of COBOL generation to the z/OS host.
  - Install prerequisites for the EGL build server and EGL runtime server products, including any changes to your COBOL compiler and runtime.
• Install the EGL build server and EGL runtime server products. See “Installing the EGL server product” on page 59 for details.
• Install the latest PTFs for the EGL build server and EGL runtime server products.
• Create a new set of libraries to contain the outputs of COBOL generation and the results from the build server.
• If you use CICS, create a new CICS region for testing the EGL-generated COBOL. Similarly, if you use IMS, create a new IMS region for testing. This technique avoids accidentally intermixing your CSP-generated code with the EGL-generated code and enables you to continue maintaining the CSP 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 “Appendix B. Converting CSP preparation templates and procedures” on page 84 for the correspondence between CSP preparation templates and procedures and the EGL build scripts and pseudo JCL.

Java target environments. Review the following sections in the VisualAge Generator to EGL Migration Guide, Chapter 10 “Language and runtime differences”:
• Differences in SQL support
• Differences in debug
• 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. Even though you did not generate C++ in CSP, much of the material with respect to name resolution rules, resource associations, SQL, and EZE special data words applies to changing from generating COBOL to generating Java.

• Generate and prepare your programs. We highly recommend that you generate 100% of your programs, FormGroups, and DataTables. This helps ensure that all your source code migrated successfully. See “Generating programs, DataTables, and FormGroups” on page 59 for rules that govern when you must regenerate. Perform the following tasks to generate your code:
  • Review “Migrating your generation information” and “Storing control information” for information about how to organize your control parts.
  • Create your build parts for generation. This includes the EGL build descriptor options, linkage option, and resource association parts. See “CSP generation options” on page 61, “Linkage table” on page 70, and “Resource associations” on page 76 for details on converting your CSP information to EGL.
  • Convert your CSP bind commands. See “Bind commands” on page 78 for details.
  • Convert your CSP link edit commands from CSP. See “Link edit commands” on page 80 for details.
  • Convert your CSP runtime JCL templates. See “Appendix C. Converting CSP runtime templates” on page 87 for details.
  • If you modified the CSP reserved words, create an EGL reserved words file. See “Appendix D. Converting CSP reserved words file” on page 90 for details.
  • Optionally, set up 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.
  • Generate the code for your runtime environments.
• Testing. We highly recommend that you test 100% of your generated programs. This helps determine if the migration of the programs resulted in any behavioral changes. As a minimum, you should test at least a representative sample of your generated programs to ensure you understand any runtime differences. Refer to the VisualAge Generator to EGL Migration Guide, Chapter 10 “Language and runtime differences” for a list of differences.

• Create library management processes:
  • Select and install a source code repository, including access from the developer workstations.
  • Define change management procedures that work with your corporate culture and your selected source code repository.
  • Develop any tools you need for your change management procedures, including the following processes:
    • Check-in 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.
  • 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, FormGroups, DataTables, and programs for each target environment. Also test your procedures for adding common code, FormGroups, 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.
  • Refine your library management procedures and tools based on the results of the pilot change cycle.
  • Plan and test backup and recovery procedures for your source code repository.

• If you will not be able to migrate all of the source code at one time, develop and test procedures for dual maintenance of common (shared) parts. See “Considerations for dual maintenance of source code” on page 50 for the possible techniques.

• Review key decisions based on the pilot project. Depending on the results of the pilot project and the changes to key decisions, rerun the pilot project with a new set of key decisions.

• Document the findings of the pilot project, including:
  • Any code problems that were found that had to be fixed so that if they occur again during the migration of other programs you remember how to fix them. This information is particularly important if you are changing runtime environments. Also include changes that can be made in the CSP code so that if you need to migrate the application several times during the pilot project, you only need to make the change once in CSP.
  • Information that developers might need to create their personal build descriptor parts.
  • References to sections of the VisualAge Generator to EGL Migration Guide or this paper that are particularly useful for your developers based on the problems you encountered during the pilot project.
  • Changes in runtime behavior that your end users will notice after migration.
  • Final library management and change control process.

• Build a task list, resource assignments, and schedule for the actual migration based on the findings from the pilot project.

• Execute your migration plan.

• Provide education for the remaining developers:
  • EGL development environment
  • EGL language
• Your source code organization in EGL, including how the code is structured into EGL projects, packages, and files
• Changes, if any, required to the CSP or EGL source code based on findings of your pilot project
• Your new source code repository
• Your new library management and change control process, including procedures for dual maintenance of common parts if that is necessary
• Your new generation process
• Mentoring, as needed, during the first few weeks of development

Due to the variety of activities required for the migration project, you need people with various skills. Table 2 provides a place to record key contacts in your organization that might be needed for the migration project depending on your runtime environments, database access, and so on.

<table>
<thead>
<tr>
<th>Support Needed Person</th>
<th>Support Needed Person</th>
<th>Phone Number</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cross System Product system administrator</td>
<td>Cross System Product build administrator / librarian</td>
<td>PC coordinator / expert</td>
<td>LAN administrator</td>
</tr>
<tr>
<td>SQL database administrator</td>
<td>DL/I database administrator</td>
<td>DFM systems programmer (if using remote VSAM)</td>
<td>RACF® authorization</td>
</tr>
<tr>
<td>COBOL systems programmer</td>
<td>CICS systems programmer</td>
<td>IMS systems programmer</td>
<td>VTAM® systems programmer</td>
</tr>
<tr>
<td>TCP/IP systems programmer</td>
<td>VisualAge Generator Developer installation</td>
<td>Rational Business Developer installation</td>
<td>EGL COBOL generator installation</td>
</tr>
<tr>
<td>EGL build server installation</td>
<td>EGL build server customization</td>
<td>EGL runtime server installation</td>
<td>EGL runtime server customization</td>
</tr>
<tr>
<td>Source code repository installation</td>
<td>Source code repository customization</td>
<td>Source code repository administrator</td>
<td>TSO Logon procedures</td>
</tr>
<tr>
<td>Access to locked areas for the servers</td>
<td>Canceling a TSO session</td>
<td>APAR support (installing)</td>
<td>Reporting problems to IBM</td>
</tr>
</tbody>
</table>
Determining how to organize your source code for EGL

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

- The differences between the CSP, 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 limitations and tradeoffs of various organization techniques in EGL.
- The capabilities provided by the migration tools to help you achieve the final organization you want in EGL.

The above topics are covered in the sections that follow.

Differences in product capabilities for organizing your code

CSP and EGL provide very different methods for organizing your source code. CSP uses MSLs. EGL uses projects, packages, and files. The VAGen method for organizing your source code falls between the two extremes, but is closer to the EGL technique. To be able to use the migration tools effectively, you need to understand the different capabilities the three products provide for organizing your source code.

**CSP code organization**

In CSP, the source code is organized into MSLs. For example, you might have mslA that contains all the code unique to SubsystemA, mslB that contains all the code unique to SubsystemB, and mslCOMMON that contains all the code that is shared by multiple subsystems.

You use the MSL concatenation sequence to determine which MSLs to consider when you develop, test, or generate your CSP applications, tables, or map groups. A member name must be unique within an MSL, but you can have duplicate part names within the MSL concatenation sequence. CSP uses a “first found” approach when determining which of several definitions of the same member name to use. The first definition that is found based on the MSL concatenation sequence is the definition that CSP uses during test or generation.

When you make a change, the updated member is always stored in the first MSL in the concatenation sequence (the read/write MSL). To keep track of changes, each developer might have a unique read/write MSL. You might also have a staging MSL to coordinate changes made by several developers, one MSL for each level of testing, and a production MSL. Typically, the developer read/write MSL, staging MSL, and test MSLs are delta MSLs and only have the changed parts. The production MSL contains the complete set of parts that are currently in production. An alternative technique for tracking changes is to have a complete set of parts in the each of the staging, test, and production MSLs.
**VAGen code organization**

In VisualAge Generator on Smalltalk, the source code is organized into Smalltalk configuration maps and Smalltalk applications. For example, you might have ConfigurationMapA that lists all the applications that contain code unique to SubsystemA, ConfigurationMapB that lists all the applications that contain code unique to SubsystemB, and ConfigurationMapCommon that lists all the applications that contain code that is shared by multiple subsystems.

*Note:* In VisualAge Generator on Java, the source code is organized into Java projects and Java packages.

The Smalltalk applications provide a more detailed organization of your source code. For example, within ConfigurationMapA, you might have one Smalltalk application for common parts that are used only within SubsystemA. This application might include data item, record, table, and function parts that are used by multiple programs. Other Smalltalk applications might contain programs and map groups, with one application containing a program and all the parts that are unique to that program.

The Smalltalk applications 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.

*Note:* VisualAge Generator on Java provides Project List Parts as the comparable concept to Smalltalk configuration maps. In VisualAge Generator on Java, you can load projects that contain duplicate parts names into your workspace, but you cannot test or generate if there are duplicate parts in your workspace. You also cannot migrate using the VAGen to EGL Migration Tool 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 ENY manager. VisualAge Generator uses a technique called versioning to freeze the code at a known level. The ENY 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 ENY 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 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 contains all the code unique to SubsystemA, ProjectB that contains all the code unique to SubsystemB, and ProjectCommon that contains all the code that is shared by multiple
subsystems. 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 packages and files provide a more detailed organization of your source code. For example, within the *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.

You might use one or more packages to contain the programs, splitting the packages based on functional areas within *SubsystemA*.

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.

**Note:** The combination of the EGL Build Path and *import* statements is similar to the MSL concatenation sequence in that it provides the set of parts to consider for development, debug, and generation.

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 backend source code repository to retain multiple versions of the code. The source code repository provides tools such as checkout/check-in, version control, and comparison tools so that you can compare what is in your workspace with other versions of the code in the 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.

### Considerations for structuring your EGL code

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 CSP MSLs might already be organized along functional lines. For example, you might have *MSLA* that contains all the code unique to *SubsystemA*, *MSLB* that contains all the code unique to *SubsystemB*, and *MSLCOMMON* that contains all the code that is shared by multiple subsystems. Therefore, as a starting point, consider migrating one CSP MSL to become one EGL project.
However, you need to consider other factors that might cause you to alter the “one MSL becomes one EGL project” strategy.

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. An EGL build is roughly equivalent to the CSP validation process, except that a build generally occurs for everything in your workspace. 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. The build must occur before you attempt to debug or generate your code.
- EGL generation time and time in the build server. The build server processing is roughly equivalent to the CSP preparation process and is not related to EGL project, package, or file size.

Table 3 shows the limitations and tradeoffs of project, package, and file sizes.

<table>
<thead>
<tr>
<th>Table 3. Limitations and Tradeoffs of Project, Package, and File Sizes</th>
<th>Limitations</th>
<th>Advantage of Smaller</th>
<th>Advantage of Larger</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Project</strong></td>
<td>Maximum of 1500 projects</td>
<td>If you only need a subset of the projects, then the workspace is smaller so the build time might be quicker</td>
<td>Fewer projects in the EGL Build Path</td>
</tr>
<tr>
<td></td>
<td>Some source code repositories might have a smaller maximum</td>
<td>Smaller projects load quicker from the source code repository</td>
<td>Less likelihood of cycles and fewer cycles in the EGL Build Path</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Minimizes the chance that two developers need to make changes to same project at the same time</td>
<td>Fewer projects to scroll through</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Fewer projects to load from the source code repository</td>
<td>Fewer projects to load from the source code repository</td>
</tr>
<tr>
<td><strong>Package</strong></td>
<td></td>
<td>An import statement of the form import package.* includes fewer part names so the build time might be quicker</td>
<td>Fewer import statements in each file</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Minimizes the chance that two developers need to make changes to same package at the same time</td>
<td>Fewer packages to scroll through</td>
</tr>
<tr>
<td><strong>File</strong></td>
<td>Performance to open or save a file degrades for large file sizes</td>
<td>When you save a file, there are fewer parts to check for changes so the build time might be quicker</td>
<td>Fewer files to scroll through</td>
</tr>
<tr>
<td></td>
<td>No maximum size, but practical size is &lt;200K</td>
<td>With good naming conventions, you can quickly find the file containing a specific part</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Minimizes the chance that two developers need to make changes to same file at the same time</td>
<td></td>
</tr>
</tbody>
</table>

18
Given the tradeoffs in Table 3, you should avoid creating giant EGL projects, packages, or files. Conversely, you should avoid creating lots of tiny EGL projects or packages. Both extremes can adversely affect performance. For example, if you create a file that contains 30,000 data items, you probably have very large file that can take several minutes to open or save. Conversely, if you create one file for each of the 30,000 data items, you have more files than you want to scroll through. You should use a “middle of the road” approach to organizing the data items. For example, if the first character of the data item names is evenly split across the alphabet, then you might create multiple files, with each file containing all the items that start with the same character. This technique creates smaller files that are quicker to open, but minimizes the total number of files you have to deal with. See “Sample organization techniques” on page 23 for additional suggestions of how to organize your code.

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 repository you select:
  - Whether the source code repository supports check out and check-in 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.
- You need to establish naming conventions for the new EGL part types such as functions and control parts. In addition, you need to establish naming conventions for EGL projects, packages, and files. See “Establishing naming conventions” on page 25 for details.
- You need to decide how to handle duplicate parts. See “Resolving duplicate CSP member names” on page 27 for details.

Using the migration tools to structure your code

To migrate from CSP to EGL, you must use two migration tools, with a manual step in between the two tools as follows:

- The VAGen MSL Migration Assistance Tool, which helps structure the code from your MSLs into Smalltalk applications.
- A manual step to create Smalltalk configuration maps that specify the Smalltalk applications and other required configuration maps to load as a group into your Smalltalk image.
- The VAGen to EGL Migration Tool, which converts all the source code that is contained in high-level Smalltalk configuration maps into EGL projects, packages, and files. This tool works in three stages:
• Stage 1 extracts the source code specified by a high-level Smalltalk configuration map from the VAGen repository and loads it into a DB2 database. Stage 1 also determines the final EGL project, package, and file where each part will be placed during Stage 3 of migration.
• Stage 2 reads the VAGen source code from the DB2 database, converts it to EGL, and stores the EGL source code into the DB2 database.
• Stage 3 reads the EGL source code from the database and creates the EGL projects, packages, and files as determined in Stage 1.

The VAGen MSL Migration Assistance Tool is designed to help you organize your code along functional lines. The VAGen to EGL Migration Tool is designed to preserve the organization you created with the VAGen MSL Migration Assistance Tool. In general, one Smalltalk configuration map becomes one EGL project and one Smalltalk application becomes one EGL package. In addition, the VAGen to EGL Migration Tool automatically splits programs, DataTables, and FormGroup parts into their own files. You can tailor Stage 1 of VAGen to EGL Migration Tool in several ways:
• Consolidate several Smalltalk configuration maps into one EGL project or consolidate several Smalltalk applications into one EGL package. Refer to the Project Consolidation white paper for details on this technique.
• Split large CommonParts.egl files or UnusedParts.egl files into smaller files based on part type or part name. Refer to the VisualAge Generator to EGL Migration Guide, Chapter 5 “Stage 1 – Extracting from Smalltalk”, section “Customizing the Stage 1 migration tool” for details on using this built-in customization for the Stage 1 tool.
• Move all data item parts to a new EGL project. This customization is particularly useful if you want to eliminate the use of global data items during your migration. Refer to the VisualAge Generator to EGL Migration Guide, Chapter 5 “Stage 1 – Extracting from Smalltalk”, section “Customizing the Stage 1 migration tool” for details on using this built-in customization for the Stage 1 tool.

Overview of the VAGen MSL Migration Assistance Tool

To migrate your source code from CSP MSLs to VAGen, use the VAGen MSL Migration Assistance Tool. This tool provides the following features:
• A tool that enables you to import a CSP External Source Format file to create a pseudo-MSL structure on the workstation. This step also converts the code from the CSP members to VAGen
Automatic conversions during migration to VisualAge Generator” on page 33 for details.

- A tool that enables you to define an MSL concatenation sequence to consider for conversion to VisualAge Generator.
- A way to identify which types of parts you want to consider for migration. For example, you can choose to initially work with just the program parts and then later review all parts in the MSL concatenation sequence to ensure that all the parts have been processed.
- A sandbox area in which you can see your proposed Smalltalk applications without committing them to the ENVY manager. The sandbox enables you to play around with your organization, for example, in the following ways:
  - See the number of parts in your proposed applications, the parts contained in each application, dependencies between the applications, and so on.
  - Move parts from one application to another.
  - Combine applications.
  - Rename applications.
- A way to specify which parts to migrate to the sandbox. When you select a part to migrate to the sandbox, the migration tool does the following things:
  - The tool automatically moves all the associates of the part. See “Member associations” on page 34 for information about which part types have associates.
  - If any of the associates of the part are already in the sandbox in a different application, the tool automatically creates a new ApplicationNode to indicate that the parts are shared. This technique helps you to identify parts that are shared by several applications.
  - Once you start to build up applications that contain common code, you can specify that you do not want to have an ApplicationNode created if a newly migrated associate is already in a specific application. You do this by marking the application as unexplodable.
  - Information about duplicate parts in the MSL concatenation sequence so that you can determine which of the duplicate parts you want to migrate.
- A way to commit your final Smalltalk applications into the ENVY manager.

See “Sample organization techniques” on page 23 for additional suggestions of ways to organize your code using the VAGen MSL Migration Assistance Tool.

See “Migrating your source code from CSP to VAGen” on page 29 for more information about how to prepare for and run the VAGen MSL Migration Assistance Tool and a table showing the chapters to review in the VisualAge Generator Migration Guide. When you review this book, you will find that VisualAge Generator has a “rule of thumb” that you should have no more than 600 to 700 parts of the same type in a Smalltalk application. For example, you can have 700 records and 700 data items in the same Smalltalk application. However, according to the “rule of thumb”, you should not have 1000 records and 1000 data items in the same Smalltalk application. Because you are just passing through VisualAge Generator, you can relax this rule somewhat. There will still be a VAGen performance degradation, but this is not a long-term issue. However, given that there is a performance degradation, it probably best to stay under 2000 parts of the same type and under 10,000 total parts in the Smalltalk application. Do not try (as some customers have attempted) to put 30,000 parts of the same type into a single Smalltalk application or 80,000 parts of various types into a single Smalltalk application. The VAGen MSL Migration Assistance Tool and VAGen on Smalltalk simply were not designed for extremely large Smalltalk applications.

If you must split a Smalltalk application due to VAGen performance reasons, use similar application names (for example, use application names AcctRecApp1 and AcctRecApp2). This technique enables you to easily tailor Stage 1 of the VAGen to EGL Migration Tool to merge the Smalltalk
applications into a single EGL package when migrating from VisualAge Generator to EGL. Refer to the Project Consolidation white paper for details of how to tailor Stage 1 to consolidate EGL projects and packages.

If you do group large numbers of data items, records, or functions into a single large Smalltalk application or if you use the technique in the Project Consolidation white paper to merge several Smalltalk applications into a single EGL package, then you should use the techniques described in the VisualAge Generator to EGL Migration Guide to split common parts files based on the part type and part name. This technique helps to avoid very large files.

**Overview of the VAGen to EGL Migration Tool**

To migrate your source code from VAGen to EGL, use the VAGen to EGL Migration Tool. This tool converts the source code in the following series of stages:

- **A Stage 1 tool** that runs on either VAGen on Smalltalk or VAGen on Java. The Stage 1 tool does the following things:
  - Enables you to specify preferences, including the following information:
    - The Smalltalk configuration maps you want to migrate.
    - Which Smalltalk configuration maps or applications to consider as common code.
    - The level of diagnostic information you want to capture.
  - Extracts the source code from the VAGen repository into a DB2 database.
  - Assigns programs, tables, and map groups to unique files. Assigns maps to the same file as the corresponding map group. Assigns data items, records, and functions to files based on their usage. Parts that are only used by one program are assigned to the same file as the program. Parts that are used by multiple programs are assigned to CommonParts.egl files. Parts that are not used by any program are assigned to UnusedParts.egl files.
  - In general, the tool converts each Smalltalk configuration map to an EGL project and each Smalltalk application to an EGL package. However, the tool is shipped as sample code so that you can tailor it to better server your needs. The VisualAge Generator to EGL Migration Guide and the Project Consolidation white paper provide examples of tailoring the Stage 1 tool.

- **A Stage 2 tool** that runs on the EGL developer product. The Stage 2 tool does the following things:
  - Enables you to specify preferences, including the following information:
    - Prefixes and suffixes used to create EGL names. For example, the Stage 2 tool uses a prefix to convert a VAGen name that is an EGL reserved word to a valid EGL name. The Stage 2 tool uses a suffix to create the name of an EGL record that contains the level 77 items from a VAGen working storage record.
    - Whether you want to convert shared data items to primitive definitions. In CSP, this is equivalent to converting from using global data items to local data items.
    - Whether you need a decimal comma converted to a decimal point.
  - Reads the VAGen source code from the DB2 database, converts it to EGL, and stores the EGL source code into the DB2 database. This conversion step uses cross-part migration.

- **A Stage 3 tool** that runs on the EGL developer product. The Stage 3 tool does the following things:
  - Reads the EGL source code from the database.
  - Creates EGL projects, packages, and files.
- Sets the EGL Build Path for each project to specify other projects that contain parts that are referenced by the current project.
- Creates the `import` statements required in each file to specify other packages within projects in the EGL Build Path that contain parts that are referenced by the current file.
- Stores the EGL source code for each part into the correct file based on the assignment done by the Stage 1 tool.

See “Migrating your source code from VAGen to EGL” on page 38 for a table showing the chapters to review in the *VisualAge Generator to EGL Migration Guide*. Be sure read Chapter 2 “Migration Tool Philosophy”, section “Techniques used by the VisualAge Generator to EGL Migration Tool” and subsection “Placing parts in EGL files” for important information about how the VAGen to EGL Migration Tool works. You need to understand this information so that you can create the appropriate organization when you migrate your source code from CSP to VAGen.

**Sample organization techniques**

After you plan an overall EGL project and package organization strategy, you can use the VAGen MSL Migration Assistance Tool to help you determine the specific EGL package where each part is to be placed. The Smalltalk applications that you create with the VAGen MSL Migration Assistance Tool become the EGL packages. After you commit your Smalltalk applications into the ENVY manager, you define Smalltalk configuration maps to specify how to group these applications. Each Smalltalk configuration map corresponds to an EGL project. The applications that you include in the Smalltalk configuration map correspond to the EGL packages that you want to include in the EGL project.

Here are some examples of how you might convert an MSL to one EGL project, but split the MSL into several EGL packages using the VAGen MSL Migration Assistance Tool:

- A DBA MSL contains data items and record definitions. This MSL might be small enough to become a single EGL package. However, for a large MSL, you might want the final EGL organization to use smaller packages by dividing the MSL in any of the following ways:
  - One EGL package for the data items and records that are shared by multiple subsystems and a separate package for each subsystem that contains the data items and records that are unique to that subsystem.
  - One EGL package for the data items and records used in a particular runtime environment.
  - One EGL package for a group of closely related records, where you want smaller packages than either of the previously mentioned techniques.

The VAGen MSL Migration Assistance Tool can help you achieve any of the above organizations. For example, to organize the DBA MSL by subsystem, use the tool to select all the records for a subsystem. When you move the records to the sandbox, the tool automatically places the data items that are associates of the records into the same Smalltalk application (for example, application `SubsystemAData`). When you move the records for the next subsystem to a different application in the sandbox (for example, application `SubsystemBData`), the tool automatically moves any data items that are shared with the records in the first subsystem to a new ApplicationNode. You can rename this ApplicationNode (for example, `CommonDataItems`) and then mark the application as unexplodable. When you move the records in the third subsystem to the sandbox, the tool creates additional ApplicationNodes as needed for parts shared with the two previous subsystems in `SubsystemAData` and `SubsystemBData`, but does not create an ApplicationNode for items that are already in the
unexplodable CommonDataItems application. You can merge the new ApplicationNodes into the CommonDataItems application. This technique enables you to gradually build up the CommonDataItems application so that it contains all the data items that are shared across multiple subsystems.

After you commit the applications to the ENVY manager, create the Smalltalk configuration maps in the following way:

- If you want one EGL DBA project, then create a Smalltalk configuration map that contains all the Smalltalk applications created from the DBA MSL.
- If you now want to group everything by subsystem, create a Smalltalk configuration map for each subsystem and include the SubsystemXData application created for that subsystem from the DBA MSL. Then create a Smalltalk configuration map to contain the CommonDataItems application and any other applications that you created for records that are shared by multiple subsystems.

An MSL contains common code that is shared by multiple subsystems. This MSL might be small enough to become a single EGL project and package. However, for a large MSL, you might want the final EGL organization to be done along functional lines, as in the following examples:

- Error and message handling.
- Security support.
- Audit log or journal support.
- Tools only used during development, not in the production system.

To organize along functional lines, use the VAGen MSL Migration Assistance Tool to select all the programs for one functional area and specify the same Smalltalk application. If several functional areas share common parts, as you move the programs for the next functional area to the sandbox, the tool automatically creates ApplicationNodes to contain the parts shared with previously migrated functional areas. You can merge all the ApplicationNodes into one Smalltalk application if you want all the shared parts together. Alternatively, you can force a specific part into a specific Smalltalk application. For example, you might want a record that is passed as a parameter to an error handling program to be stored in the same application with the error handling program even though the record is used by numerous other programs.

After you commit the applications to the ENVY manager, create one or more Smalltalk configuration maps, depending on whether you want to group all the functional areas from the original common MSL into a single EGL project or whether you want one EGL project for each functional area.

In addition to the techniques mentioned above, you might have your code already structured along functional lines, but that organization is not reflected in your MSLs. In this case, you might want the final EGL organization to reflect an underlying (but currently hidden) structure, as in the following example:

- You might have used a naming convention to help you identify parts that belong to a system or subsystem. For example, ACCTR might be the first 5 characters of all the parts that are unique to the accounts receivable system. ACCTP might be the first 5 characters of all the parts that are unique to the accounts payable system. In this case, depending on the total number of parts, you might want the final EGL organization to be done in any of the following ways:
  - For a very large number of parts, include all the parts that start with ACCTR in one EGL project and all the parts starting with ACCTP in a different project. Split each project into
multiple EGL packages based on components or functional areas within the accounts receivable or accounts payable system.

- For a large number of parts, include all the parts that start with **ACCTR** in one EGL project with one EGL package and all the parts that start with **ACCTP** in another EGL project with one EGL package.
- For a smaller number of parts, include all the parts in a single EGL Accounting project, and split the parts starting with **ACCTR** and **ACCTP** into different EGL packages within that project.
- For an even smaller number of parts, include all the parts in a single EGL Accounting project with one EGL package.

You can use the VAGen MSL Migration Assistance Tool with any of the above approaches. The only difference is the number of Smalltalk applications (eventual EGL packages) that you create in the sandbox. When you create the Smalltalk configuration maps, include the applications based on the corresponding EGL project and package structure that you want to achieve.

- For a menu system, if there are 10 options on the main menu, depending on the total number of parts, you might want the final EGL organization to be done in any of the following ways:
  - For a large number of parts, include all the programs that are used when the user selects option 1 in one EGL package, all the programs that are used when the user selects option 2 in another EGL package, and so on. Then put the packages into different EGL projects.
  - For a smaller number of parts, include all the programs that are used when the user selects option 1 in one EGL package, all the programs that are used when the user selects option 2 in another EGL package, and so on. Then include all the packages in the same EGL project.
  - For an even smaller number of parts, include all the programs in the same package. For example, if you have a small set of programs that are frequently called by other programs, you might group this small set of programs into a single EGL package.
  - If you use any of the above techniques, consider called programs as well as programs that are transferred to using the XFER or DXFR statements.
  - If you have multiple layers of menus, you could use this technique at any of the menu levels depending on how larger or small you want your final EGL projects and packages to be. If you want larger EGL projects and packages, use the technique at the higher levels of menus. If you want smaller EGL projects and packages, use the technique at the lowest level of menus.

You can use the VAGen MSL Migration Assistance Tool with any of the above approaches. The only difference is the number of Smalltalk applications (eventual EGL packages) that you create in the sandbox. When you create the Smalltalk configuration maps, include the applications based on the corresponding EGL project and package structure that you want to achieve.

**Establishing naming conventions**

You probably already have naming conventions for parts like programs, tables, maps, and records that you are migrating from CSP. You can continue to use your existing naming conventions for these parts because, for the most part, the VAGen MSL Migration Assistance Tool and the VAGen to EGL Migration Tool retain the existing part names. An exception to this occurs if the part name conflicts with an EGL reserved word or starts with the # or @ sign. In this case, the VAGen to EGL Migration Tool does the following things:
• Changes the part name for data items, records, functions, and maps by adding a prefix to the part name.
• Renames PSBs because these CSP parts are converted to EGL PSBRecords.
• Does not change the part name for programs, DataTables or FormGroups because these parts frequently have references from non-EGL programs or the runtime environment.

You should establish naming conventions for the following things:
• Functions. Because functions are created from both CSP processes and statement groups, you might want to establish a naming convention for function parts that is similar to your CSP conventions for processes and statement groups. For example, if the sixth position of the CSP member name is a P or an S to indicate a process or statement group, you might want to use an F to indicate a function in the sixth position of any new EGL function names.
• Control parts. The EGL build descriptor, linkage options, resource associations, bind control, and link edit parts did not exist in CSP. Therefore, you need to create naming conventions for these parts. For suggestions on build descriptor parts, linkage option parts, and resource associations parts, see “Storing control information” on page 45. For the rules governing names for bind control parts, see “Bind commands” on page 78. For the rules governing names for link edit parts, see “Link edit commands” on page 80.
• EGL project names. Consider the following factors:
  • EGL project names can be mixed case. Typically, the first character of the project name is an uppercase, alphabetic character. For the rest of the name, it is customary to capitalize the first letter of each word in the name. For example, AccountingSystem.
  • If you have several subsystems, name the projects so that you can distinguish the subsystem and have all the projects for a single subsystem grouped together in the Project Explorer view. For example, if you have an Accounting system and a Payroll system you might choose to use the following project names: AccountingSystemAccountsReceivable, AccountingSystemAccountsPayable, and PayrollSystemHourly, PayrollSystemSalary, and PayrollSystemTaxReports.
  • Consider the need for horizontal scrolling in the Project Explorer view. You might want to use shorter names to avoid horizontal scrolling. For example, instead of AccountingSystemAccountsReceivable, consider using alternatives such as AcctReceivable and AcctRec.
  • If you want common projects to appear at the bottom of the list, you might choose to use the constant ZZZ as a prefix. For example, consider using AcctRecZZZCommonXxxxx (where Xxxxxx is other meaningful information) for common projects that are shared by multiple functional areas within the Accounts Receivable system and ZZZCommonXxxxx for common projects that are shared across multiple systems.
  • After you establish a naming convention for the EGL projects, use this naming convention when you create Smalltalk configuration maps during the migration from CSP to VisualAge Generator. When you migrate from VisualAge Generator to EGL, the VAGen to EGL Migration Tool converts each Smalltalk configuration map to an EGL project. See “Relating your EGL naming scheme to the sandbox” on page 35 for details.
• EGL package names. Consider the following factors:
  • EGL package names follow Java package naming conventions.
    • Typically package names are all lower case, with the words separated by periods. Each word in the dot notation corresponds to a subfolder. For example, accounts.receivable.functionalarea1 corresponds to a folder structure of \EGLSource\accounts\receivable\functionalarea1.
• EGL reserved words cannot be used as a word in the package name. For example, `case.subsystem1` is an invalid package name, but `cases.subsystem1` is valid. For a list of the EGL reserved words, refer to VisualAge Generator to EGL Migration Guide, Appendix A “Reserved words”, subsection “EGL reserved words”.
• You must not use Java reserved words as a word in the package name. For example, using `break.subsystem1` as a package name is not a good practice because `break` is a Java reserved word. For a list of the Java reserved words, refer to VisualAge Generator to EGL Migration Guide, Appendix A “Reserved words”, subsection “Java reserved words”.
• Consider the need for horizontal scrolling in the Project Explorer view. You might want to use shorter names to avoid horizontal scrolling. For example, instead of `accounts receivable.functionalarea1`, consider using alternatives such as `acct.rec.functionalarea1`, `acctrec.functionalarea1`, or `acctrec.area1`.
• If you want common packages to appear at the bottom of the list of packages within the EGLSource folder, you might choose to use `zzz` as a prefix. For example, within the accounts receivable project, consider using `acctrec.zzz.xxxxx` (where `xxxxx` is other meaningful information) for common packages. Within a common project, consider using `zzz.xxxxx` for the package names.
• If you think that you eventually might be developing Web applications, consider that you might need to have unique package names across a Web server. A typical naming convention is `com.companyID.systemID.subsystemID.otherInformation`. For example, the EGL product plug-in names are in the form: `com.ibm.etools.egl.xxxxx`.
• As a best practice, a given package name should only be used within a single EGL project.
• After you establish a naming convention for the EGL packages, use this naming convention when you create Smalltalk applications during the migration from CSP to VisualAge Generator. When you migrate from VisualAge Generator to EGL, the VAGen to EGL Migration Tool converts each Smalltalk application to an EGL project. See “Relating your EGL naming scheme to the sandbox” on page 35 for details.
• Version names. If your source code repository supports versioning, you should establish a naming convention for the versions. Using a default naming convention (for example, 1.0, 1.1, and so on) might be a simple way to start. Alternatively, you might want a naming convention that ties the version to a change request or defect number.
  
  Note: The version names you use in VisualAge Generator do not carry forward to EGL.

Resolving duplicate CSP member names

You should resolve (or at least understand) duplicate CSP member names before starting the migration to VisualAge Generator. Duplicates in CSP can arise in the following situations:
• A duplicate member was accidentally copied into the wrong MSL and was never deleted. In this case, you must determine the correct version of the member.
• Duplicate members have the same name, but different member types. This might require renaming one of the members and changing all references to it.
• The duplicate member is in a staging MSL or test MSL and represents work-in-progress. In this case, consider the following factors:
  • If you can move the work-in-progress members forward into production before you migrate, then this simplifies the migration project because you only need to migrate the production-level MSLs to EGL.
If you must migrate the members in the work-in-progress MSLs, consider migrating in the following way:

- Migrate the production MSLs to VAGen and EGL. Version the code in both VAGen and in your EGL source code repository.
- Then migrate the work-in-progress MSLs to VAGen and EGL.
- For details on migrating work-in-progress MSLs to VAGen, refer to VisualAge Generator Migration Guide, Chapter 27 “Migrating production and work-in-progress MSLs to VAGen 4.0 on Smalltalk” and Chapter 28 “VAGen on Smalltalk case studies based on various MSL structures”.
- For details on migrating multiple versions of code from VAGen to EGL, refer to the VisualAge Generator to EGL Migration Guide, Chapter 2 “Migration Tool Philosophy”, sections “Controlling the order for processing migration sets” and “Overwriting and merging files”.
- The duplicate member is in a staging MSL or test MSL, was used purely for testing, and was never intended for production. In this case, you might not want to migrate this member.
Determining which VAGen product to use

You can use either VisualAge Generator on Java or VisualAge Generator on Smalltalk. However, VisualAge Generator on Smalltalk is the recommended product. The Smalltalk version of the VAGen MSL Migration Assistance Tool is faster, more reliable, and easier to use than the Java version. In addition, if you shutdown VisualAge Generator and save your image, the contents of the migration tool are preserved. This makes it much easier to handle the migration using VAGen on Smalltalk. The main advantage of using the Java version of the VAGen MSL Migration Assistance Tool is that the terminology in the Java version of VisualAge Generator is similar to the final EGL terminology.

Preparing to migrate your CSP/370AD source code

Note: For information on how to export an External Source Format file in batch mode, refer to SAA® Cross System Product Commands and Utilities manual (SH23-6544), Chapter 1 “CSP/370AD Batch Commands”. For information on how to export an External Source Format file using the online utilities, refer to Chapter 5 “MSL Utilities” in the same book.

If you are migrating your source code from CSP/370AD, you need to complete the following steps to prepare your source code for migration:

1. When you download your External Source Format files to the workstation, a conversion from EBCDIC to ASCII occurs. Run a test to check that special characters are converted correctly. For example:
   a. Create a process member in CSP/370AD. Add a comment to the process that contains all the characters you can type on your keyboard, including all the special characters. Be sure to include the not equal sign (¬).
   b. Export this process to an External Source Format file.
   c. Download the file to the workstation.
   d. Import the file into VisualAge Generator.
   e. Open the process (VAGen function) with the Function Editor and verify that all the characters are correct.
      • If the only character that has a problem is the not equal sign, “Appendix E. Code page conversions”, subsection “Using the hptrules.nls file” on page 91. Repeat the steps starting with step (d) to verify that this resolves the problem.
      • If other characters have problems, you need to try a different download technique. Repeat the steps starting with step (c) to verify that the new technique resolves the problem.

2. Export an External Source Format file for an entire MSL from CSP/370AD. Be sure to save the exported External Source Format file for use in migrating resource association information to VisualAge Generator.

3. Download the export file to the workstation.

4. Repeat steps (2) and (3) for each MSL. Using a separate External Source Format file for each MSL preserves as much information as possible for organizing your source code. Potentially, each MSL becomes one EGL project.
Preparing to migrate your CSP/2AD source code

If you are migrating your source code from CSP/2AD, you need to complete the following steps to prepare your source code for migration:

1. When you move your External Source Format files from OS/2 to the Windows operating system, a code page conversion occurs for single-byte languages. Run a test to check that special characters are converted correctly. For example:
   a. Create a process member in CSP/2AD. Add a comment to the process that contains all the characters you can type on your keyboard, including all the special characters. Be sure to include the not equal sign (¬ or ^).
   b. Export this process to an External Source Format file.
   c. Transfer the file to the Windows operating system.
   d. Import the file into VisualAge Generator.
   e. Open the process (VAGen function) with the Function Editor and verify that all the characters are correct.
      • If the only character that has a problem is the not equal sign, see “Appendix E. Code page conversions”, subsection “Using the hptrules.nls file” on page 91. Repeat the steps starting with step (d) to verify that this resolves the problem.
      • If other characters have problems, you need to use a special conversion utility. For details, see “Appendix E. Code page conversions”, subsection “Changing from OS/2 to a Windows Operating System” on page 92. Repeat the steps starting with step (c) to verify that the new technique resolves the problem.

2. Export an External Source Format file for an entire MSL from CSP/2AD. Be sure to save the exported External Source Format file for use in migrating resource association information to VisualAge Generator.

3. Move the export file to the Windows operating system.

4. Repeat steps (2) and (3) for each MSL. Using a separate External Source Format file for each MSL preserves as much information as possible for organizing your source code. Potentially, each MSL becomes one EGL project.

Using the VAGen MSL Migration Assistance Tool

Regardless of whether your source code is from CSP/370AD or CSP/2AD, use the VAGen MSL Migration Assistance Tool to migrate the External Source Format file into VisualAge Generator. This tool converts the CSP members into VAGen parts. Assuming that you use VAGen on Smalltalk, refer to the VisualAge Generator Migration Guide, Part 4, Chapters 26 through 33 for details on how to run this tool.

The following is a list of enhancements that were made to VisualAge Generator prior to Version 4.5 that you can ignore in the VisualAge Generator Migration Guide:

• GUIs, Smalltalk views, packaging views, and non-visual classes
• VAGen Templates
• Library management using TeamConnection®

In addition, you can ignore information about the following topics:

• Information specific to migrating CSP Version 3.3 or earlier.
• Information specific to migrating VAGen Version 2.x, including information about sharing the workstation MSLs if you migrate to VAGen on OS/2. You must migrate to VAGen on a Windows operating system because the VAGen to EGL Migration Tool is only supported on Windows operating systems.
• Subapplications – do not use Smalltalk subapplications because they do not have any corresponding representation in EGL.
• Ownership – use the Library Supervisor as the owner for all configuration maps, applications, and part classes. Ownership information is ignored by the VAGen to EGL Migration Tool.
• Generating and testing your code in VAGen – wait until you complete the migration to EGL and only generate and test with the EGL code.

Use Table 4 as a guide to *VisualAge Generator Migration Guide*. Table 4 describes important topics and information, in addition to the topics listed above, that you should ignore.

**Table 4. Chapters in *VisualAge Generator Migration Guide***

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Description and Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>Comparing MSLs and ENVY on VAGen 4.0 on Smalltalk</td>
<td><strong>Ignore:</strong> The material from this chapter has been rewritten based on migrating MSLs to EGL and is included in this paper.</td>
</tr>
<tr>
<td>24</td>
<td>General migration considerations for VAGen 2.x and Cross System Product to Smalltalk</td>
<td><strong>Ignore:</strong> The material from this chapter has been rewritten based on migrating MSLs to EGL and is included in this paper.</td>
</tr>
<tr>
<td>25</td>
<td>Pre-migration checklist</td>
<td><strong>Ignore:</strong> The material from this chapter has been rewritten based on migrating MSLs to EGL and is included in this paper.</td>
</tr>
</tbody>
</table>
| 26      | The MSL Migration Assistance Tool on Smalltalk | Provides an overview of the VAGen MSL Migration Assistance Tool.  
**Important Topics:** “Building MSL directories” provides information about how to create pseudo-MSLs for use by the VAGen MSL Migration Assistance Tool.  
**Ignore:** Discussion of using the VAGen Version 2.x workstation MSLs directly. You must move CSP/2AD MSLs from OS/2 to the Windows operating system. |
| 27      | Migrating production and work-in-progress MSLs to VAGen 4.0 on Smalltalk | Provides an overview of how to migrate your MSLs based on whether they are production code or work-in-progress MSLs.  
**Ignore:** Discussion of using the VAGen Version 2.x workstation MSLs directly. You must move CSP/2AD MSLs from OS/2 to the Windows operating system. |
| 28      | VAGen on Smalltalk case studies based on various MSL structures | Provides an overview of how to migrate your MSLs based on various ways of structuring your code into subsystem and common MSLs.  
**Ignore:**  
• “MSLs containing code from VisualAge Generator Templates or BW*Wizard”. These technologies were not available for CSP.  
• “MSLs from marketing or other demonstrations” |
<table>
<thead>
<tr>
<th>Chapter Title</th>
<th>Description and Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>29 Running the MSL Migration Assistance Tool on Smalltalk</td>
<td>Provides detail and task-specific information about how to run the VAGen MSL Migration Assistance Tool.</td>
</tr>
<tr>
<td></td>
<td><strong>Ignore:</strong></td>
</tr>
<tr>
<td></td>
<td>• “Creating users and setting the current user”. There is no need for multiple users because ownership is ignored when migrating to EGL.</td>
</tr>
<tr>
<td></td>
<td>• “Loading a feature”. This is only required for GUIs.</td>
</tr>
<tr>
<td></td>
<td>• “Handling code page changes”. The information has been rewritten and is included in this paper in “Appendix E. Code page conversions”.</td>
</tr>
<tr>
<td></td>
<td>• Subsection “Duplicates for business logic” in section “Handling Duplicates”. This material only applies to VisualAge Generator Templates or BW*Wizard.</td>
</tr>
<tr>
<td>30 Using VAGen Import to migrate VAGen 2.x and Cross System Product code to Smalltalk</td>
<td>Provides an alternative way of migrating to VAGen using the VAGen Import command. If you only have one MSL with just a few CSP applications, you might be able to migrate using this technique.</td>
</tr>
<tr>
<td>31 Completing the ENVY setup on Smalltalk</td>
<td>Provides information about versioning applications and creating and versioning configuration maps. The VAGen to EGL Migration Tool requires versioned configuration maps.</td>
</tr>
<tr>
<td></td>
<td><strong>Important topic:</strong> “Testing a configuration map”. Be sure to do this to ensure that your configuration maps are correct before using the VAGen to EGL Migration Tool.</td>
</tr>
<tr>
<td></td>
<td><strong>Ignore:</strong></td>
</tr>
<tr>
<td></td>
<td>• “Changing the manager of a configuration map”</td>
</tr>
<tr>
<td></td>
<td>• “Assigning ownership of a VAGen part class”, including subsections “Adding group members” and “Changing the ownership of a VAGen part class”</td>
</tr>
<tr>
<td></td>
<td>• “Changing the manager of an application”</td>
</tr>
<tr>
<td>Chapter Title</td>
<td>Description and Comments</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>32 Completing your migration on Smalltalk</td>
<td>Provides information on creating control parts and importing work-in-progress. <strong>Important topic:</strong> - “Defining control information”. Only reference this section when directed to it by this paper’s recommendations in “Appendix A. Converting CSP control information”. - “Importing work-in-progress”. Provides task-specific information about the VAGen Import command. <strong>Ignore:</strong> - “Generating programs and packaging views”. Wait to generate until you have migrated the code to EGL. - “Migrating VSAM files”. EGL does not support local VSAM for debug purposes. - “Converting an RTABLE to a Linkage Table”. This only applies to VisualGen® Version 2.2.</td>
</tr>
<tr>
<td>33 Hints and tips on Smalltalk</td>
<td>Provides information about how to use the System Transcript, VisualAge Organizer, and VAGen Parts Browser windows. <strong>Ignore:</strong> “Refreshing the MSL Migration Assistance Tool”. It is very unlikely that there would be a change to the tool.</td>
</tr>
</tbody>
</table>

**Automatic conversions during migration to VisualAge Generator**

VisualAge Generator automatically makes the following changes to your CSP applications during migration:
- Members become VAGen parts.
- Process and statement group members are converted to function parts.
- References to processes and statement groups are converted to function invocation statements, in which the invoked function has no parameters.
- PERFORM statements and Unconditional Branch statements are converted to function invocation statements, in which the invoked function has no parameters.
- Subscript parentheses are changed to brackets in item names in the following places:
  - Statements in functions (processes and statement groups) and the program flow statements.
  - Host variable names in SQL statements.
  - Comparison value item in DL/I specifications.
  - EZEDLPCB used in a called parameter list.
- Calls to EZE service routines are converted to the corresponding function invocation statement. A statement to set the value of EZEREPLY is also added before the function invocation.
**Member associations**

The VAGen MSL Migration Assistance Tool uses the associates of a CSP member when determining how to organize your source code into Smalltalk applications (Java packages). Similarly, the VAGen to EGL Migration Tool uses the associates of a VAGen part when determining how to organize your source code into EGL files. Table 5 shows the associations between the CSP members and VAGen parts.

<table>
<thead>
<tr>
<th>CSP Member or VAGen Part</th>
<th>Possible Associates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Item</td>
<td>None</td>
</tr>
<tr>
<td>CSP Process or Statement Group</td>
<td>None</td>
</tr>
<tr>
<td>VAGen Function</td>
<td>Any shared data items (CSP global data items)</td>
</tr>
<tr>
<td>Record</td>
<td>Any shared data items (CSP global data items)</td>
</tr>
<tr>
<td>Table</td>
<td>Any shared data items (CSP global data items)</td>
</tr>
<tr>
<td>PSB</td>
<td></td>
</tr>
</tbody>
</table>
  - DL/I segment records specified in the PSB  
  - Any shared data items (CSP global data items) used by the DL/I segment records  
  - Any item used as a secondary index field in the PSB |
| Map                      |  
  - Any other maps in the same map group  
  - Any function (CSP statement group) used as an edit routine  
  - Any table used as an edit routine  
  - Any shared data item (CSP global data item) in the table |
| Map Group                |  
  - Any maps in the map group  
  - The associates of any maps in the map group |
| CSP Application          |  
  - Map groups  
  - Maps that are actually used in the program  
  - Records  
  - Tables  
  - PSB  
  - All functions (CSP processes and statement groups) used by the program  
  - The associates of any of the above parts  
  - Any shared data item (CSP global data item) referenced as a called parameter |
| VAGen Program            |  
  - Map groups  
  - Maps that are actually used in the program  
  - Records  
  - Tables  
  - PSB  
  - All functions (CSP processes and statement groups) used by the program  
  - The associates of any of the above parts  
  - Any shared data item (CSP global data item) referenced as a called parameter |

**Notes:**
- CSP global data items become VAGen shared data items.
- CSP processes and statement groups are merged into VAGen functions.
- CSP applications become VAGen programs.
- For CSP, the only associations that can be detected are those that exist in the current MSL concatenation sequence using the first found members.
- For VisualAge Generator, the only associations that can be detected are those that exist within the current Smalltalk image (or Java workspace).
The VAGen to EGL Migration Tool detects and provides the necessary import statements for the following additional associations:

- If a data item specifies a table or function as an edit routine, the table or function is treated as an associate of the data item.
- If a statement in a function explicitly references a table, the table is treated as an associate of the function.

The VAGen to EGL Migration Tool only includes declarations in a program for DL/I segment records that are actually used in the program. Therefore, the tool effectively removes the indirect association between the program and any DL/I segment records referenced in the PSB that are not used in the program.

**Hints and Tips for the VAGen MSL Migration Assistance Tool**

The following sections provide some additional information about using the VAGen MSL Migration Assistance Tool.

**Relating your EGL naming scheme to the sandbox**

After you have decided on a proposed EGL project and package structure and a proposed EGL naming scheme, you are ready to migrate as follows:

- The VAGen MSL Migration Assistance Tool creates Smalltalk applications that eventually become your EGL packages. If your EGL package name should be acctrec.functionalarea1.componenta, then name the Smalltalk application as AcctrecFunctionalarea1Componenta. Then in Stage 1 of the VAGen to EGL Migration Tool, select the following preferences:
  - Use package naming dot notation
  - Convert package names to lower case
  
  **Note:** If you use VAGen on Java, the VAGen MSL Migration Assistance Tool creates Java packages. Be sure to specify the Java package names exactly as you want EGL package names to appear. That avoids having to rename the packages during Stage 1 of the VAGen to EGL Migration Tool. The two preferences do not apply to Stage 1 tool on Java.

- When you create the Smalltalk configuration maps, create one Smalltalk configuration map for each EGL project. Give the Smalltalk configuration map the same name as the corresponding EGL project. Include in the configuration map all the applications that correspond to the EGL packages that you want to have in the corresponding EGL project.
  
  **Note:** If you use VAGen on Java, be sure to specify the Java project names exactly as you want the EGL project names to appear. That avoids having to rename the projects during Stage 1 of the VAGen to EGL Migration Tool.

**Potential problems with ESF**

Early releases of CSP allowed invalid data to be stored in the MSLs. If the External Source Format files contain invalid members, you must either correct the member on CSP/370AD and export the External Source Format file for the member again or correct the External Source Format file by editing it. The following are examples of some of the types of problems that you might encounter:

- A data item that has a length of 0.
• A map that contains a NUM field with a date edit, but the field is not long enough to contain a date.

Using the MSL Migration Assistance Tool log file

The VAGen MSL Migration Assistance Tool log file is named mslmig.log and is located in the same directory as your image. This log file stores the results of Write to File operations. You can use a text editor to view and print the log file.

Improving performance

You might want to do the following to improve the performance of the VAGen MSL Migration Assistance Tool:
• Dedicate a single workstation to use for migration. Attempting to use multiple workstations limits the tool’s ability to detect duplicates and common code and to determine when previously missing parts have been found. Having the tool provide this information more than compensates for any potential time savings that might be gained by using several workstations.
• Create the ENVY manager on the workstation that is used for migration so that performance is not degraded by network traffic.
• Create the pseudo-MSLs on the workstation that is used for migration so that performance is not degraded by network traffic.
Manual steps between the tools

There are a few steps that you must do by hand between running the migration tools.

Creating configuration maps

After you commit your applications to the ENVY manager, you need to create Smalltalk configuration maps to define the sets of applications that you want to load as a group into your image. A high-level Smalltalk configuration map is configuration map that is not specified as a required map in any other configuration map. Each high-level Smalltalk configuration map should be a group of applications and any required configuration maps that you want to process as a migration set in the VAGen to EGL Migration Tool. This high-level Smalltalk configuration map (migration set) should contain all the parts needed to generate a group of programs that you want to migrate as set. Refer to the VisualAge Generator Migration Guide, Chapter 31, “Completing the ENVY setup on Smalltalk” for details on how to define a configuration map.

Validating your configuration maps and programs

After you have defined your configuration maps, verify the migration set definition using these steps:
1. Unload the Smalltalk applications from your image.
2. Load one of the high-level Smalltalk configuration maps and its required maps into your image.
3. Select the programs in your image and validate them.
   a. If the programs validate without errors, then this ensures that the VAGen to EGL Migration Tool has the information it needs to do a good cross-part migration.
   b. If there are errors, resolve them. The following are some typical errors:
      • Missing parts, including information messages about using implicit items that are really due to missing data item definitions. Locate the missing parts in your CSP MSLs and ensure that the parts are migrated to VisualAge Generator.
      • Information messages due to using implicit items. Save all the messages to a file. Use the techniques described in the Implicits Tool white paper to create definitions for the implicit items.
   c. Version any applications that were changed. Update and version the configuration maps.
   d. Repeat the above steps until the programs validate without errors.
4. Repeat the above steps for each high-level Smalltalk configuration map so that your programs are valid before you run the VAGen to EGL Migration Tool.

Defining implicit data items

In CSP and VisualAge Generator, it is possible to specify that a program allows implicit data items. In this case, if an unqualified data item is used in a statement, but not defined anywhere in the program, test and generation automatically create a data item of the appropriate type based on how the data item is used. EGL does not permit implicit data items. If you have used implicit data items, the Implicits Tool white paper provides a method for creating the data item definitions. You should use the Implicits Tool before you use the Stage 1 VAGen to EGL Migration Tool. Using the tools in this order ensures that the data item definitions are available when you convert to EGL and that the necessary record definitions are placed with the corresponding program.
Migrating your source code from VAGen to EGL

Use the VAGen to EGL Migration Tool to migrate your source code from VAGen to EGL. Refer to the VisualAge Generator to EGL Migration Guide for details on how to run this tool.

The following is a list of enhancements that were made to VisualAge Generator that you can ignore in the VisualAge Generator to EGL Migration Guide:
- UNICODE data type
- Message queue (MQ) record
- User Interface (UI) record
- Web transaction program and related EZE words (EZEUIERR and EZEUILOC)
- Function parameters, local storage, and return values
- SQL access now supports cursor with hold
- Math functions
- String functions
- Object scripting and related EZE words (EZESCRPT)
- Other EZE words (EZEBYTES)
- C++ generation
- Java generation

The following enhancements to VisualAge Generator can affect your CSP applications during the migration to EGL:
- Ability to pass any data item on a CALL statement (not restricted to level 77 items). In addition, data items on a CALL statement can be qualified and subscripted. The EGL calling program might not be able to resolve a data item reference due to this change.

Use Table 6 as a guide to the Chapters in VisualAge Generator to EGL Migration Guide. Table 6 describes important topics and information, in addition to the topics listed above, that you should ignore.

Table 6. Chapters in VisualAge Generator to EGL Migration Guide

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Description and Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Migration Overview</td>
</tr>
<tr>
<td></td>
<td>Provides an overview of the migration process involved in converting from VAGen Version 4.5 to EGL.</td>
</tr>
<tr>
<td>2</td>
<td>Migration Tool Philosophy</td>
</tr>
<tr>
<td></td>
<td>Provides an overview of the VAGen to EGL Migration Tool and the techniques it uses.</td>
</tr>
<tr>
<td>3</td>
<td>Handling ambiguous situations</td>
</tr>
<tr>
<td></td>
<td>Describes situations in which the VAGen to EGL Migration Tool cannot always migrate correctly.</td>
</tr>
<tr>
<td></td>
<td>Ignore: “Shared edits and messages”, “Fill characters for shared data items”, and “Reserved words and UI record names”. These topics only apply when there are UI records.</td>
</tr>
<tr>
<td></td>
<td>Ignore: “I/O error value LOK”. This topic only applies to iSeries®.</td>
</tr>
<tr>
<td>4</td>
<td>Stage 1 – Extracting from Java</td>
</tr>
<tr>
<td></td>
<td>Ignore: Assumes you use VAGen on Smalltalk.</td>
</tr>
<tr>
<td>Chapter Title</td>
<td>Description and Comments</td>
</tr>
<tr>
<td>---------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>5 Stage 1 – Extracting from Smalltalk</td>
<td>Describes how to run Stage 1 to load the VAGen to EGL migration database from VAGen on Smalltalk.</td>
</tr>
<tr>
<td>6 Stage 2 – Conversion to EGL syntax</td>
<td>Describes how to run Stage 2 to convert to EGL source and store the EGL source in the migration database.</td>
</tr>
<tr>
<td>7 Stage 3 – Import</td>
<td>Describes how to run Stage 3 to extract the EGL source from the migration database and load it into your EGL workspace.</td>
</tr>
<tr>
<td>8 Running migration in single file mode</td>
<td>Describes how to import a VAGen Version 4.5 External Source Format file directly into EGL. Use this technique for CSP control information that you first convert to VAGen Version 4.5 as recommended in this paper’s “Appendix A. Converting CSP control information”.</td>
</tr>
<tr>
<td>9 Completing your migration</td>
<td>Describes additional steps you need to take to finish the migration and to ensure that your source code is correct.</td>
</tr>
</tbody>
</table>

**Ignore:**
- Subsection “Reviewing Java generation build descriptor options” in section “Reviewing your EGL build descriptor parts”. The Java options only apply if you are generating Java.
- “Establishing a program-specific bind control part”. The correct information for migrating CSP customers is included in this paper in “Appendix A. Converting CSP control information”, subsection “Bind commands”.
- “Reviewing your VGWebTransactions”. This information only applies to Web Transactions and UI records.
- “Converting VAGen preparation templates and procedures to EGL build scripts”. The correct information for migrating CSP customers is included in this paper in “Appendix B. Converting CSP preparation templates and procedures”.
- “Converting VAGen runtime templates”. The correct information for migrating CSP customers is included in this paper in “Appendix C. Converting CSP runtime templates”.
- “Converting the VAGen reserved words file”. The correct information for migrating CSP customers is included in this paper in “Appendix D. Converting CSP reserved words file”.
- “Generating and testing with Java generation”.
<table>
<thead>
<tr>
<th>Chapter Title</th>
<th>Description and Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 Language and runtime differences</td>
<td>Describes potential differences in debug or runtime.</td>
</tr>
<tr>
<td></td>
<td><strong>Ignore:</strong></td>
</tr>
</tbody>
</table>
|                                                 | • “Differences in generated Java”.
|                                                 | • “Differences between host and workstation environments”.
|                                                 | • “Differences between distributed CICS and native workstation environments”.
|                                                 | • “Difference between generated C++ and generated Java”.
| A Reserved words                                 | Lists the EGL, SQL, and Java reserved words.                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| B Relationship of VisualAge and EGL Language Elements | Compares the VAGen and EGL language elements and serves as a guide if you are trying to write the comparable EGL statement. To relate the CSP syntax to the VAGen syntax, see “Automatic conversions during migration to VisualAge Generator” on page 33.                                                                                                                                                                                                                           |
|                                                 | **Ignore:**                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|                                                 | • For data items and records, ignore the tables that deal solely with UI properties.
|                                                 | • In the “EZE words” section, ignore the subsections on “String EZE words”, “Math EZE words”, “User interface EZE words”, and “Object Scripting EZE words”.
|                                                 | • The following list shows additional language changes made after CSP Version 4.1:
|                                                 |   • CALL statement can pass any data item
|                                                 |   • Additional options for EZECONCT
|                                                 |   • SQL CURSOR WITH HOLD
|                                                 |   • Maximum of 18 digits for numeric data items.
|                                                 | • Use the “Control parts” and “Symbolic parameters” sections only in conjunction with this paper’s “Appendix A. Converting CSP control information”.
<p>|                                                 | • In the “Other generation information” section, ignore the subsections on “Preparation templates and procedures” and “Runtime templates”. The correct information for migrating CSP customers is included in this paper in “Appendix B. Converting CSP preparation templates and procedures” and “Appendix C. Converting CSP runtime templates” respectively. |
| C Messages from the migration tools               | Provides message text, explanation, and user response for messages issued by the VAGen to EGL Migration Tool.                                                                                                                                                                                                                                                                                                                                                             |
| D Messages in the Problems view                  | Provides deliberately invalid text strings that the VAGen to EGL Migration Tool inserts into the EGL source code when the tool is unable to correctly convert the VAGen source code. These text strings also apply to migrated CSP code.                                                                                                           |</p>
<table>
<thead>
<tr>
<th>Chapter Title</th>
<th>Description and Comments</th>
</tr>
</thead>
</table>
| E             | IWN.xxx messages in the Problems view  
- Provides message text, explanation, and user response for EGL messages that might appear in the Problems view.  
- Only messages that are likely to occur when you migrate VAGen code or which have a special explanation for migrated VAGen code are included.  
- Most of these messages also apply to migrated CSP code. |
| F             | Situations where incorrect External Source Format causes problems in creation of EGL  
- Provides specific VAGen Version 4.5 APARs that you should install before trying to migrate.  
- If you are migrating to EGL Version 7.1, this paper contains the most current information.  
- If you are migrating to a later release of EGL, check this appendix for any additional APARs that might be required. |
| G             | Migration Database  
- Describes how to create and manage the DB2 database used by the VAGen to EGL Migration Tool. |
| H             | Migration tool performance  
- Provides examples of VAGen to EGL Migration Tool performance based on numbers of parts, processor speed, and so on. |
| I             | VisualAge Generator and EGL interoperability  
- Provides information about the situations in which you must regenerate your programs.  
**Ignore:**  
- “VisualAge Generator and EGL interoperability on iSeries”.  
- “VisualAge Generator and EGL interoperability for Web Transactions”. |

**Hints and Tips for the VAGen to EGL Migration Tool**

The following sections provide some additional information about using the VAGen to EGL Migration Tool.

**Converting your VAGen naming scheme to EGL**

Stage 1 of the VAGen to EGL Migration Tool converts your Smalltalk applications to EGL packages.  
Be sure to specify the following Stage 1 preferences:  
- **Use package naming dot notation**  
- **Convert package names to lower case**

You can use the techniques described in the following white paper or documentation to tailor the Stage 1 migration tool:  
- **Project Consolidation** white paper to merge Smalltalk configuration maps into an EGL project or merge Smalltalk applications into an EGL package.  
- The **VisualAge Generator to EGL Migration Guide**, Chapter 5 “Stage 1 – Extracting from Smalltalk”, section “Customizing the Stage 1 migration tool” describes a technique to split the CommonParts.egl files into multiple files based on the part type and part name.  
  If you grouped
large numbers of data items, records, or functions into a single large Smalltalk application when you migrated from CSP to VAGen or if you use the technique in the Project Consolidation white paper to merge several Smalltalk applications into a single EGL package, then you should use the technique described in the VisualAge Generator to EGL Migration Guide to help to avoid very large files.

**Improving performance**

You might want to do the following to improve the performance of the VAGen MSL Migration Assistance Tool:

- Install EGL on the same workstation as VisualAge Generator.
- Create the ENVY manager on the workstation that is used for migration so that performance is not degraded by network traffic.
- Create the migration database on the workstation that is used for migration so that performance is not degraded by network traffic.
- Be sure to run the runstats.bat file from a DB2 command prompt window after Stage 1 and before Stage 2. This dramatically improves performance because DB2 is able to adjust for the large number of rows added to the database in Stage 1.
Finishing the migration of your code

After you have migrated your CSP source code to EGL, there are some additional changes that you must make by hand. You must also migrate your generation information.

Additional changes to make by hand

After you run the VAGen to EGL Migration Tool, you must make the following changes by hand:

- If there are errors in the log file from the VAGen to EGL Migration Tool or in the Problems view, you must correct the errors. Refer to the VisualAge Generator to EGL Migration Guide, Chapter 9 “Completing your migration”, section “Reviewing your EGL source code” for details of the changes that might be required.

- If your runtime environment is IMS/VS, ensure that the segmented=yes property is specified for your CSP main transaction (EGL main text) programs that use the converse statement and segmented=no is specified for main transaction programs that only use XFER with a map and First Map to display maps (EGL show statement and inputForm properties). In releases of CSP prior to Version 4.1, the execution mode (EGL segmented property) is specified at generation time. In CSP Version 4.1, the execution mode is specified on the application member at definition time. However, if you migrated from an earlier version of CSP to Version 4.1 and never generated the application using CSP Version 4.1, the CSP execution mode might have been set incorrectly, resulting in the wrong value for the segmented property in EGL.

  Note:
  The VAGen to EGL Migration Tool only includes the segmented property for EGL main text programs. Therefore, you can use a File Search for this property to find all programs that are potentially affected by this problem.

- For each EGL DataTable, ensure that the EGL shared and resident properties are set correctly based on the CSP options you specified when you generated the table. In CSP, shared and resident are specified at generation time. In EGL, the shared and resident properties are specified at definition time in the DataTable part.

  - If you saved CSP generation options, the shared and resident properties are stored in the CSP table member. The shared property appears in the External Source Format for the table on the :genopts tag in the TYPEUSE attribute. The values SHARED and SINGLE convert to shared=yes and shared=no respectively. The resident property appears in the External Source Format for the table on the :genopts tag in the RESIDENT attribute. The values Y and N convert to resident=yes and resident=no respectively. The migration tools make these conversions for you. However, you should review the values for the shared and resident properties very carefully because the information stored in the CSP table member reflects the last time you specified to save the information – which is not necessarily the information you used the last time you generated the table.

  - If you did not save CSP generation options, the VAGen MSL Migration Assistance Tool defaults to TYPEUSE=SHARED and RESIDENT=N. The VAGen to EGL Migration Tool converts this to shared=yes and resident=no. You should carefully review the values for the shared and resident properties for each of the EGL DataTables.

  Note:
  To find all the DataTables, use the EGL Search with * as the Search string, DataTable as the Search For part type, and Part declarations as the Limit To value.

- For each EGL program, ensure that the EGL deleteAfterUse property for the use declaration for each DataTable used by the program is set correctly based on the CSP option you specified when
you generated the application. In CSP, the Keep After Use property is specified for each table used by an application when you generate the application. In EGL, the `deleteAfterUse` property is specified at definition time in the program part on the `use` declaration statement for each DataTable.

- If you saved CSP generation options, the Keep After Use property is stored in the CSP application member on the `gentable` tag in the KEEP attribute. The values Y and N convert to `deleteAfterUse=no` and `deleteAfterUse=yes` respectively. Note that the meaning of the `deleteAfterUse` property is the opposite of Keep After Use, so the value must be reversed. The migration tools make this conversion for you. However, you should review each use declaration in each program very carefully because the information stored in the CSP application member reflects the last time you specified to save the information – which is not necessarily the information you used the last time you generated the application.
- If you did not save CSP generation options, the VAGen MSL Migration Assistance Tool defaults to KEEP=Y. The VAGen to EGL Migration Tool converts this to `deleteAfterUse=no` and then omits the `deleteAfterUse` property from the `use` declaration for the table because it is the default value. You should carefully review the `use` declaration statements for each table used in each program.

**Note:**
The VAGen to EGL Migration Tool only includes the `deleteAfterUse` property if the value is `deleteAfterUse=yes`. Therefore, you cannot search on `deleteAfterUse` to find the use declarations for all the DataTables. To find all the programs, use EGL Search and specify asterisk (*) as the Search string. Program as the Search For part type, and Part declarations as the Limit To value. Then review the `use` declarations for all the DataTables in each program.

### Migrating your generation information

In addition to your CSP source code that is stored in MSLs, you have other information that is used during generation that is stored in various places in CSP. You must migrate this information in the following ways:

- Information you specified through the online generation user interface or with the following commands for batch generation:
  - ASSOCIATE command. This information is now in the EGL resource associations part. See “Resource associations” on page 76 for details.
  - SETGEN command which includes the following information:
    - The information about which map groups and tables to generate is now in the `genFormGroup`, `genHelpFormGroup`, and `genDataTables` build descriptor options in the EGL build descriptor parts. See “CSP generation options” on page 61 for details.
    - For tables, the information about SHARED and RESIDENT is now in the `shared` and `resident` properties in the EGL DataTable part. The information about KEEP AFTER USE is now in the `deleteAfterUse` property in the program’s `use` declaration for the DataTable. The migration tools might not convert these properties correctly. See “Additional changes to make by hand” on page 43 for details about setting these properties for the EGL DataTable and program parts.
  - GENERATE command. The generation options are now in the EGL build descriptor part. See “CSP generation options” on page 61 for details.
- Information about other generation options that you specified through the online generation user interface or as a member of a PDS for batch generation. This information is now in the EGL build descriptor part. See “CSP generation options” on page 61 for details.
Information saved in the application MSL member the last time the application was generated and the SAVE(YES) generation option was specified. This information is also included in the External Source Format files in the following tags:

- The :targsys tag. This information is now in the system build descriptor option of the EGL build descriptor part. See “CSP generation options” on page 61 for details.
- The :genopts tag for a program. This information is now in the EGL build descriptor part. See “CSP generation options” on page 61 for details.
- The :genopts tag for a table. The information about SHARED and RESIDENT is now in the shared and resident properties in the EGL DataTable part. The migration tools might not convert the shared and resident properties correctly. See “Additional changes to make by hand” on page 43 for details about setting these properties for the EGL DataTable parts.
- The :gentable tag for a program. The information about KEEP AFTER USE is now in the deleteAfterUse property in the program’s use declaration for the table. The migration tools might not convert the deleteAfterUse property correctly. See “Additional changes to make by hand” on page 43 for details about setting this property for the EGL program parts.
- The :genfile tag for a program. This information is now in the EGL resource associations part. See “Resource associations” on page 76 for details.

Other information used at generation or preparation time:

- Linkage table. This information is now in the EGL linkage options part. See “Linkage table” on page 70 for details.
- Bind commands. This information is now in the EGL bind control part. See “Bind commands” on page 78 for details.
- Link edit commands. This information is now in the EGL link edit part. See “Link edit commands” on page 80 for details.
- Preparation JCL templates. EGL uses a build server and build scripts. See “Appendix B. Converting CSP preparation templates and procedures” on page 84 for details.
- Runtime JCL templates. EGL also provides templates for runtime JCL. See “Appendix C. Converting CSP runtime templates” on page 87 for details on how to convert your runtime JCL templates.
- Reserved words. EGL supports additions to its reserved words file. See “Appendix D. Converting CSP reserved words file” on page 90 for details on how to convert your CSP reserved word file, if any.

Storing control information

In CSP, control information (generation options, linkage table, resource associations, bind commands, and link edit commands) is stored in various places outside the MSL. Some information is optionally stored in the application members.

In EGL, the information is stored in EGL parts in an EGL build file, with the file extension .eglbld.

You need to determine how to organize the EGL control information. For example, you probably need to use different build descriptor options for debug, development, test, and production so that you can specify different load libraries for the outputs of generation and the outputs of the build server. Similarly, you might need to point to different DB2 systems or specify different linkage options or resource associations depending on your level of testing.

The best way to organize your control parts is to collect all the control information for a subsystem in a single project, single package, and single build parts file. This avoids the need to specify EGL
Build Path information and the need for `import` statements in the `.eglbld` files, thus making it easier to maintain the control information. You might name the project `sssZZZControlInformation`, where `sss` is the same prefix you use for other projects in the subsystem and `ZZZ` is a constant so that the control information project is always near the bottom of the scrolling lists.

**Note:**

The example below only considers build descriptor, linkage option, and resource associations parts. You also need to establish naming conventions for bind control and link edit parts, but there are specific rules that govern these part names. You can include the bind control and link edit parts in the same build parts file with other options. However, if you only generate a program for one runtime environment, it might be better to place the corresponding bind control and link edit parts in the same package with the program. See “Bind commands” on page 78 and “Link edit commands” on page 80 for details on required naming conventions for these part types in EGL.

You might use the following naming conventions for your EGL build parts:

- `sssCommonBuildOpts`
- `sssVvvvvvCommonBuildOpts`
- `sssVvvvvvXxxxBuildOpts`
- `sssVvvvvvXxxxLinkageOpts`
- `sssXxxxResourceAssociations`
- `sssDebugBuildOpts`
- `sssDebugLinkage`
- `sssDebugResourceAssociations`

**Where:**

- `sss` Is the subsystem ID. This should be the same prefix that you used for the other projects in the subsystem. If you want the control projects to always appear at the bottom of the list of projects for the subsystem, consider using `sssZZZ` as the prefix.
- `Vvvvvv` Indicates the runtime environment. For example, ZOSBATCH, ZOSCICS, IMSVS, and IMSBMP.
- `Xxxx` Indicates the level of testing. For example, Development, Test, Production, and Emergency changes.

The `sssCommonBuildOpts` part contains build descriptor options that are common across all levels of testing and across all runtime environments. This might include options such as `sysCodes`, `targetNLS`, and `math`. Build descriptor options that have the same values for the runtime environment across all levels of testing go into `sssVvvvvvCommonBuildOpts`. Options that have different values for the runtime environment based on the level of test go into `sssVvvvvvXxxxBuildOpts`.

The build descriptor parts that vary for `Vvvvvv` specify options that vary based on the runtime environment or which are specific to that environment. For example, you might need to specify different values for the `linkage` build descriptor option for each runtime environment. Some options, such as `endCommarea`, `twaOffset`, and `workDBType`, only apply to CICS and are only included in
the `szsZOSCICSXxxxBuildOpts` set of build descriptor parts. Similarly, some options, such as `genRunFile`, only apply to IMS BMP or z/OS Batch and are only included in the relevant set of build descriptor parts.

The build descriptor parts that vary for both `Vvvvv` and `Xxxx`, specify different values for certain options, such as the `projectID`, that need to vary with both the runtime environment and the level of testing for which you are generating. For example, the `projectID` build descriptor option enables you to vary the high-level qualifier of the runtime load library. If you use the default data set naming conventions, the runtime environment is automatically included as the second qualifier. However, if you use different data set naming conventions, you might need to vary the `projectID` for each runtime environment and each level of testing. For example, you might need `projectID="MYIMSVS.TEST"` for generating to your IMS/VS test environment and `projectID="CICS.PROD"` for generating to your CICS production environment.

The resource associations part is unique in that it is organized based on the `fileName` property specified in the EGL serial, indexed, and relative record part definitions. For each `fileName`, you can specify the resource association for each runtime environment. Therefore, you might be able to use a single resource associations part for all the runtime environments in your subsystem. See “Resource associations” on page 76 for information about when you must use multiple resource associations parts. In addition, the resource associations part uses `fileName="printer"` to specify the association for any print output. Similar to serial, indexed, and relative files, you can specify the resource association for the printer for each runtime environment in the same resource associations part.

In CSP you can chain generation options members together using the `OPTIONS(nextOptionsMember)` option to point to the next option member in the chain. In EGL, the equivalent build descriptor option is `nextBuildDescriptor`. Table 7 shows an example of chaining the build descriptor parts for the test level of a z/OS CICS system.

### Table 7. Chaining build descriptor parts

<table>
<thead>
<tr>
<th><code>szsZOSCICS_TEST</code></th>
<th><code>szsZOSCICS_COMMON</code></th>
<th><code>szsCOMMON</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>system=&quot;ZOSCICS&quot;</code></td>
<td><code>endCommarea=&quot;NO&quot;</code></td>
<td><code>sysCodes=&quot;YES&quot;</code></td>
</tr>
<tr>
<td><code>projectID=&quot;TEST.ZOSCICS.HLQ&quot;</code></td>
<td><code>twaOffset=&quot;100&quot;</code></td>
<td><code>targetNLS=&quot;ENU&quot;</code></td>
</tr>
<tr>
<td><code>linkage=&quot;szsZOSCICSTestLinkageOpts&quot;</code></td>
<td><code>workDBType=&quot;AUXMAIN&quot;</code></td>
<td><code>math=&quot;COBOL&quot;</code></td>
</tr>
<tr>
<td><code>resourceAssociations=&quot;sztTestResourceAssociations&quot;</code></td>
<td><code>nextBuildDescriptor=&quot;szsCOMMON&quot;</code></td>
<td><code>nextBuildDescriptor=&quot;szsCOMMON&quot;</code></td>
</tr>
</tbody>
</table>

If you permit developers to generate such that the outputs of generation and build server processing go the developer’s private load libraries, you can extend the technique described above. In this case, you can create one project for each developer that contains that developer’s private build descriptor options. You might call this project `ZZZDdddDControlProject`, where `Dddd` is the developer name or initials. Having a separate project for each developer makes it easier for the developer to control any special options needed for testing. The developer might define a build descriptor part named `DdddVvvvvXxxxOpts`, where `Vvvvv` varies based on the runtime environment and `Xxxx` varies based on the level of testing. Typically, the developer only needs to specify a few options such as the following options:
- **projectID** to specify the high-level qualifier for the developer’s private load library.
- **destUserID** and **destPassword** to specify the developer’s userID and password for the host where the build server runs.
- **sqlID** and **sqlPassword** to specify the developer’s SQL userID and password if you include the validateSQLStatements=”YES” build descriptor option.

The developer’s build descriptor part then chains to the appropriate runtime and test build descriptor part for the subsystem (for example, **Ddddd ZOSCICS_TEST_Opts** can chain to the **sssZOSCICS_TEST** part shown in Table 7).

You can set default build descriptors for your entire workspace. To set the workspace default build descriptor preferences, from the Preferences window, expand **EGL -> Default Build Descriptor** and then set the following preferences:
- **Debug build descriptor** preference to **sssDebugBuildOpts**
- **Target system build descriptor** to either **sssVvvvvvXxxxBuildOpts** or **DdddVvvvvvXxxxOpts**, where **Vvvvvv** varies based on the runtime environment and **Xxxx** varies based on the level of test for which you typically generate.
Setting up your EGL development environment

Setting up your development environment requires more than just converting the CSP source code to EGL. You must be able to make changes to the code, and then debug and generate. To have a working development environment, you need to consider the following factors:

- During cross-part migration, some information has moved from one part in CSP to a different part in EGL. You need to be aware of these differences when you make changes to your code.
- Depending on how long it takes you to migrate, you might need to make provisions for dual maintenance of the common parts on CSP and EGL.
- Depending on the logic of your programs, you might need to make use of some special techniques for debugging your programs in EGL.
- Depending on the logic of your programs, you might need to make special provisions for connectivity to the host.
- When you debug your programs, you need to be aware of differences between the host and workstation environments.
- You might want to use batch commands.

The above topics are discussed in the sections that follow.

Information that moves during cross-part migration

For most parts and statements, the EGL source code has a direct relationship to the original CSP source code. However, during cross-part migration, the VAGen to EGL Migration Tool moves some information. The following list provides some examples:

- In CSP, there is a REDEFINED record type. For example, if RecordA is a working storage record and RecordB is a REDEFINED record that specifies that RecordA is the record that RecordB redefines, then RecordA and RecordB occupy the same physical storage, but provide different layouts of the fields. In EGL, both RecordA and RecordB become basicRecords. When you define a program and include a record declaration for RecordB, you specify the redefines property to indicate that RecordB occupies the same physical storage as RecordA. The VAGen to EGL Migration Tool automatically converts RecordB to a basicRecord and adds the redefines property to the programs that use RecordA and RecordB.
- In CSP, you can include level 77 items in a program’s primary working storage record. EGL does not support level 77 items. The VAGen to EGL Migration Tool splits a working storage record into two basicRecords as follows:
  - An EGL basicRecord that includes all the items except the level 77 items. This record has the same name as the original working storage record.
  - A second basicRecord that contains all the level 77 items from the original record. This record has the same name as the original working storage record, with an appended suffix.
- In CSP, you only specify the name of an SQL table in the SQL record definition. In an I/O process, you specify the SQL record name. Test and generation automatically retrieve the SQL table information for the process from the record definition. This occurs even if you modify the SQL statement. In EGL, debug and generation only retrieve the SQL table information for default I/O statements. If you modify the SQL statement, you must specify the SQL table name in the I/O statement. The VAGen to EGL Migration Tool includes the SQL table information in any modified SQL I/O statements. However, if you later change the SQL table name, you must be sure to change any functions that contain modified SQL I/O statements.
In addition to the changes made by the VAGen to EGL Migration Tool, the VAGen MSL Migration Assistance Tool moves the following information from generation options to parts:

- In CSP, when you generate a table, you specify the SHARED and RESIDENT properties. In EGL, `shared` and `resident` are properties that you specify when you define a DataTable part.
- In CSP, when you generate a program, you specify the KEEP AFTER USE property for each table that the program uses. In EGL, the `deleteAfterUse` property (the opposite of KEEP AFTER USE) is specified on the `use` declaration for the DataTable when you define the program.

Considerations for dual maintenance of source code

External Source Format provides a human-readable form for both CSP and VAGen source code. You can migrate a CSP External Source Format file forward to VisualAge Generator and then migrate a VisualAge Generator Version 4.5 External Source Format file forward to EGL. However, you cannot migrate a VAGen External Source Format file back to CSP.

EGL is written in a human-readable form. Therefore, there is no need for External Source Format. There is no way to migrate EGL source code back to either VisualAge Generator or CSP.

If you are not migrating all of your CSP source code at the same time and have parts that are needed by both the CSP code and the EGL code, you need to make changes to the common parts in one of the following ways:

- Make the change in CSP, then export an External Source Format file for the part, migrate it to VisualAge Generator and from there to EGL.
- Make the change in both the CSP and EGL versions of the part.

Recommendation: Before you start migration, perform the following tasks:

- Generate and test all the CSP applications that are work-in-progress and move them into production.
- Freeze your CSP development and maintenance.

This technique enables you to migrate just your production level CSP code and to avoid dual maintenance of the source. If you must do dual maintenance, try to limit the changes to a small number of parts and then make the changes in both the CSP and EGL versions of the part. Making the changes both places can be easier because you do not have to determine all the related parts that are required for cross-part migration.

Special techniques for debugging in EGL

EGL provides two special techniques to help you debug z/OS host programs on the workstation. These techniques are applicable in a variety of situations. The techniques are as follows:

- EGL enables you to specify that you want to debug in EBCDIC mode. This technique avoids unnecessary (and sometimes unwanted) data conversions between ASCII code pages on the workstation and EBCDIC code pages on the z/OS host.
- EGL provides the `sysVar.systemType` variable to enable you to determine if the program is running in debug mode. You might be able to use this variable to adjust the program logic to deal with host connectivity and workstation environmental differences.
These techniques are described in more detail in the sections that follow.

**Debugging in EBCDIC mode**

EBCDIC is the encoding mechanism used for storing data on the z/OS host and controls the collating sequence. ASCII is the encoding mechanism used for storing data on the workstation and controls the collating sequence for the EGL developer, debugger, and Java-generated programs. UNICODE is the encoding mechanism for displaying or entering data or literals on the workstation and for interacting with SQL. The difference in the EBCDIC and ASCII collating sequences can affect such things as:

- Range match tables.
- Specific values coded for the high value or low value of a key.
- Comparison of key values (for example, when doing a 2-file match based on a key field).
- Automatic conversion of data when it is moved between the workstation and the z/OS host for VSAM I/O or a call statement.

The following is an example of the problems that can occur due to automatic conversion between ASCII and EBCDIC. Consider the situation in which RECORDA is defined with the following formats:

```plaintext
Record RecordA type indexedRecord
  03 MYKEY char(9);
  03 MYRECORDTYPE char(1); // values are A, B, and C
  03 MYDATA char(90);
end

Record RecordB type basicRecord
  03 MYKEY char(9);
  03 MYRECORDTYPE char(1);
  03 MYDATA char(90);
  04 CHARFIELD_B1 char(50);
  04 BINFIELD_B bin(9,2);
  04 DECIMAL_B decimal(11,2);
  04 CHARFIELD_B2 char(30);
end

Record RecordC type basicRecord
  03 MYKEY char(9);
  03 MYRECORDTYPE char(1);
  03 MYDATA char(90);
  04 CHARFIELD_C1 char(40);
  04 BINFIELD_BC bin(9,2);
  04 CHARFIELD_C2 char(10);
  04 DECIMAL_BC decimal(11,2);
  04 CHARFIELD_C3 char(30);
end
```

The program provides the following record declarations:

```plaintext
RECORDA RECORDA;
RECORDB RECORDB {redefines="RECORDA"};
RECORDC RECORDC {redefines="RECORDA"};
```

Given this set of record declarations, all three records occupy the same physical storage, with different definitions depending on the value of MYRECORDTYPE. Your program logic checks the value of MYRECORDTYPE and then performs whatever processing is required using the record definition that corresponds to the value of MYRECORDTYPE.
Any I/O for the record must always be done using RECORDA because it is defined as an indexed record. In this case, for remote VSAM I/O, all the data is converted automatically from ASCII to EBCDIC when the data is sent to the z/OS host and then converted back from EBCDIC to ASCII when the data is returned to the workstation. This means that if the data in the physical record is actually in the format of RECORDB or RECORDC, the binary and decimal fields are converted incorrectly. Similar problems occur if you move MYDATA to or from any parent field that has a substructure that contains non-character data.

In addition, if you use RECORDA in any call statements, the data in the record is automatically converted between ASCII and EBCDIC before and after the call statement. If the data in the physical record is actually in the format of RECORDB or RECORDC, the binary and decimal fields are converted incorrectly.

To help with these situations, EGL provides a preference that enables you to specify that you want to debug using the EBCDIC collating sequence. If you select this preference and debug your program, all your data is stored in EBCDIC mode and only converted to UNICODE when it is displayed on the workstation (for example, when a textForm is conversed or when data is displayed in the Variables view of the Debug perspective). Using EBCDIC mode for debugging avoids the collating sequence issues listed above. In addition, when you debug in EBCDIC mode, conversion does not occur for remote VSAM I/O or for a call statement, thus avoiding the problems with record redefinitions. To set the preference, follow these steps:
1. From the workbench window, click Window -> Preferences.
2. Expand EGL and select Debug.
3. Set the Character Encoding preference to the EBCDIC code page for your host system. Refer to the EGL online helps for details of how to set and use this preference.

When you use the Character Encoding preference to specify the code page information for your z/OS host system, you should not specify a conversion table for the callLink entry in the debug linkage options part.

**Testing the value of sysVar.systemType**

In CSP, you can test EZESYS to determine the runtime environment. The EGL equivalent is sysVar.systemType. EGL also permits you to test whether you are running in the debugger. For example, if your runtime environment is z/OS CICS or IMS/VS and you want to transfer to another transaction, you must specify the transaction code. However, in the debug environment, you must specify a program name. To help with this situation, you can change your program logic in the following way to determine whether you are running in the debugger:

```
if (sysVar.systemType is debug)
    sysVar.transferName = ”PROGNAME”;
else
    sysVar.transferName = ”TRXNAME”   // original statement
end
transfer to transaction sysVar.transferName passing recordName;
```

PROGNAME is the name of the EGL program that is invoked when you specify TRXNAME as the transaction name in the z/OS CICS or IMS/VS runtime environment. You can check the value of sysVar.systemType whenever there is a difference between what you want to do when debugging the program and what you want to do at runtime.
Before you debug your program, you must set a preference to indicate that the `sysVar.systemType` is debug. To set this preference, follow these steps:

1. From the workbench window, click **Window -> Preferences**.
2. Expand **EGL** and select **Debug**.
3. Select the **Set systemType to DEBUG** preference.

   **Note:**
   
   If you do not select the **Set systemType to DEBUG** preference, the debugger uses the value of the `system` build descriptor option in your debug build descriptor part. Using a value such as `system=“ZOSCICS”` in the debug build descriptor part enables you to test the logic of an **if** or **while** statement, but does not enable you to run a statement that is only valid in the ZOSCICS runtime environment. For example, if `system=“ZOSCICS”` in your debug build descriptor part, the **if** statement in the following program logic tests true, but the `sysLib.purge` statement is ignored because it is not valid in the debug environment:
   
   ```plaintext
   if (sysVar.systemType is zoscics)
       sysLib.purge(“MYQUEUE”);
   end
   ```

**Considerations for the EGL debugger and z/OS host connectivity**

There are some special considerations when you debug an EGL program on the workstation. These topics are discussed in the following sections:

- Calling programs on the z/OS host
- Accessing SQL databases
- Accessing DL/I databases
- Accessing VSAM files
- Accessing CICS-specific resources

**Calling programs on the z/OS host**

If your CSP applications call non-CSP programs, you need to plan for testing this scenario with the EGL debugger. Depending on your CSP environment and what the non-EGL program does, you might use one of the following techniques:

- Use a stub EGL program to simulate what the non-EGL program does so that all testing can occur on the workstation.
- Create a workstation version of the non-EGL program. The workstation version can be a C, C++, or Java program.
- Use a remote call to the z/OS host version of the non-EGL program. This technique is only possible for non-EGL programs that run in the CICS or IMS/VS environment. In addition, for debugging with remote calls, there is a restriction that the total length of all records passed on the remote call:
  - For CICS the restriction is 32567 characters. If you use parmForm=COMMPTR (the default linkage for CICS), there might be a problem with the total number of characters. If you use parmForm=COMMDATA, then the total length is highly likely (but not guaranteed) to be within the limit.
  - There is a similar restriction on the length of data passed on a remote call to IMS/VS.
  - Refer to the **EGL Programmer’s Guide** or online helps for details of how to define a remote call for debug purposes.
Use the technique described in “Testing the value of sysVar.systemType”.

If you use a remote call to the z/OS host, follow these steps:
1. From the workbench window, click Window -> Preferences.
2. Expand EGL and select Debug.
3. Set the Remote User and Remote Password preferences. These preferences are used for security access to the z/OS host.

Depending on the data that you pass as parameters to the z/OS program, you might want to consider debugging in EBCDIC mode as described in “Debugging in EBCDIC mode”. Consider the following example: a program defines a record that has a large character field whose format varies depending on the value of another field in the record. The program always uses the record containing the large character field for the call statement, but uses different redefinitions for the record when doing most of the program logic. In this example, debugging in EBCDIC mode avoids converting the large character field between ASCII and EBCDIC and thus avoids corruption of substructured fields that are really binary or other types of numeric data in the record redefinitions.

Accessing SQL databases

If your CSP applications use SQL, you can debug the programs using SQL tables on the workstation or on your z/OS host system. Based on your decision, you need to install the appropriate DB2 product. If you plan to use SQL tables on z/OS, you need to establish connectivity to the host. Refer to the DB2 documentation and the EGL online helps for details of how to establish the connectivity.

If you plan to use SQL tables on the workstation, you might want to set the collating sequence to match the host collating sequence so that data appears in the same sequence as at runtime.

Regardless of where your SQL database is located, you must include the following build descriptor options in your debug build descriptor part: sqlDB, sqlJDBCDriverClass, and sqlValidationConnectionURL. Refer to the EGL online helps for details on how to specify the values.

You can optionally include the following SQL-related build descriptor options: sqlID and sqlPassword. If you do not include your SQL user ID and password in the debug build descriptor part, follow these steps:
1. From the workbench window, click Window -> Preferences.
2. Expand EGL and select Debug.
3. Select the Prompt for SQL user ID and password when needed preference. This preference causes a prompt window to appear the first time an SQL access is attempted in the debugger.

You must also consider any character fields that are substructured by other types of data. Consider the following example: an SQL record has a large character field whose format varies depending on the value of another field in the record. A program uses the record containing the large character field for the SQL I/O statement, but uses different redefinitions for the field when doing most of the program logic. EGL converts the data to UNICODE before the SQL call and DB2 converts the UNICODE data to EBCDIC before accessing the host database. If the field is defined as character data, but actually contains binary or decimal data, there is not necessarily a corresponding code point for each byte during the conversions. To deal with this problem, follow these steps:
1. Change the SQL table definition of the column to specify “FOR BIT DATA”. This instructs DB2 to not convert the character field from UNICODE to EBCDIC.
2. Change the EGL SQL record definition and include the asBytes=yes property for the character field. This instructs EGL to not convert the character field to UNICODE before the SQL call.

3. Debug in EBCDIC mode by setting the EGL Character Encoding preference as described in “Debugging in EBCDIC mode”. This causes the data to be stored in EBCDIC mode so that no conversion is necessary.

If you use a character field that is substructured by other types of data and use a local SQL database, you can use the same technique, except set the EGL Character Encoding preference to the default value so that the data stays in ASCII.

**Accessing DL/I databases**

If you CSP applications use DL/I, refer to the VisualAge Generator to EGL Migration Guide, Chapter 10 “Language and runtime differences”, section “Differences in DL/I support” for assistance in modifying your IMS or DL/I PSBs for use with the EGL debugger. In addition, refer to the EGL Programmer’s Guide for information about how to define the information required by the debugger to process DL/I I/O statements.

**Accessing VSAM files**

If your CSP applications implement serial, indexed, or relative files as VSAM files, you need to plan for testing this scenario with the EGL debugger. For a serial file, you can use a non-VSAM sequential file on the workstation or a remote VSAM file on z/OS. For indexed and relative files, you must use a remote VSAM file on z/OS.

Remote access to VSAM files requires that you install Distributed FileManager (DFM) on the workstation. For details, locate the following directory: `InstallationDirectory\bin`

Unzip the file named VSAMWIN.zip into a new directory and then follow the directions in the INSTALL.README file.

You must also install and configure Distributed FileManager on z/OS. Refer to the manual z/OS Distributed FileManager Guide and Reference (SC26-7395) for details. The white paper Accessing VSAM files on OS/390 from VisualAge Generator is also useful. It is available in the “Additional links” section at the following Web site: [http://www-306.ibm.com/software/awdtools/visgen/library/v45docs.html](http://www-306.ibm.com/software/awdtools/visgen/library/v45docs.html)

Even though the product names mentioned in the white paper are different, you can use the white paper for guidance in setting up remote VSAM access with the following changes:
- Sections I through IV. The general process for setting up remote VSAM files is the same as described in the white paper, with the following changes:
  - In section III, the manual DFM/MVS Guide and Reference (SC26-4915) is replaced by the manual z/OS Distributed FileManager Guide and Reference (SC26-7395) mentioned above.
  - In section IV, subsection “Configure DFM”, there are the following changes:
    - You do not need to install or download or install IBM VisualAge for COBOL. Instead of vsamnt.zip, use the VSAMWIN.zip file shipped with EGL.
• Set the environment variables as specified in the INSTALL.README from the VSAMWIN.zip file shipped with EGL.
• Run the DFMCFG.EXE command as described in the INSTALL.README file after setting the environment variables.
• In section IV, subsection “Stopping DFM”, the referenced manual Distributed FileManager User’s Guide (SC26-7134) is included in the VSAMWIN.zip file as DFM.pdf.
• Section V. The main difference is that in EGL you specify a remote VSAM file in the following way:
  • Create a debug resource associations part to specify the file name information (rather than specifying the information in the resource association file as in VisualAge Generator). For details about how to specify the remote VSAM file name in the debug resource associations part, refer to the topic “VSAM support” in the EGL online helps.
  • Create a debug build descriptor part and use the resourceAssociations build descriptor option to point to the debug resource associations part (rather than specifying a resource association file in a preference as in VisualAge Generator).
  • Set your EGL debug build descriptor preference to point to your debug build descriptor part.
• Sections VI through IX. Ignore these sections. They do not apply to EGL.
• Section X.
  • Ignore the information on setting trace options. It does not apply to EGL.
  • The same I/O return codes are used in EGL. For a complete list of the VSAM reply messages refer to the manual SMARTdata Utilities VSAM Application Programming Interface Reference (SC26-7133), Chapter 6, “VSAM API Reply Messages”, subsection “Reply Messages”. This manual is included in the VSAMWIN.zip file as VSAM.pdf.
• Section XI. The sample VTAM definition applies to EGL.

Depending on the layout of your VSAM records, you might want to consider debugging in EBCDIC mode as described in “Debugging in EBCDIC mode”. Consider the following example: a serial, indexed, or relative record has a large character field whose format varies depending on the value of another field in the record. The program always uses the record containing the large character field for the I/O statement, but uses different redefinitions for the record when doing most of the program logic. In this case, debugging in EBCDIC mode avoids converting the large character field between ASCII and EBCDIC when the I/O occurs and thus avoids corruption of substructured fields that are really binary or other types of numeric data in the record redefinitions.

**Accessing CICS-specific resources**

If you use CSP/370AD in the CICS environment, you have access to CICS-specific file types, such as transient data queues, temporary storage queues, and spool files from the CSP Interactive Test Facility. These file types are not available in the EGL debugger. When testing with the EGL debugger, you can use the following techniques:
• For a serial file associated with a transient data queue, temporary storage queue or spool file, you can use either a local sequential file or a remote VSAM file.
• For a relative file associated with a temporary storage queue, you can use a remote VSAM file. However, some system functions such as sysLib.purge (EZEPURGE in CSP) are not available in the debugger.
• For a printer file associated with a transient data queue or spool file, you can use a local file.
Recoverable files are not available in EGL. If you use recoverable files in the CICS environment, be sure to make a back-up copy of any serial, indexed, relative, or print files that you might need to restore before starting your debugger session.

If you use CSP/370AD in the CICS environment, you have access to CICS-specific facilities. These facilities are not available in the EGL debugger. When testing with the EGL debugger, the following statements are ignored:
- AUDIT service routine (EGL sysLib.audit), which writes entries to the CICS journal.
- CREATX service routine (EGL vgLib.startTransaction), which invokes an asynchronous CICS transaction.
- EZEPURGE special function word (EGL sysLib.purge), which purges as CICS temporary storage queue.

In addition, the XFER statement (EGL transfer to transaction statement), which specifies a transaction name in CICS or IMS, requires a program name when you debug the program.

You can use the technique described in “Testing the value of sysVar.systemType” to vary the processing that occurs when you debug your program to account for these differences between CICS and the debug environment.

Considerations for the EGL debugger differences from the z/OS host

You need to be aware of differences between debugging with EGL on the workstation and running a generated program on the z/OS host. These topics are discussed in the following sections:
- Program logic that varies based on the user ID or terminal ID
- Arithmetic calculations
- Keyboard mapping
- Double-byte languages

Program logic that varies based on the user ID or terminal ID

Your program logic might use the user ID or terminal ID to control the processing. For example, you might display employee salary information to a manager but not to a non-manager employee. If you need to use a particular user ID or terminal ID when you debug, follow these steps:
1. From the workbench window, click Window -> Preferences.
2. Expand EGL and select Debug.
3. Set the terminalID and userID preferences. These preferences set the initial value of the sysVar.terminalID and sysVar.userID respectively. If you need to use different values later in your debug session, you can change the values using the Variables view of the Debug perspective.
**Arithmetic calculations**

The EGL debug facility emulates the math="COBOL" method of storing intermediate results for arithmetic calculations. EGL debug does not provide emulation of the math="CSPAE" method of storing intermediate results. Therefore, your results might vary between the debug facility and your generated COBOL.

**Keyboard mapping**

For debug, some keys are not generally available on a workstation keyboard. Table 8 shows the key mappings used in EGL when you converse a textForm.

<table>
<thead>
<tr>
<th>3270 Keys</th>
<th>EGL mappings on Windows Operating Systems</th>
<th>EGL mappings on Linux and AIX</th>
</tr>
</thead>
<tbody>
<tr>
<td>PF1 - PF12</td>
<td>F1 - F12</td>
<td>F1 - F12</td>
</tr>
<tr>
<td>PF13 - PF24</td>
<td>Shift+F1 - Shift+F12</td>
<td>press Ctrl+S and then press F1 - F12</td>
</tr>
<tr>
<td>PA1 - PA3</td>
<td>Ctrl+F1 - Ctrl+F3</td>
<td>press Ctrl+A and then press F1 - F3</td>
</tr>
</tbody>
</table>

**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.

**Double-byte languages**

For debug with double-byte languages, it is not necessary to type an SI or SO to indicate a switch between SBCS and DBCS data when you enter data into a textForm field. However, if your data is being sent to the z/OS host during debug or you use the z/OS host for your runtime environment, you should ensure that the EGL needsSOSI validation property is specified for all MBCHAR fields on your textForms. This property ensures that the data you enter at the workstation is short enough to fit into the fields when the SO and SI characters are included at the z/OS host.

**Batch Command interface**

In CSP, there is a command interface to enable you to run WHERE USED, LIST, LISTA (List Associates), and other commands against your MSLs. EGL does not provide comparable batch commands. All processing, with the exception of a batch generation capability, is done online in your workspace. Refer to the EGL online helps for information about batch generation.
Migrating your runtime environment

You can use the EGL developer server product to replace CSP/370RS Version 2.1 as a runtime environment. You can modify any runtime JCL (z/OS Batch or IMS BMP) or runtime CLIST (for MVS TSO) that references the CSP/370RS Version 2.1 load library to use the load library for the EGL developer server product.

Installing the EGL server product

The EGL server product must be installed in a separate SMP/E zone and have different target libraries from CSP/370RS Version 2.1 and CSP/370AD Version 4.1. You should run CSP/370AD and CSP/370RS in a different CICS region or MVS/TSO system from the EGL server product.

If you placed any CSP/370RS Version 2.1 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 CSP/370RS Version 2.1 and the EGL server product because this can cause unpredictable results. Therefore, if you removed the CSP/370RS load modules from the CSP/370RS 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 CSP/370RS modules back into the CSP/370RS SELALMD load library.
2. Remove the CSP/370RS 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 server product SELALMD load library.

You cannot delete CSP/370RS Version 2.1 or CSP/370AD Version 4.1 until you complete both of the following tasks:
- Migrate all your generated applications and programs to use the EGL server product.
- Migrate all your development and generation to EGL.

After you have completed the above procedures, you can then delete both CSP/370AD Version 4.1 and CSP/370RS Version 2.1.

Generating programs, DataTables, and FormGroups

In general, programs, DataTables, and FormGroups that were generated with CSP/370AD Version 4.1 do not have to be generated again to use them with the EGL server product. However, we strongly recommend that you generate and test all your programs, DataTables, and FormGroups with EGL as a way of verifying that your source code migrated correctly.

If you decide not to generate and test everything, the following considerations apply:
- You must regenerate programs, DataTables, and FormGroups that were only generated with the CSP Version 3.3 COBOL generation facility and then moved forward to CSP/370RS Version 2.1 without regeneration. This consideration only applies to programs generated for IMS/VS, IMS BMP, or z/OS Batch. z/OS CICS programs are not affected because COBOL Generation for MVS CICS was not introduced until CSP/370AD Version 4.1.
• EGL COBOL generation eliminates the mapping services program for textForms in all environments and adds a FormGroup format module for the IMS environments. EGL generation also adds information to DataTables that can be used as edit routines, such as matchValid, matchInvalid, and rangeChk DataTables. The ability to share DataTables and FormGroups between an application generated with CSP/370AD Version 4.1 and a program generated with EGL depends on whether the generation was done by CSP/370AD Version 4.1 or EGL. The rules for using DataTables and FormGroups are as follows:
  • 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 validationDataTable 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 validationDataTable 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 any program, DataTable or FormGroup is generated with EGL, then you must use the EGL server product.

If you do not follow the above rules, you might receive the messages ELA00046P, ELA00051P, or ELA00208P; ABEND code ELAW in z/OS CICS; or ABEND code 1610 in non-CICS environments. Refer to the EGL Server Guide for more information about these messages and ABEND codes.
Appendix A. Converting CSP control information

There is no direct migration from CSP to EGL for control information (generation options, resource association, linkage table, bind commands, and link edit commands). In CSP, this information is stored in PDS members and, in some cases, in the application member of the MSL. In EGL, the information is stored as parts in .eglbld files. The following sections provide the correspondence, when available, between CSP options and EGL options. However, you should review the options in the EGL online helps or in the *EGL Generation Guide* because there are numerous new options that you might need to use.

All the tables that show the correspondence between CSP and EGL control information also include a VisualAge Generator 4.5 column. This column is included as an aid in finding additional details in the *VisualAge Generator to EGL Migration Guide*, Appendix B, section “Control Parts”. Note that CSP and VAGen options and values are not case sensitive. EGL options and values are case sensitive. Therefore it is best to use the EGL Build Parts Editor when creating your build control parts.

**CSP generation options**

In CSP, generation options are specified in several places:
- Through the user interface when you do online generation.
- Using the GENERATE command when you do batch generation.
- Using a generation options member in a PDS when you do batch generation. These options can also be specified on the GENERATE command.

Both batch and online generation permit you to save some of the generation options in the application member if you specify SAVE(YES) as a generation option. If you export the application member as an External Source Format file, the :genopts tag provides the generation option information. Except where noted in Table A1, the GENERATE command and the :genopts tag attributes are identical. However, the information stored in the application member reflects the last time you specified to save the information – which is not necessarily the information you used the last time you generated the application. Therefore, you should carefully review any information that was saved with the application member. None of the generate options listed in Table A2 are saved in the application member.

In addition, in CSP, when you generate an application, you can specify information about the tables and map groups that the program uses in the following way:
- Through the user interface when you do online generation.
- Using the SETGEN command when you do batch generation.

Either method enables you to specify the following information:
- For each map group, whether or not to generate the map group.
- For each table, whether or not to generate the table.
- For each table, the SHARED, RESIDENT, and KEEP AFTER USE properties.

In EGL, all the generation options are specified in build descriptor parts and stored in an .eglbld file. You can create a build descriptor part using the EGL Build Parts Editor. You can specify a default
build descriptor part at the file, package, EGL source folder, EGL project, or workbench level. When you generate, EGL uses the first default build descriptor part that it finds starting from the file and working outward toward the workbench.

In addition, in EGL, when you generate a program, EGL changes the way in which you specify information about the associated DataTables and FormGroups as follows:

- Use the `genFormGroup` and `genHelpFormGroup` build descriptor options to indicate whether you want to generate the program’s main FormGroup or help FormGroup respectively.
  
  **Recommendation:** Specify YES for both build descriptor options. This ensures that both FormGroups are generated whenever you generate a program.

- Use the `genDataTables` build descriptor option to indicate whether you want to generate all the DataTables used by the program or none of them. You cannot selectively specify which DataTables to generate.
  
  **Recommendation:** Specify NO for this build descriptor option. This avoids generating a DataTable with every program that uses it.

- For tables, the information about SHARED and RESIDENT is now in the `shared` and `resident` properties in the EGL DataTable part. The information about KEEP AFTER USE is now in the `deleteAfterUse` property in the program’s `use` declaration for the table.
  
  **Recommendation:** The migration tools might not migrate these properties correctly. See “Additional changes to make by hand” on page 43 for details about setting these properties for the EGL DataTable and program parts.

Table A1 shows the CSP generation options that you can only specify on the GENERATE command and the corresponding EGL build descriptor options. Table A2 shows the CSP generation options that you can specify either on the GENERATE command or in a generation options member in a PDS and the corresponding EGL build descriptor options. The VisualAge Generator 4.5 column is included as an aid in finding additional details in the VisualAge Generator to EGL Migration Guide, Appendix B, section “Control Parts”, subsection “Generation options part”.

In addition to the options shown in the tables, you must always include the following EGL build descriptor options to ensure the same processing as in CSP:

- `vagCompatibility="YES"`.
- `truncateExtraDecimals="YES"`.
- `destPort` – is required to specify the port to use when transferring generation outputs to the host.
- `projectID` -- by default, when you generate an EGL part for a z/OS runtime environment, the outputs of COBOL generation are transferred to data sets that use the same default naming convention that is used by CSP/370AD Version 4.1. In CSP, the high-level qualifier is specified by the USERID generation option. In EGL, the high-level qualifier is specified by the `projectID` build descriptor option. In both CSP/370AD Version 4.1 and EGL, the runtime environment is used as the second qualifier. To preserve your CSP-generated COBOL code, create new data sets for the EGL-generated COBOL code using the following naming conventions based on your runtime environment:
  
  - For IMS/VS and IMS BMP, change the `projectID` build descriptor option to specify a different high-level qualifier for the EGL data sets from what you specified in the CSP USERID generation option.
  - For z/OS CICS and z/OS Batch, the runtime environment names have changed from MVSCICS and MVS BATCH to ZOSCICS and ZOSBATCH. Therefore, if you use the default naming convention, you can keep the same `projectID` build descriptor and just allocate a new set of data sets using ZOSCICS and ZOSBATCH as the second qualifier.
You might also need to include the following EGL build descriptor options:

- **decimalSymbol** – required if you use a comma as the decimal indicator for your runtime environment.
- **clientCodeSet** and **serverCodeSet** – required if you need code page conversion from anything other than IBM-850 (Latin-1 country) on the workstation and IBM-037 (US English) on the host.
- **linkage** – required if you use an EGL linkage options part.
- **resourceAssociations** – required if you use an EGL resource associations part.
- **bind** – required only if you need a bind control template or a program-specific bind command for multiple runtime environments. The **bind** option is not required if the program-specific bind control part has the same part name as the program.
- **linkedit** – required only if you need a program-specific link edit command for multiple runtime environments. The **linkedit** option is not required if the program-specific link edit part has the same part name as the program.
- **templateDir** – required only if you need to customize the EGL-provided runtime JCL templates.
- **reservedWord** – required only if you need to provide additional reserved words.
- **genDirectory** – where to place the outputs of COBOL generation on the workstation before they are uploaded to the z/OS host.

**Recommendation:** Because there are numerous new EGL build descriptor options, it is best to review all the options and determine if there are additional options that you need to set. Create an .eglbld file and then use the EGL Build Parts Editor to create an EGL build descriptor part for each runtime environment. For a quick way of determining all the options that apply to a particular runtime environment, make the following changes in the EGL Build Parts

- Set the **Build option filter** drop down list to the **All** option for your runtime environment.
- Clear the **Show only specified options** checkbox.

You must also create a build descriptor part to use for debug. You can place multiple build descriptor parts in a single .eglbld file.

As a starting point, create one EGL build descriptor part for each CSP generation option PDS member. In CSP, you chained generation option PDS members using the OPTIONS(nextMemberName) option. Similarly, in EGL, you can chain build descriptor options using the nextBuildDescriptor="nextPartName" option. See “Storing control information” on page 45 for an example.

### Table A1. CSP GENERATE command options and EGL build descriptor options

<table>
<thead>
<tr>
<th>Cross System Product 4.1</th>
<th>VisualAge Generator 4.5</th>
<th>EGL</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BATCH</strong> (YES</td>
<td>NO)</td>
<td>Supported by creating a command file.</td>
</tr>
<tr>
<td><strong>Note:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- There is no equivalent :genopts information.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>FOLD</strong> (YES</td>
<td>NO)</td>
<td>/fold</td>
</tr>
<tr>
<td></td>
<td>/nofold</td>
<td></td>
</tr>
<tr>
<td><strong>JOBNAME</strong> (name)</td>
<td>/jobname=name</td>
<td>Not supported. Refer to the EGL online helps for $USERID, which is the closest replacement.</td>
</tr>
<tr>
<td><strong>LINES</strong> (55</td>
<td>number)</td>
<td>/lines=number</td>
</tr>
<tr>
<td>Cross System Product 4.1</td>
<td>VisualAge Generator 4.5</td>
<td>EGL</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------------</td>
<td>-----</td>
</tr>
<tr>
<td>LINKAGE(library.member)</td>
<td>/linkage=partName</td>
<td>linkage=&quot;partName&quot;</td>
</tr>
<tr>
<td>Note:</td>
<td></td>
<td>Note:</td>
</tr>
<tr>
<td>• EZEOP is the default library name.</td>
<td></td>
<td>• Linkage tables are stored in VAGen parts.</td>
</tr>
<tr>
<td>LOCVALID(YES</td>
<td>NO)</td>
<td>/locvalid</td>
</tr>
<tr>
<td>Note:</td>
<td></td>
<td>Note:</td>
</tr>
<tr>
<td>• The equivalent :genopts attribute is VALIDLOC.</td>
<td></td>
<td>• Linkage options information is stored in EGL parts.</td>
</tr>
<tr>
<td>MAPS(ALL</td>
<td>NONE )</td>
<td>/genmaps</td>
</tr>
<tr>
<td>Note:</td>
<td></td>
<td>genFormGroup=&quot;NO&quot;</td>
</tr>
<tr>
<td>• The equivalent :genopts attributes are MAPGRP1 and GENGRP1 for the program’s map group and MAPGRP2 and GENGRP2 for the program’s help map group.</td>
<td>/genhelpmaps</td>
<td>genHelpFormGroup=&quot;YES&quot;</td>
</tr>
<tr>
<td></td>
<td>/nogenmaps</td>
<td>genHelpFormGroup=&quot;NO&quot;</td>
</tr>
<tr>
<td></td>
<td>/nogenhelpmaps</td>
<td></td>
</tr>
<tr>
<td></td>
<td>genFormGroup=&quot;NO&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>genHelpFormGroup=&quot;YES&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>genHelpFormGroup=&quot;NO&quot;</td>
<td></td>
</tr>
<tr>
<td>Recommendation:</td>
<td></td>
<td>• Specify YES for both build descriptor options. This ensures that both FormGroups are generated whenever you generate a program.</td>
</tr>
<tr>
<td>MEMBER(name)</td>
<td>The part name is entered in the user interface or specified immediately after the GENERATE command.</td>
<td>You select the projects, folders, or files that you want to generate.</td>
</tr>
<tr>
<td>Note:</td>
<td></td>
<td>Note:</td>
</tr>
<tr>
<td>• There is no equivalent :genopts attribute, but the information is stored in the application part.</td>
<td></td>
<td>• Build descriptor options are stored in EGL parts</td>
</tr>
<tr>
<td>OPTIONS(library.member)</td>
<td>/options=partName</td>
<td>nextBuildDescriptor=&quot;partName&quot;</td>
</tr>
<tr>
<td>Note:</td>
<td></td>
<td>Note:</td>
</tr>
<tr>
<td>• EZEOP is the default library name.</td>
<td></td>
<td>• Generation options are stored in VAGen parts</td>
</tr>
<tr>
<td>PRINT(YES</td>
<td>NO)</td>
<td>/listing</td>
</tr>
<tr>
<td>Note:</td>
<td></td>
<td>Note:</td>
</tr>
<tr>
<td>• There is no equivalent :genopts information.</td>
<td></td>
<td>• Build descriptor options are stored in EGL parts</td>
</tr>
<tr>
<td>RESET(YES</td>
<td>NO)</td>
<td>Not supported.</td>
</tr>
<tr>
<td>Note:</td>
<td></td>
<td>Note:</td>
</tr>
<tr>
<td>• There is no equivalent :genopts information.</td>
<td></td>
<td>• Build descriptor options are stored in EGL parts</td>
</tr>
<tr>
<td>SAVE(YES</td>
<td>NO)</td>
<td>Not supported.</td>
</tr>
<tr>
<td>Cross System Product 4.1</td>
<td>VisualAge Generator 4.5</td>
<td>EGL</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-------------------------</td>
<td>-----</td>
</tr>
<tr>
<td>SQLVALID(YES</td>
<td>NO)</td>
<td>/sqlvalid</td>
</tr>
<tr>
<td>Note:</td>
<td>/nosqlvalid</td>
<td></td>
</tr>
<tr>
<td>• The equivalent :genopts attribute is VALIDSQL.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYSTEM(systemID)</td>
<td>/system=systemID</td>
<td>system=&quot;systemID&quot;</td>
</tr>
<tr>
<td>Where systemID is one of the following:</td>
<td>Where systemID is one of the following:</td>
<td>Where systemID is one of the following:</td>
</tr>
<tr>
<td>• MVSBATCH</td>
<td>• MVSBATCH</td>
<td>• ZOSBATCH</td>
</tr>
<tr>
<td>• MVSCICS</td>
<td>• MVSCICS</td>
<td>• ZOSCICS</td>
</tr>
<tr>
<td>• IMSBMP</td>
<td>• IMSBMP</td>
<td>• IMSBMP</td>
</tr>
<tr>
<td>• IMS/VS</td>
<td>• IMSVS</td>
<td>• IMSVS</td>
</tr>
<tr>
<td>• TSO</td>
<td>• TSO</td>
<td>• Not supported</td>
</tr>
<tr>
<td>• OS2CICS</td>
<td>• OS2CICS</td>
<td>• Not supported</td>
</tr>
<tr>
<td>Note:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• The system information is stored on the :targsys tag in the application member.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRANSID(‘txid’)</td>
<td>/transid=primaryID,restartID</td>
<td>startTransactionID=&quot;primaryID&quot; restartTransactionID=&quot;restartID&quot;</td>
</tr>
<tr>
<td>Note:</td>
<td></td>
<td>Note:</td>
</tr>
<tr>
<td>• The equivalent :genopts attribute is SEGTRAN.</td>
<td></td>
<td>The EGL startTransactionID option is the equivalent of the CSP TRANSID option.</td>
</tr>
<tr>
<td>TABLES(ALL</td>
<td>NONE)</td>
<td>/gentables</td>
</tr>
<tr>
<td></td>
<td>/nogentables</td>
<td>genDataTables=&quot;NO&quot;</td>
</tr>
<tr>
<td>Recommendation:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Specify NO for this build descriptor option. This avoids generating a DataTable with every program that uses it.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Table A2. CSP generation options and EGL build descriptor options

<table>
<thead>
<tr>
<th>Cross System Product 4.1</th>
<th>VisualAge Generator 4.5</th>
<th>EGL</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSISQL(YES</td>
<td>NO)</td>
<td>/ansisql</td>
</tr>
<tr>
<td>CHECKTYP(n)</td>
<td>/checktype = xxxx</td>
<td>checkType = “xxxx”</td>
</tr>
<tr>
<td>Where n is one of the following:</td>
<td>Where xxxx is one of the following:</td>
<td>Where xxxx is one of the following:</td>
</tr>
<tr>
<td>- 0</td>
<td>- none</td>
<td>- NONE</td>
</tr>
<tr>
<td>- 1</td>
<td>- low</td>
<td>- LOW</td>
</tr>
<tr>
<td>- 2</td>
<td>- all</td>
<td>- ALL</td>
</tr>
<tr>
<td>CICSDBCS(YES</td>
<td>NO)</td>
<td>/ciesdbs</td>
</tr>
<tr>
<td>COMMLVL(n)</td>
<td>/COMMENTLEVEL = n</td>
<td>commentLevel=&quot;n”</td>
</tr>
<tr>
<td>Where n is one of the following:</td>
<td>Where n has the same meanings as in CSP</td>
<td>Where n is one of the following:</td>
</tr>
<tr>
<td>- N - omit all comments except generation options</td>
<td>- 0 – no comments</td>
<td>- 0 – no comments</td>
</tr>
<tr>
<td>- 0 - minimum</td>
<td>- 1 – comments are included</td>
<td>- 1 – comments are included</td>
</tr>
<tr>
<td>- 1 - alias names, standard generation information</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- 2 - program and table prologs and function descriptions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- 3 - record prologs and item descriptions (the default)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- 4 - source statements and comments</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CONVT(xxxxxxxx)</td>
<td>/contable=xxxxxxxx</td>
<td>clientCodeSet=&quot;yyyyyyyyy”</td>
</tr>
<tr>
<td>Note:</td>
<td>Note:</td>
<td>serverCodeSet=&quot;zzzzzzzz”</td>
</tr>
<tr>
<td>- This option only applies to CICS OS/2.</td>
<td>- The values are the same as in CSP, except that ELACNENP (upper case English) is no longer supported. Use ELACNENU (mixed case English) as the replacement.</td>
<td>Note:</td>
</tr>
<tr>
<td>DATA(24</td>
<td>31)</td>
<td>/data = 24</td>
</tr>
<tr>
<td>ERRDEST(destinationID)</td>
<td>/errdest=destinationID</td>
<td>errorDestination=&quot;destinationID”</td>
</tr>
<tr>
<td>EXECJCL(YES</td>
<td>NO)</td>
<td>/runfile</td>
</tr>
<tr>
<td></td>
<td>/norunfile</td>
<td>genRunFile=&quot;NO”</td>
</tr>
<tr>
<td>FASTPATH(YES</td>
<td>NO)</td>
<td>/fastpath</td>
</tr>
<tr>
<td></td>
<td>/nofastpath</td>
<td>imsFastPath=&quot;NO”</td>
</tr>
<tr>
<td>FFFF(YES</td>
<td>NO)</td>
<td>/endcommarea</td>
</tr>
<tr>
<td></td>
<td>/noendcommarea</td>
<td>endCommarea=&quot;NO”</td>
</tr>
</tbody>
</table>

Note: Refer to the VisualAge Generator to EGL Migration Guide, Appendix B, section “Control Parts”, subsection “Conversion table names used in generation options parts” for details of the mapping between CONVT and the two EGL build descriptor options.
<table>
<thead>
<tr>
<th>Cross System Product 4.1</th>
<th>VisualAge Generator 4.5</th>
<th>EGL</th>
</tr>
</thead>
<tbody>
<tr>
<td>INEDIT(ALL</td>
<td>INONLY)</td>
<td>/inedit=all</td>
</tr>
<tr>
<td>INIADDWS=YES</td>
<td>NO)</td>
<td>/initaddws</td>
</tr>
<tr>
<td>INITRECD(YES</td>
<td>NO)</td>
<td>/initrecd</td>
</tr>
<tr>
<td>JOBCARD(library:member)</td>
<td>/jobcard=fileName</td>
<td>Not supported. The build servers handle the job card.</td>
</tr>
<tr>
<td>LEFTJUST=YES</td>
<td>NO)</td>
<td>/leftjust</td>
</tr>
<tr>
<td>LOG(logID</td>
<td>NO)</td>
<td>/log=logID</td>
</tr>
<tr>
<td>MATH(CSP</td>
<td>COBOL)</td>
<td>/math=cspae</td>
</tr>
<tr>
<td>MFSDEV(mfsDeviceInfo)</td>
<td>/mfsdev=(mfsDeviceInfo)</td>
<td>mfsDevice</td>
</tr>
<tr>
<td>MFSEATTR(YES</td>
<td>NO</td>
<td>mfsExtendedAttr=&quot;YES&quot;</td>
</tr>
<tr>
<td>MFSFEAT(IGNORE</td>
<td>NOIGNORE)</td>
<td>/mfsignore</td>
</tr>
</tbody>
</table>

**Note:**
- The CSP `mfsDeviceInfo` is identical in format to the VAGen `mfsDeviceInfo`. Refer to the *VisualAge Generator Server Guide for MVS, VSE, and VM* for details of the syntax.

**Note:** Refer to the *VisualAge Generator to EGL Migration Guide*, Appendix B, section on “Control Parts”, subsection on “Generation options part” for details on how the VAGen to EGL migration tool converts this option. You need to make the same conversions when determining the information to enter in the EGL Build Parts Editor.
- When you edit build descriptor parts using the EGL Build Parts Editor, there is a special tool bar icon called “Show MFS Device Properties” to assist you in entering the information in the correct format.

**Note:**
<table>
<thead>
<tr>
<th>Cross System Product 4.1</th>
<th>VisualAge Generator 4.5</th>
<th>EGL</th>
</tr>
</thead>
<tbody>
<tr>
<td>MFSTEST(YES</td>
<td>NO)</td>
<td>/mfstest</td>
</tr>
<tr>
<td></td>
<td>/nomfstest</td>
<td>mfsUseTestLibrary=&quot;NO&quot;</td>
</tr>
<tr>
<td>MSP(whatever)</td>
<td>/msp=whatever</td>
<td>formatServicePgmType=&quot;whatever&quot;</td>
</tr>
<tr>
<td>Where whatever is one or more of the following separated by commas:</td>
<td>Where whatever is one or more of the following separated by commas:</td>
<td>Where whatever is one or more of the following separated by commas:</td>
</tr>
<tr>
<td>• ALL (the default)</td>
<td>• all</td>
<td>• ALL</td>
</tr>
<tr>
<td>• GSAM</td>
<td>• gsam</td>
<td>• GSAM</td>
</tr>
<tr>
<td>• MFS</td>
<td>• mfs</td>
<td>• MFS</td>
</tr>
<tr>
<td>• SEQ</td>
<td>• seq</td>
<td>• SEQ</td>
</tr>
<tr>
<td>NLS(whatever)</td>
<td>/targetnls=whatever</td>
<td>targetNLS=&quot;whatever&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EGL uses the same 3-character NLS codes as CSP. However, EGL does not support ENP (uppercase English).</td>
</tr>
<tr>
<td>NULLFILL(YES</td>
<td>NO)</td>
<td>/nullfill</td>
</tr>
<tr>
<td></td>
<td>/nonnullfill</td>
<td>fillWithNulls=&quot;NO&quot;</td>
</tr>
<tr>
<td>NUMOVFL(YES</td>
<td>NO)</td>
<td>/numovfl</td>
</tr>
<tr>
<td></td>
<td>/nonumovfl</td>
<td>checkNumericOverflow=&quot;NO&quot;</td>
</tr>
<tr>
<td>OPTIONS(library,member)</td>
<td>/options=partName</td>
<td>nextBuildDescriptor=&quot;partName&quot;</td>
</tr>
<tr>
<td>Note:</td>
<td>Note:</td>
<td>Note:</td>
</tr>
<tr>
<td>• EZEOP is the default library.</td>
<td>• Generation options are stored in VAGen parts.</td>
<td>• Build descriptor options are stored in EGL parts.</td>
</tr>
<tr>
<td>PREPJCL(YES</td>
<td>NO)</td>
<td>/preppfile</td>
</tr>
<tr>
<td></td>
<td>/nopreppfile</td>
<td>buildPlan=&quot;NO&quot;</td>
</tr>
<tr>
<td>RECOVERY(REQ</td>
<td>NOREQ)</td>
<td>/recovery</td>
</tr>
<tr>
<td></td>
<td>/norecovery</td>
<td>restoreCurrentMsgOnError=&quot;NO&quot;</td>
</tr>
<tr>
<td>RT(transactionID)</td>
<td>/rt=transactionID</td>
<td>returnTransaction=&quot;transactionID&quot;</td>
</tr>
<tr>
<td>SETFULL(YES</td>
<td>NO)</td>
<td>/setfull</td>
</tr>
<tr>
<td></td>
<td>/noSetfull</td>
<td>setFormItemFull=&quot;NO&quot;</td>
</tr>
<tr>
<td>SPA(size,ADF,position)</td>
<td>/spa=size,ADF,position</td>
<td>spaSize=&quot;size&quot;</td>
</tr>
<tr>
<td>Note:</td>
<td>Note:</td>
<td>Note:</td>
</tr>
<tr>
<td>• size can be 0.</td>
<td>• size, ADF, and position have the same meanings as in CSP.</td>
<td>The single CSP option splits into 3 EGL options.</td>
</tr>
<tr>
<td>• ADF and position are optional.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPZERO(YES</td>
<td>NO)</td>
<td>/spzero</td>
</tr>
<tr>
<td></td>
<td>/nospzero</td>
<td>spacesZero=&quot;NO&quot;</td>
</tr>
<tr>
<td>SUBMJCL(library,member)</td>
<td>Not supported</td>
<td>Not supported</td>
</tr>
</tbody>
</table>
### Cross System Product 4.1

**SYMPARM(symbolID, 'value')**

/symparm=symbolID, 'value'

**Note:**
- See “Symbolic parameters” on page 82 for the correspondence between the CSP and EGL predefined symbols.
- Also refer to the *VisualAge Generator to EGL Migration Guide*, Appendix B, section on “Control Parts”, subsections “Generation options part” and “Symbolic parameters” for details on how the VAGen to EGL migration tool converts this option.
- When you edit build descriptor parts using the EGL Build Parts Editor, there is a special area called “Symbolic parameters” to assist you in entering the information in the correct format.

### VisualAge Generator 4.5

**SYNCDXFR(YES | NO)**

/syncdxfr
/nosyncdxfr

**Note:**

- `synchOnPgmTransfer=“YES”`
- `synchOnPgmTransfer=“NO”`

**SYNCXFER(YES | NO)**

/syncxfer
/nosyncxfer

**Note:**

- `synchOnTrxTransfer=“YES”`
- `synchOnTrxTransfer=“NO”`

**SYSCODES(YES | NO)**

/syscodes
/nosyscodes

**Note:**

- `sysCodes=“YES”`
- `sysCodes=“NO”`

**TWAOFF(offset)**

/twaoff=offset

**Note:**

- `twaOffset=“offset”`

**USERID(xxxxxxxx)**

/projectid=xxxxxxxx

**Note:**
- See the detailed information about “projectID” earlier in this section.

**VALIDMIX(YES | NO)**

/validmix
/novalidmix

**Note:**

- `validateMixedItems=“YES”`
- `validateMixedItems=“NO”`

**WORKDB(xxxx)**

/workdb=xxxx

**Note:**

Where `xxxx` is one of the following:
- AUX
- MAIN
- DLI
- SQL

### EGL

**symbolicParameter**

**Note:**

- `symbolID` and `value` are the symbolic parameters.

<table>
<thead>
<tr>
<th>Cross System Product 4.1</th>
<th>VisualAge Generator 4.5</th>
<th>EGL</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYMPARM(symbolID, 'value')</td>
<td>/symparm=symbolID, 'value'</td>
<td>symbolicParameter</td>
</tr>
<tr>
<td>SYNCDXFR(YES</td>
<td>NO)</td>
<td>/syncdxfr</td>
</tr>
<tr>
<td>SYNCXFER(YES</td>
<td>NO)</td>
<td>/syncxfer</td>
</tr>
<tr>
<td>SYSCODES(YES</td>
<td>NO)</td>
<td>/syscodes</td>
</tr>
<tr>
<td>TWAOFF(offset)</td>
<td>/twaoff=offset</td>
<td>twaOffset=“offset”</td>
</tr>
<tr>
<td>USERID(xxxxxxxx)</td>
<td>/projectid=xxxxxxxx</td>
<td>projectID=“xxxxxxx”</td>
</tr>
<tr>
<td>VALIDMIX(YES</td>
<td>NO)</td>
<td>/validmix</td>
</tr>
<tr>
<td>WORKDB(xxxx)</td>
<td>/workdb=xxxx</td>
<td>workDBType=“xxxx”</td>
</tr>
</tbody>
</table>

Where `xxxx` is one of the following:
- AUX
- MAIN
- DLI
- SQL
Linkage table

In CSP, the LINKAGE option specifies the library (DD name of a PDS) and member that contains the linkage table you want to use during generation. If none of your generation options specify the LINKAGE option, you can skip this section.

In EGL, the linkage information is specified in a linkage options part and stored in an .eglbld file. You can create this part using the EGL Build Parts Editor. You specify which linkage options part to use by setting the linkage build descriptor option.

In CSP, there are 4 types of entries in the linkage table. In EGL there are 4 equivalent types of entries in a linkage options part, but some of the have different names as shown in Table A3.

Table A3. CSP linkage table entries and corresponding EGL entries

<table>
<thead>
<tr>
<th>Cross System Product 4.1</th>
<th>VisualAge Generator 4.5</th>
<th>EGL</th>
</tr>
</thead>
<tbody>
<tr>
<td>:calllink</td>
<td>:calllink</td>
<td>callLink</td>
</tr>
<tr>
<td>:crtxlink</td>
<td>:crtxlink</td>
<td>asynchLink</td>
</tr>
<tr>
<td>:dxfrlink</td>
<td>:dxfrlink</td>
<td>transferLink, specifically the transferToProgram entry</td>
</tr>
<tr>
<td>:filelink</td>
<td>:filelink</td>
<td>fileLink</td>
</tr>
</tbody>
</table>

Recommendation:

- To create an EGL linkage options part to use for generation, note that the CSP linkage table options are a subset of those supported in VAGen. Therefore, follow these steps:
  1. Download the CSP linkage table file to a workstation file.
  2. Create a VAGen linkage table part.
     
     **Note:** For step by step details of how to create a VAGen control part, refer to VisualAge Generator Migration Guide, Chapter 32 “Completing your migration on Smalltalk”, section “Defining control information”.
  3. From the Parts Editor window, click File -> Read from File to retrieve your CSP linkage table.
  4. Save the VAGen linkage table part.
  5. Export a VAGen External Source Format file for the linkage table part and use the VAGen to EGL Migration Tool Single File Mode to migrate the linkage table part.
  6. Use the EGL Build Parts Editor to update the following entry:
     - For the callLink entry, set the EGL location property if the CSP location value was set to (or defaulted to) CICS.
     - To create the EGL linkage options parts for debug, review the EGL linkage information in the online helps and determine which programs, communication protocols, and so on that you want to use when debugging your EGL programs. You should only need to create EGL callLink entries for debugging.

The following sections describe each of the entries and the correspondence between CSP and EGL.
When you generate for z/OS, you only need EGL \texttt{callLink} entries in the same situations in which a CSP :calllink entry is required. Therefore, if you do not have a CSP linkage table or your CSP linkage table does not include :calllink entries, you do not need to create an EGL linkage options part and \texttt{callLink} entries to use for generation. However, for debug purposes, if you need to call a non-EGL program or a previously generated EGL program on the z/OS host, you must create a linkage options part to use for debug and include a \texttt{callLink} entry for each z/OS program you need to call from the debug facility. This debug linkage options part provides information about the communications protocol to use to call the non-EGL program on z/OS. The debug linkage options parts are beyond the scope of this paper. If you need to create a debug linkage options part, refer to the \textit{EGL Generation Guide} or the EGL online helps.

Table \ref{tab:calllink} shows the CSP :calllink options and the corresponding EGL \texttt{callLink} options. The VisualAge Generator 4.5 column is included as an aid in finding additional details in the \textit{VisualAge Generator to EGL Migration Guide}, Appendix B, section “Control Parts”, subsection “Linkage table parts”, subtopic “calllink”.

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|}
\hline
\textbf{Cross System Product 4.1} & \textbf{VisualAge Generator 4.5} & \textbf{EGL} \\
\hline
\texttt{:calllink} & \texttt{:calllink} & \texttt{callLink} \\
\texttt{linktype=xxxx} & \texttt{linktype=xxxx} & \texttt{Type of call, where the EGL equivalent options are the following:} \\
Where \textit{xxxx} is one of the following: & Where \textit{xxxx} is one of the following: & \begin{itemize}
\item \texttt{localCall}
\item \texttt{localCall}
\item \texttt{remoteCall}
\end{itemize} \\
\begin{itemize}
\item DYNAMIC
\item STATIC
\item CICSLINK
\item REMOTE
\end{itemize} & \begin{itemize}
\item \texttt{dynamic}
\item \texttt{static}
\item \texttt{cicslink}
\item \texttt{remote}
\end{itemize} \\
\hline
\texttt{applname=programName} & \texttt{applname=programName} & \texttt{pgmName="programName"} \\
\textbf{Note:} & \textbf{Note:} & \textbf{Note:} \\
\begin{itemize}
\item An * can be used as a wild card in the last character of the \texttt{programName}.
\end{itemize} & \begin{itemize}
\item An * can be used as a wild card in the last character of the \texttt{programName}.
\end{itemize} & \begin{itemize}
\item An * can be used as a wild card in the last character of the \texttt{programName}.
\end{itemize} \\
\hline
\end{tabular}
\caption{CSP :calllink options and EGL callLink options}
\end{table}
<table>
<thead>
<tr>
<th>Cross System Product 4.1</th>
<th>VisualAge Generator 4.5</th>
<th>EGL</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>linktype=xxxx</code></td>
<td><code>linktype=xxxx</code></td>
<td><code>linkType=xxxx</code></td>
</tr>
<tr>
<td>Where <code>xxxx</code> is one of the following:</td>
<td>Where <code>xxxx</code> is one of the following:</td>
<td>Where <code>xxxx</code> is one of the following:</td>
</tr>
<tr>
<td>• DYNAMIC</td>
<td>• dynamic</td>
<td>• DYNAMIC</td>
</tr>
<tr>
<td>• STATIC</td>
<td>• static</td>
<td>• STATIC</td>
</tr>
<tr>
<td>• CICSLINK</td>
<td>• cicslink</td>
<td>• CICSLINK</td>
</tr>
</tbody>
</table>

**Note:**
- This conversion for `linkType` only applies if it is an EGL `localCall` entry.

<table>
<thead>
<tr>
<th>parmform=xxxx</th>
<th>parmform=xxxx</th>
<th>parmform=“xxxx”</th>
</tr>
</thead>
<tbody>
<tr>
<td>Where <code>xxxx</code> is one of the following:</td>
<td>Where <code>xxxx</code> is one of the following:</td>
<td>Where <code>xxxx</code> is one of the following:</td>
</tr>
<tr>
<td>• OSLINK</td>
<td>• oslink</td>
<td>• OSLINK</td>
</tr>
<tr>
<td>• COMMPTR</td>
<td>• commptr</td>
<td>• COMMPTR</td>
</tr>
<tr>
<td>• COMMDATA</td>
<td>• commdata</td>
<td>• COMMDATA</td>
</tr>
<tr>
<td>• CICOSLINK</td>
<td>• cicsoslink</td>
<td>• CICOSLINK</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>contable=xxxx</th>
<th>contable=xxxx</th>
<th>conversionTable=“xxxx”</th>
</tr>
</thead>
<tbody>
<tr>
<td>Where <code>xxxx</code> is one of the following:</td>
<td>Where <code>xxxx</code> is one of the following:</td>
<td>Where <code>xxxx</code> is one of the following:</td>
</tr>
<tr>
<td>• <code>conversionTableName</code></td>
<td>• <code>conversionTableName</code></td>
<td>• <code>conversionTableName</code></td>
</tr>
<tr>
<td>• ‘*’</td>
<td>• ‘*’</td>
<td>• ‘*’</td>
</tr>
<tr>
<td>• EZECONVT</td>
<td>• EZECONVT</td>
<td>• PROGRAMCONTROLLED</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>location=xxxx</th>
<th>location=xxxx</th>
<th>location=“xxxx”</th>
</tr>
</thead>
<tbody>
<tr>
<td>Where <code>xxxx</code> is one of the following:</td>
<td>Where <code>xxxx</code> is one of the following:</td>
<td>Where <code>xxxx</code> is one of the following:</td>
</tr>
<tr>
<td>• CICS</td>
<td>• <code>systemName</code></td>
<td>• CICS</td>
</tr>
<tr>
<td>• EZELOC</td>
<td>• EZELOC</td>
<td>• PROGRAMCONTROLLED</td>
</tr>
</tbody>
</table>
**:crtxLink entry**

When you generate for z/OS, you only need EGL `asynchLink` entries in the same situations in which a CSP :crtxlink entry is required. Therefore, if you do not have a CSP linkage table or your CSP linkage table does not include :crtxlink entries, you do not need to create an EGL linkage options part and `asynchLink` entries to use for generation. The `asynchLink` entry is not used by the EGL debugger, so do not include an `asynchLink` entry in your EGL linkage options parts that are used for debug.

Table A5 shows the CSP :crtxlink options and the corresponding EGL `asynchLink` options. The VisualAge Generator 4.5 column is included as an aid in finding additional details in the VisualAge Generator to EGL Migration Guide, Appendix B, section “Control Parts”, subsection “Linkage table parts”, subtopic “crtxlink”.

### Table A5. CSP :crtxlink options and EGL asynchLink options

<table>
<thead>
<tr>
<th>Cross System Product 4.1</th>
<th>VisualAge Generator 4.5</th>
<th>EGL</th>
</tr>
</thead>
<tbody>
<tr>
<td>:crtxlink</td>
<td>:crtxlink</td>
<td>asynchLink</td>
</tr>
<tr>
<td><code>linktype=xxxx</code></td>
<td><code>linktype=xxxx</code></td>
<td>Type of invocation, where the EGL equivalent options are the following:</td>
</tr>
<tr>
<td>Where <code>xxxx</code> is one of the following:</td>
<td>Where <code>xxxx</code> is one of the following:</td>
<td></td>
</tr>
<tr>
<td>• LOCAL</td>
<td>• <code>local</code></td>
<td>• localAsynch</td>
</tr>
<tr>
<td>• REMOTE</td>
<td>• remote</td>
<td>• remoteAsynch</td>
</tr>
<tr>
<td><code>recdname=recordName</code></td>
<td><code>recdname=recordName</code></td>
<td><code>recordName=&quot;recordName&quot;</code></td>
</tr>
<tr>
<td><code>contable=xxxx</code></td>
<td><code>contable=xxxx</code></td>
<td><code>conversionTable=&quot;xxxx&quot;</code></td>
</tr>
<tr>
<td>Where <code>xxxx</code> is one of the following:</td>
<td>Where <code>xxxx</code> is one of the following:</td>
<td></td>
</tr>
<tr>
<td>• <code>conversionTableName</code></td>
<td>• <code>conversionTableName</code></td>
<td></td>
</tr>
<tr>
<td>• &quot;*&quot;</td>
<td>• &quot;*&quot;</td>
<td></td>
</tr>
<tr>
<td>• EZECONVT</td>
<td>• EZECONVT</td>
<td></td>
</tr>
<tr>
<td><code>location=xxxx</code></td>
<td><code>location=xxxx</code></td>
<td><code>locationSpec=&quot;xxxx&quot;</code></td>
</tr>
<tr>
<td>Where <code>xxxx</code> is one of the following:</td>
<td>Where <code>xxxx</code> is one of the following:</td>
<td></td>
</tr>
<tr>
<td>• CICS</td>
<td>• CICS</td>
<td></td>
</tr>
<tr>
<td>• EZELOC</td>
<td>• EZELOC</td>
<td></td>
</tr>
</tbody>
</table>
When you generate for a z/OS environment, you only need EGL `transferToProgram` entries in the same situations in which a CSP :dxfrlink entry is required. Therefore, if you do not have a CSP linkage table or your CSP linkage table does not include :dxfrlink entries, you do not need to create an EGL linkage options part with `transferToProgram` entries to use for generation. You should not need to include a `transferToProgram` entry in your EGL linkage options parts that are used for debug.

Table A6 shows the CSP :dxfrlink options and the corresponding EGL `transferToProgram` options. The VisualAge Generator 4.5 column is included as an aid in finding additional details in the VisualAge Generator to EGL Migration Guide, Appendix B, section “Control Parts”, subsection “Linkage table parts”, subtopic “dxfrlink”.

**Note:** The EGL `transferToProgram` is one of two entries in the EGL `transferLink` category when you use the EGL Build Parts Editor to edit a linkage options part. The other category, `transferToTransaction`, is new for EGL. In general, you do not need to create a `transferToTransaction` entry unless you are generating Java.

**Table A6. CSP :dxfrlink options and EGL transferToProgram options in transferLink**

<table>
<thead>
<tr>
<th>Cross System Product 4.1</th>
<th>VisualAge Generator 4.5</th>
<th>EGL</th>
</tr>
</thead>
<tbody>
<tr>
<td>:dxfrlink</td>
<td>:dxfrlink</td>
<td>transferToProgram</td>
</tr>
<tr>
<td>fromappl=fromProgramName</td>
<td>fromappl=fromProgramName</td>
<td>fromPgm=&quot;fromProgramName&quot;</td>
</tr>
<tr>
<td>toappl=toProgramName</td>
<td>toappl=toProgramName</td>
<td>toPgm=&quot;toProgramName&quot;</td>
</tr>
<tr>
<td>linktype=xxxx</td>
<td>linktype=xxxx</td>
<td>linkType=&quot;xxxx&quot;</td>
</tr>
</tbody>
</table>

Where ```xxxx``` is one of the following:

- **DYNAMIC**
- **STATIC**
- **NONCSP**

Where ```xxxx``` is one of the following:

- dynamic
- static
- noncsp

Where ```xxxx``` is one of the following:

- **DYNAMIC**
- **STATIC**
- **EXTERNALLYDEFINED**
When you generate for a z/OS environment, you only need EGL fileLink entries in the same situations in which a CSP :filelink entry is required. Therefore, if you do not have a CSP linkage table or your CSP linkage table does not include :filelink entries, you do not need to create an EGL linkage options part and fileLink entries to use for generation. The fileLink entry is not used by the EGL debugger, so do not include a fileLink entry in your EGL linkage options parts that are used for debug. If you need to access serial, indexed, or relative VSAM files on your z/OS system, see “Accessing VSAM files” on page 55 for information.

Table A7 shows the CSP :filelink options and the corresponding EGL fileLink options. The VisualAge Generator 4.5 column is included as an aid in finding additional details in the VisualAge Generator to EGL Migration Guide, Appendix B, section “Control Parts”, subsection “Linkage table parts”, subtopic “filelink”.

<table>
<thead>
<tr>
<th>Cross System Product 4.1</th>
<th>VisualAge Generator 4.5</th>
<th>EGL</th>
</tr>
</thead>
<tbody>
<tr>
<td>linktype=xxxx</td>
<td>linktype=xxxx</td>
<td>Type of file, where the EGL equivalent options are as follows:</td>
</tr>
<tr>
<td>Where xxxx is one of the following:</td>
<td>Where xxxx is one of the following:</td>
<td></td>
</tr>
<tr>
<td>• LOCAL</td>
<td>• LOCAL</td>
<td>• localFile</td>
</tr>
<tr>
<td>• REMOTE</td>
<td>• REMOTE</td>
<td>• remoteFile</td>
</tr>
<tr>
<td>filename=fileName</td>
<td>filename=fileName</td>
<td>fileName=fileName</td>
</tr>
<tr>
<td>contable=xxxx</td>
<td>contable=xxxx</td>
<td>conversionTable=&quot;xxxx&quot;</td>
</tr>
<tr>
<td>Where xxxx is one of the following:</td>
<td>Where xxxx is one of the following:</td>
<td></td>
</tr>
<tr>
<td>• conversionTableName</td>
<td>• conversionTableName</td>
<td>• conversionTableName</td>
</tr>
<tr>
<td>• ‘*’</td>
<td>• ‘*’</td>
<td>• ‘*’</td>
</tr>
<tr>
<td>• EZECONVT</td>
<td>• EZECONVT</td>
<td>• PROGRAMCONTROLLED</td>
</tr>
<tr>
<td>location=xxxx</td>
<td>location=xxxx</td>
<td>locationSpec=&quot;xxxx&quot;</td>
</tr>
<tr>
<td>Where xxxx is one of the following:</td>
<td>Where xxxx is one of the following:</td>
<td></td>
</tr>
<tr>
<td>• CICS</td>
<td>• CICS</td>
<td>• CICS</td>
</tr>
<tr>
<td>• EZELOC</td>
<td>• EZELOC</td>
<td>• PROGRAMCONTROLLED</td>
</tr>
</tbody>
</table>
**Resource associations**

In CSP, resource association information for serial, indexed, relative, and print files is specified in one of two ways:

- Through the user interface when you do online generation.
- Using the ASSOCIATE command when you do batch generation.

Either method permits you to save the resource association information in the application member if you specify SAVE(YES) as a generation option. If you export the application member as an External Source Format file, the :genfile tag provides the resource association information. Except where noted in Table A8, the ASSOCIATE command and the :genfile tag attributes are identical. However, the information stored in the application member reflects the last time you specified to save the information – which is not necessarily the information you used the last time you generated the application. Therefore, you should carefully review any information that was saved with the application member.

In EGL, the resource association information is specified in a resource associations part and stored in an .eglbld file. You can create this part using the EGL Build Parts Editor. You specify which resource associations part to use by setting the `resourceAssociations` build descriptor option.

When you generate for a z/OS environment, you only need an EGL resource association part in the same situations in which you provided CSP resource association information. For example, if you use the default resource association in CSP, then you do not need to create an EGL resource association part for generation. However, if you need to access serial, indexed, or relative VSAM files on your z/OS system when you debug programs, see “Accessing VSAM files” on page 55 for information.

**Recommendation:**

- To create the EGL resource association parts for generation, export the External Source Format for each CSP application that uses serial, indexed, relative, or print output. You only need to export the application members. Review the External Source Format file for the :genfile tag to obtain the last resource association information that you saved. Create the EGL resource association information in the following way:
  - Use the EGL Build Parts Editor to create an EGL resource association part.
  - Use the information from the :genfile tag in the External Source Format file and the information in Table A8 to create the entries in the EGL resource association part. If the system attribute is not specified on the :genfile tag, use the SYSTEM value specified on the :targsys tag instead. Create the entries for the file names in the following way:
    - If you generate a program for multiple runtime environments, create one entry for the file name with one `system` subentry for each runtime environment.
    - If the file names used by two CSP applications are different, you can put the information for the files into the same EGL resource association part.
    - If the file names used by two CSP applications are the same and the runtime environments are the same, but the resource association information is different, you must create different EGL resource association parts. If the runtime environments are
different, create one entry for the file name with one system subentry for each runtime environment.

- You can use an * as a wild card at the end of the file name to reduce the number of entries.
- You must also provide resource association information for debug. You can do this in one of two ways:
  - If you do not generate for a Windows operating system, you can include the debug information in the same resource association part that you use for generation. Simply include an additional system subentry for the file name, set the system to win and specify the fileType and systemName for the Windows operating system.
  - If you plan to generate for the Windows operating system, create a separate resource association part to use for debug. This enables you to have one set of file names to use for your generated Java code and a different set of file names to use for debugging.

Note: Even if you select the EGL Debug preference Set systemType to DEBUG, the system for the resource association is win when you run the debugger on a Windows operating system.

Table A8 shows the CSP ASSOCIATE command options and the corresponding EGL resource association options. The VisualAge Generator 4.5 column is included as an aid in finding additional details in the VisualAge Generator to EGL Migration Guide, Appendix B, section “Control Parts”, subsection “Resource association part”.

Table A8. CSP ASSOCIATE command options and EGL resource associations parts

<table>
<thead>
<tr>
<th>Cross System Product 4.1</th>
<th>VisualAge Generator 4.5</th>
<th>EGL</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILE(filename</td>
<td>EZEPRINT)</td>
<td>FILE(filename</td>
</tr>
<tr>
<td>Note:</td>
<td></td>
<td>fileName=&quot;printer&quot;</td>
</tr>
<tr>
<td>• FILENAME is the equivalent option for the :genfile tag in External Source Format.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Not available.</td>
<td>/system=systemID</td>
<td>This is the EGL target system. The corresponding systemID values are the following:</td>
</tr>
<tr>
<td>Note:</td>
<td></td>
<td>• ZOSBATCH</td>
</tr>
<tr>
<td>• For online generation, the target system is derived from the system you select.</td>
<td></td>
<td>• ZOSCICS</td>
</tr>
<tr>
<td>• For batch generation, the target system is specified on the next GENERATE command that follows the ASSOCIATE command.</td>
<td></td>
<td>• IMSBMP</td>
</tr>
<tr>
<td>• SYSTEM is the equivalent option for the :genfile tag in the External Source Format. If the SYSTEM attribute is not specified on the :genfile tag, use the SYSTEM value specified on the :targsys tag instead.</td>
<td></td>
<td>• IMSVS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Not supported</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Not supported</td>
</tr>
</tbody>
</table>
### Bind commands

If you do not use SQL, you can skip this section.

In CSP, the EZEBIND DD statement in the generation job specifies the library (PDS) that contains the DB2 BIND commands. If there is already a member in the library with the same name as the CSP application, generation assumes that this is an application-specific bind command that you have created and does not replace the member. If there is no member with the same name as the CSP application, generation uses a bind control template to create a bind command for the application. By default, the bind control templates bind plans and assume that there is one CSP application in each DB2 plan. The plan name is the same as the CSP application name. The specific template used by generation varies based on the type of program, the runtime environment, and generation options.
The EZEJTMP DD statement in the generation job specifies the library (PDS) that contains the bind command templates. You might have modified the default templates to bind packages.

Note that you might have used both techniques in CSP – modifying the templates for most programs and providing an application-specific bind command for a few programs that have special requirements. In either case, all of the resulting bind commands should be in the library specified by the EZEBIND DD statement in your generation JCL. You might need to compare each member in this library to the original templates to determine if it is an application-specific bind command or a bind command generated from the templates. Note that the EZEBIND DD statement generally points to a different library for each runtime environment, so you might need to check for the same application in multiple libraries.

By default, EGL does not use bind control templates. Instead, EGL automatically creates a DB2 BIND command that binds a plan and assumes that there is one EGL program in the plan. Similar to CSP, the plan name is the same as the EGL program name. You can change the EGL bind commands in either of two ways:

- Create an EGL Bind Control part to use as a template for creating the DB2 BIND commands. For example, if you bind packages in CSP, you should use this technique. Follow the recommendation below for EGL bind control templates.
- Create an EGL Bind Control part to use for a specific program. If you wrote your own DB2 BIND commands in CSP, we strongly recommend that you consider changing to bind packages so that you no longer need to use program-specific bind control parts in EGL. If you must create program-specific BIND commands, follow the recommendation below for EGL program-specific bind control parts.

**Recommendation for EGL Bind control templates:**
Table A9 shows the CSP runtime environments and the corresponding bind command templates. Some of the templates are used for multiple environments. Except for the location of your CSP BIND command templates and the template names, the steps required for this technique are the same as for a customer migrating from VisualAge Generator. Refer to the *VisualAge Generator to EGL Migration Guide*, Chapter 9 “Completing your migration”, section “Establishing a bind control part to use as a template” for details.

<table>
<thead>
<tr>
<th>Environment and Details</th>
<th>Cross System Product 4.1 BIND Template</th>
</tr>
</thead>
<tbody>
<tr>
<td>MVS CICS</td>
<td></td>
</tr>
<tr>
<td>With DB2</td>
<td>EZEBINDA</td>
</tr>
<tr>
<td>MVS Batch</td>
<td></td>
</tr>
<tr>
<td>DL/I and DB2</td>
<td>EZEBINDA</td>
</tr>
<tr>
<td>DB2 only</td>
<td>EZEBINDN</td>
</tr>
<tr>
<td>IMS/VS</td>
<td></td>
</tr>
<tr>
<td>DL/I with DB2 work database</td>
<td>EZEBINDR</td>
</tr>
<tr>
<td>DL/I and DB2 with DB2 work database</td>
<td>EZEBIND</td>
</tr>
<tr>
<td>DL/I and DB2 without work database</td>
<td>EZEBINDA</td>
</tr>
</tbody>
</table>
Recommendation for EGL program-specific BIND control parts:
If you need to create an EGL program-specific BIND command to use for generation, note that the CSP BIND commands are identical to those required for VAGen, but somewhat different from those required for EGL. Therefore, create the program-specific BIND control part by following these steps:
1. Create one VAGen Bind Control part for each program-specific bind control member of the PDS pointed to by the EZEBIND DD statement in your CSP generation JCL by following these steps:
   a. Download the CSP BIND command member to a workstation file.
   b. Create a VAGen Bind Control part.
      
      Note: For step by step details of how to create a VAGen control part, refer to VisualAge Generator Migration Guide, Chapter 32 “Completing your migration on Smalltalk”, section “Defining control information”.
   • If you only generate the application for one environment, name the VAGen Bind Control part programName.BND, where programName is the same as your CSP application name.
   • If you generate the same application for multiple runtime environments, name the VAGen Bind Control part programName.environment, where environment varies based on the EGL runtime environment for which the BIND command is being created. For example, set environment to ZOSBATCH rather than MVSBATCH.
   c. From the Parts Editor window, click File -> Read from File to retrieve your downloaded CSP bind command file.
   d. Save the VAGen bind control part.
2. Export a VAGen External Source Format file containing all the bind control parts and use the VAGen to EGL Migration Tool Single File Mode to migrate these parts. The migration tool strips the .BND suffix from the bind control part name. The migration tool also converts any periods (.) in the bind control part name to underscores (_).

During generation, EGL searches for a bind control part with the same name as the program. Therefore, if you used a VAGen suffix of .BND, you do not need to do anything in EGL to have it recognize your bind control part. However, if you generate for multiple runtime environments, you must create one build descriptor part for each program and runtime environment combination that requires a special bind control part. In the program’s build descriptor part, set the bind build descriptor option to programName_environment so that EGL finds the correct bind control part for the program when it is generated for that runtime environment.

Link edit commands

In CSP, generation creates the link edit statements for the program based on the type of program, its database access, and the runtime environment. CSP generation includes the link edit statements
directly in the preparation JCL. In some situations such as a static link to a PLI program, you must provide your own link edit commands. In this case, the EZELINK DD statement in the generation job specifies the library (PDS) that contains special link edit commands that you want to use when you prepare your application. If there is a PDS member with the same name as the application you are generating, the CSP preparation job includes an extra step to relink the generated COBOL program based on the link edit commands you provided.

Note that you might have used both techniques in CSP – using the default link edit commands for most programs and providing an application-specific link edit command for a few programs that have special requirements. Only the application-specific link edit commands are in the library specified by the EZELINK DD statement. Note that the EZELINK DD statement generally points to a different library for each runtime environment, so you might need to check for the same application in multiple libraries.

In EGL, special link edit commands are required in the same situations as in CSP. If you used special link edit commands in CSP, you must create program-specific link edit command parts in EGL. Follow the recommendation below for creating the program-specific link edit parts.

**Recommendation for program-specific link edit parts:**

If you need to create an EGL program-specific link edit command to use for generation, note that the CSP link edit commands are identical to those required for VAGen, but somewhat different from those required for EGL. Therefore, follow these steps:

1. Create one VAGen Link Edit part for each link edit member of the PDS pointed to by the EZELINK DD statement in your CSP generation JCL by following these steps:
   a. Download the CSP BIND command member to a workstation file.
   b. Create a VAGen Link Edit part.

   **Note:** For step by step details of how to create a VAGen control part, refer to VisualAge Generator Migration Guide, Chapter 32 “Completing your migration on Smalltalk”, section “Defining control information”.
   - If you only generate the application for one environment, name the VAGen Link Edit part `programName.LKG`, where `programName` is the same as your CSP application name.
   - If you generate the same application for multiple runtime environments, name the VAGen Link Edit part `programName.environment`, where `environment` varies based on the EGL runtime environment for which the link edit command is being created. For example, set `environment` to `ZOSBATCH` rather than `MVSBATCH`.
   c. From the Parts Editor window, click File -> Read from File to retrieve your CSP link edit command file.
   d. Save the VAGen link edit part.

2. Export a VAGen External Source Format file containing all the link edit parts and use the VAGen to EGL Migration Tool Single File Mode to migrate these parts. The migration tool strips the .LKG suffix from the link edit part name. The migration tool also converts any periods (.) in the link edit part name to underscores (_).

During generation, EGL searches for a link edit part with the same name as the program. Therefore, if you used a VAGen suffix of .LKG, you do not need to do anything in EGL to have it recognize your link edit part. However, if you generate for multiple runtime environments, you must create one build descriptor part for each program and runtime environment combination that requires a special link edit part. In the program’s build descriptor part, set the `linkedit` build descriptor option to
programName_environment so that EGL finds the correct link edit part for the program when it is generated for that runtime environment.

Symbolic parameters

In CSP, you can use symbolic parameters in the preparation and runtime JCL templates. You can also use symbolic parameters in your bind and link edit control parts. There are 3 types of symbolic parameters:

- Member-related, which are based on the specific member being generated.
- File-related, which are used to create DD statements for serial, indexed, or relative files used in the application being generated.
- User-defined, which are ones that you create and then specify the value in your generation options.

In EGL, symbolic parameters are also supported for use in runtime JCL templates and in your bind and link edit build parts. EGL uses the same 3 categories of symbolic parameters. However, the names of some symbolic parameters have changed and some symbolic parameters are no longer supported.

In CSP, you set the value of a symbolic parameter using the SYMPARM generation option. In EGL, you set the value using the symbolicParameter build descriptor option. See the SYMPARM generation option information in Table A2 for details.

Table A10 shows the member-related symbolic parameters. Table A11 shows the file-related symbolic parameters. Table A12 shows the predefined user-defined symbolic parameters. All 3 tables include the VisualAge Generator 4.5 column as an aid in finding additional details in the VisualAge Generator to EGL Migration Guide, Appendix B, section “Control Parts”, subsection “Symbolic parameters”.
### Table A10. CSP member-related symbolic parameters

<table>
<thead>
<tr>
<th>Cross System Product 4.1</th>
<th>VisualAge Generator 4.5</th>
<th>EGL</th>
</tr>
</thead>
<tbody>
<tr>
<td>EZEDATA</td>
<td>EZEDATA</td>
<td>DATA</td>
</tr>
<tr>
<td>EZEDLI</td>
<td>EZEDLI</td>
<td>ESEDLI – DL/I only</td>
</tr>
<tr>
<td>EZEENTRY</td>
<td>EZEENTRY</td>
<td>Not supported</td>
</tr>
<tr>
<td>EZEENV</td>
<td>EZEENV</td>
<td>SYSTEM</td>
</tr>
<tr>
<td>EZEGERDATE</td>
<td>EZEGERDATE</td>
<td>EZEGERDATE</td>
</tr>
<tr>
<td>EZEGBR</td>
<td>EZEGBR</td>
<td>EZEGBR</td>
</tr>
<tr>
<td>EZEGETIME</td>
<td>EZEGETIME</td>
<td>EZEGETIME</td>
</tr>
<tr>
<td>EZEJOB</td>
<td>EZEJOB</td>
<td>Not supported</td>
</tr>
<tr>
<td>EZEMBR</td>
<td>EZEMBR</td>
<td>In a JCL script, link edit part, or bind part, EZEALIAS. Otherwise, EZEMBR</td>
</tr>
<tr>
<td>EZEMSG</td>
<td>EZEMSG</td>
<td>Not supported</td>
</tr>
<tr>
<td>EZEPLS</td>
<td>EZEPLS</td>
<td>EZEPLS</td>
</tr>
<tr>
<td>EZEPSB</td>
<td>EZEPSB</td>
<td>Not supported – DL/I and IMS only</td>
</tr>
<tr>
<td>EZESQL</td>
<td>EZESQL</td>
<td>EZESQL</td>
</tr>
<tr>
<td>EZETRAN</td>
<td>EZETRAN</td>
<td>Not supported</td>
</tr>
<tr>
<td>EZETRO</td>
<td>EZETRO</td>
<td>Not supported</td>
</tr>
<tr>
<td>EZEUSR</td>
<td>EZEUSR</td>
<td>EZEUSR</td>
</tr>
<tr>
<td>EZEUSRID</td>
<td>EZEUSRID</td>
<td>EZEUSRID</td>
</tr>
<tr>
<td>EZEXAPP</td>
<td>EZEXAPP</td>
<td>Not supported</td>
</tr>
</tbody>
</table>

### Table A11. CSP file-related symbolic parameters

<table>
<thead>
<tr>
<th>Cross System Product 4.1</th>
<th>VisualAge Generator 4.5</th>
<th>EGL</th>
</tr>
</thead>
<tbody>
<tr>
<td>EZEBLK</td>
<td>EZEBLK</td>
<td>EZEBLK</td>
</tr>
<tr>
<td>EZEDBD</td>
<td>EZEDBD</td>
<td>Not supported</td>
</tr>
<tr>
<td>EZEEDD</td>
<td>EZEEDD</td>
<td>EZEEDD</td>
</tr>
<tr>
<td>EZEDSN</td>
<td>EZEDSN</td>
<td>EZEDSN</td>
</tr>
<tr>
<td>EZELRECL</td>
<td>EZELRECL</td>
<td>EZELRECL</td>
</tr>
<tr>
<td>EZEREFCFM</td>
<td>EZEREFCFM</td>
<td>EZEREFCFM</td>
</tr>
</tbody>
</table>

### Table A12. CSP predefined user-defined symbolic parameters

<table>
<thead>
<tr>
<th>Cross System Product 4.1</th>
<th>VisualAge Generator 4.5</th>
<th>EGL</th>
</tr>
</thead>
<tbody>
<tr>
<td>EKZAUTH</td>
<td>EKZAUTH</td>
<td>EKZAUTH</td>
</tr>
<tr>
<td>EKZINST</td>
<td>EKZINST</td>
<td>EKZINST</td>
</tr>
</tbody>
</table>
Appendix B. Converting CSP preparation templates and procedures

In CSP, preparation templates are used to generate the JCL necessary to do the DB2 precompile, CICS translate, COBOL compile, link edit, and bind. The template that is used depends on the type of member that is generated, the runtime environment, and, for application members, the type of database access. The preparation templates invoke JCL procedures to provide the actual steps in the preparation process. The EZEJTMP DD statement in the generation job specifies the library (PDS) that contains the templates for your preparation JCL.

In EGL, the preparation process is handled by an EGL build server. The EGL build scripts merge the information previously contained in the CSP preparation template and preparation procedures.

**Recommendation:** Review your CSP preparation templates and procedures to determine whether you customized them. If so, the best way to modify and debug an EGL build script is as follows:

- Specify `prep="NO"` in your build descriptor options.
- 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.
- Manually upload the outputs of generation to the data sets you plan to use on the z/OS host.
- Create preparation JCL for the program. You can use your CSP 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 generated by EGL to determine if there are additional steps, libraries, and so on required for EGL.
- Test the preparation JCL until you are satisfied that your modifications are correct.
- Convert the preparation JCL to pseudo-JCL so it can be used as a build script. For details, refer to the *EGL Server Guide*.
- Specify `prep="YES"` in your build descriptor options.
- Generate the program again for your runtime environment. The outputs of generation should now be uploaded and prepared automatically using the build server.
- 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.

Table B1 shows the CSP preparation templates and procedures used for preparing the outputs of COBOL generation. The template name is shown first, followed by the procedure name. For most of the procedure names, the first 3 characters are ELA and the remaining characters indicate the steps included in the procedure as follows: P (DB2 precompile), T (CICS translate), C (COBOL compile), L (link edit), and B (DB2 bind).

**Note:** CSP also uses BIND control templates. See “Bind commands” on page 78 for details on converting these templates.
<table>
<thead>
<tr>
<th>Environment and Details</th>
<th>Cross System Product 4.1 Template and Procedure</th>
<th>VisualAge Generator 4.5 Template and Procedure</th>
<th>EGL Build Script</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MVS CICS</strong> Application</td>
<td>EZECCCL ELATCL</td>
<td>EFK2MPCB ELATCL</td>
<td>FDATCL</td>
</tr>
<tr>
<td>No DB2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>With DB2</td>
<td>EZECPCLB ELAPCLB</td>
<td>EFK2MPCA ELAPCLB</td>
<td>FADAPTCL followed by FDABIND</td>
</tr>
<tr>
<td><strong>Map Group</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>EZEACL ELACL</td>
<td>EFK2MMCA ELACL</td>
<td>FDAACL</td>
</tr>
<tr>
<td></td>
<td>EZECLMGL ELAL</td>
<td>EFK2MMTF ELAL</td>
<td>FDALINK</td>
</tr>
<tr>
<td><strong>MVS Batch</strong> Application</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No DB2</td>
<td>EZECCCL ELACL</td>
<td>EFK2MPBA ELACL</td>
<td>FDABCL</td>
</tr>
<tr>
<td>DL/I and DB2</td>
<td>EZEDPCLB ELAPCLB</td>
<td>EFK2MPBB ELAPCLB</td>
<td>FADAPCL followed by FDABIND</td>
</tr>
<tr>
<td>DB2 only</td>
<td>EZEEMCCL ELAPCLB</td>
<td>EFK2MPBC ELAPCLB</td>
<td>FADAPCL followed by FDABIND</td>
</tr>
<tr>
<td><strong>Map Group</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>EZEACL ELACL</td>
<td>EFK2MMCA ELACL</td>
<td>FDAACL</td>
</tr>
<tr>
<td><strong>IMS/VS</strong> Application</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DL/I only</td>
<td>EZEICL ELACL</td>
<td>EFK2MPIA ELACL</td>
<td>FDAACL</td>
</tr>
<tr>
<td>DL/I with DB2 work database</td>
<td>EZEICLB ELACL</td>
<td>EFK2MPIB ELACL</td>
<td>FADAPCL followed by FDABIND</td>
</tr>
<tr>
<td>DL/I and DB2</td>
<td>EZEIPCLB ELAPCLB</td>
<td>EFK2MPIE ELAPCLB</td>
<td>FADAPCL followed by FDABIND</td>
</tr>
<tr>
<td><strong>Map Group</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>EZEMFSCCL ELACL</td>
<td>EFK2MMCB ELACL</td>
<td>FDAACL</td>
</tr>
<tr>
<td></td>
<td>EZEMFST MFSTEST</td>
<td>EFK2MMST MFSTEST</td>
<td>FDAMFS</td>
</tr>
<tr>
<td></td>
<td>EZEMFSU MFSUTL</td>
<td>EFK2MMSU MFSUTL</td>
<td>FDAMFS</td>
</tr>
<tr>
<td><strong>IMS BMP</strong> Application</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DL/I only</td>
<td>EZEBCL ELACL</td>
<td>EFK2MPIA ELACL</td>
<td>FDABCL</td>
</tr>
<tr>
<td>DL/I and DB2</td>
<td>EZEBPCLB ELAPCLB</td>
<td>EFK2MPIB ELAPCLB</td>
<td>FADAPCL followed by FDABIND</td>
</tr>
<tr>
<td><strong>Map Group</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>EZEMFSCCL ELACL</td>
<td>EFK2MMCB ELACL</td>
<td>FDAACL</td>
</tr>
<tr>
<td></td>
<td>EZEMFST MFSTEST</td>
<td>EFK2MMST MFSTEST</td>
<td>FDAMFS</td>
</tr>
<tr>
<td>Environment and Details</td>
<td>Cross System Product 4.1 Template and Procedure</td>
<td>VisualAge Generator 4.5 Template and Procedure</td>
<td>EGL Build Script</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-------------------------------------------------</td>
<td>-----------------------------------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td></td>
<td>EZEMFSU  MFSUTL</td>
<td>EFK2MMSU  MFSUTL</td>
<td>FDAMFS</td>
</tr>
<tr>
<td></td>
<td>EZEACL  ELACL</td>
<td>EFK2MMCA  ELACL</td>
<td>FDACL</td>
</tr>
<tr>
<td>MVS TSO Application</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No DB2</td>
<td>EZETCL  ELACL</td>
<td>EFK2MPTA  ELACL</td>
<td>Not supported</td>
</tr>
<tr>
<td>DB2 without DL/I</td>
<td>EZETPCLB  ELAPCLB</td>
<td>EFK2MPTB  ELAPCLB</td>
<td>Not supported</td>
</tr>
<tr>
<td>Map Group</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>EZEACL  ELACL</td>
<td>EFK2MMCA  ELACL</td>
<td>Not supported</td>
</tr>
<tr>
<td></td>
<td>EZECMGL  ELAL</td>
<td>EFK2MMTF  ELAL</td>
<td>Not supported</td>
</tr>
<tr>
<td>All Application Relink</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>EZELINKR  ELARLINK</td>
<td>EFK2MPRE  ELARLINK</td>
<td>FDALINK</td>
</tr>
<tr>
<td>Table</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>EZEACL  ELACL</td>
<td>EFK2MMCA  ELACL</td>
<td>FDACL</td>
</tr>
</tbody>
</table>
Appendix C. Converting CSP runtime templates

In CSP, runtime templates are used to generate sample runtime JCL for the MVS Batch and IMS BMP runtime environments or a sample CLIST for the MVS TSO runtime environment. There are 3 types of runtime templates:

- Execution JCL and CLIST templates that create the main portion of the sample runtime JCL or CLIST.
- 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 application 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.

The EZEJTMP DD statement in the generation job specifies the library (PDS) that contains the templates for your preparation JCL.

In EGL, there are similar runtime templates. The templateDir build descriptor option points to the directory where the templates reside. The default directory is the following directory:

`SharedInstallDirectory\plugins\com.ibm.etools.egl.generators.cobol_version\MVStemplates`

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

Recommendation:

- If you need to tailor the runtime JCL templates, consider the following points:
  - Create your own directory outside the shared 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.
  - Use your CSP runtime templates as a starting point and compare each template to the corresponding EGL template to determine what tailoring you might require. Be sure to change the CSP symbolic parameters to the corresponding EGL symbolic parameters as shown in Tables A10 through A12.

Table C1 shows the runtime CSP templates that are used to generate the basic runtime JCL. Table C2 shows the file and database allocation templates that are used to create DD statements within the generated runtime JCL. Table C3 shows the file and database allocation placeholder templates that are used to indicate 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. All 3 tables include the corresponding VAGen and EGL information.
### Table C1. CSP runtime templates

<table>
<thead>
<tr>
<th>Environment and Details</th>
<th>Cross System Product 4.1 Runtime JCL Template</th>
<th>VisualAge Generator 4.5 Runtime JCL Template</th>
<th>EGL</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MVS Batch</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Called application</td>
<td>EZECALL</td>
<td>EFK2MEBA</td>
<td>fda2meba.tpl</td>
</tr>
<tr>
<td>No databases</td>
<td>EZEMBACH</td>
<td>EFK2MEBE</td>
<td>fda2mebc.tpl</td>
</tr>
<tr>
<td>DL/I only</td>
<td>EZEMDLI</td>
<td>EFK2MEBC</td>
<td>fda2mebc.tpl</td>
</tr>
<tr>
<td>DB2 only</td>
<td>EZEMDB2</td>
<td>EFK2MEBD</td>
<td>fda2mebd.tpl</td>
</tr>
<tr>
<td>DL/I and DB2</td>
<td>EZEMDBS</td>
<td>EFK2MEBB</td>
<td>fda2mebb.tpl</td>
</tr>
<tr>
<td><strong>IMS BMP</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Called application</td>
<td>EZECALL</td>
<td>EFK2MEBA</td>
<td>fda2meba.tpl</td>
</tr>
<tr>
<td>DL/I only</td>
<td>EZEBMP</td>
<td>EFK2MEIB</td>
<td>fda2meib.tpl</td>
</tr>
<tr>
<td>DL/I and DB2</td>
<td>EZEBMP2</td>
<td>EFK2MEIA</td>
<td>fda2meia.tpl</td>
</tr>
<tr>
<td><strong>MVS TSO CLI$$</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Called application</td>
<td>EZETCALL</td>
<td>EFK2META</td>
<td>Not supported</td>
</tr>
<tr>
<td>No databases</td>
<td>EZETSO</td>
<td>EFK2METD</td>
<td>Not supported</td>
</tr>
<tr>
<td>DL/I only</td>
<td>EZETDLI</td>
<td>EFK2METB</td>
<td>Not supported</td>
</tr>
<tr>
<td>DB2 only</td>
<td>EZEDB2</td>
<td>EFK2METC</td>
<td>Not supported</td>
</tr>
</tbody>
</table>

### Table C2. CSP file and database allocation templates

<table>
<thead>
<tr>
<th>Environment and Details</th>
<th>Cross System Product 4.1 Runtime JCL Template</th>
<th>VisualAge Generator 4.5 Runtime JCL Template</th>
<th>EGL</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MVS Batch and IMS BMP</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DL/I database in MVS Batch</td>
<td>EZEMDBDD</td>
<td>EFK2MDLI</td>
<td>fda2mdli.tpl</td>
</tr>
<tr>
<td>VSAM or VSAMRS input for serial, indexed, or relative files</td>
<td>EZEVSDDI</td>
<td>EFK2MVSI</td>
<td>fda2mvsi.tpl</td>
</tr>
<tr>
<td>VSAM or VSAMRS output for serial, indexed, or relative files</td>
<td>EZEVSDDO</td>
<td>EFK2MVSO</td>
<td>fda2mvso.tpl</td>
</tr>
<tr>
<td>SEQ or SEQRS input for serial files</td>
<td>EZEQSDDI</td>
<td>EFK2MSDI</td>
<td>fda2msdi.tpl</td>
</tr>
<tr>
<td>SEQ or SEQRS output for serial files</td>
<td>EZEQSDDO</td>
<td>EFK2MSDO</td>
<td>fda2msdo.tpl</td>
</tr>
<tr>
<td>GSAM input for serial files</td>
<td>EZEGRSDI</td>
<td>EFK2MGSI</td>
<td>fda2mgsi.tpl</td>
</tr>
<tr>
<td>GSAM output for serial files</td>
<td>EZEGRSDDO</td>
<td>EFK2MGSO</td>
<td>fda2mgso.tpl</td>
</tr>
</tbody>
</table>
### GSAM file in an IMS BMP

<table>
<thead>
<tr>
<th>Environment and Details</th>
<th>Cross System Product 4.1 Runtime JCL Template</th>
<th>VisualAge Generator 4.5 Runtime JCL Template</th>
<th>EGL</th>
</tr>
</thead>
<tbody>
<tr>
<td>MVS Batch and IMS BMP</td>
<td>EZEMDBAL</td>
<td>EFK2MTDL</td>
<td>Not supported</td>
</tr>
<tr>
<td>XFER or DXFR to EZEAPP</td>
<td>EZEMXEAP</td>
<td>EFK2MEZA</td>
<td>fda2meza.tpl</td>
</tr>
<tr>
<td>CALL, XFER or DXFR to a specific application or RT generation option transfers to a specific application</td>
<td>EZEMXAPP</td>
<td>EFK2MCAL</td>
<td>fda2mcal.tpl</td>
</tr>
<tr>
<td>Application uses EZEDEST or EZEDESTP</td>
<td>EZEMDEST</td>
<td>EFK2MEZD</td>
<td>fda2mezd.tpl</td>
</tr>
</tbody>
</table>

### MVS TSO CLIST Templates

<table>
<thead>
<tr>
<th>Environment and Details</th>
<th>Cross System Product 4.1 Runtime JCL Template</th>
<th>VisualAge Generator 4.5 Runtime JCL Template</th>
<th>EGL</th>
</tr>
</thead>
<tbody>
<tr>
<td>XFER or DXFR to EZEAPP</td>
<td>EZETXEAP</td>
<td>EFK2MTEA</td>
<td>Not supported</td>
</tr>
<tr>
<td>CALL, XFER or DXFR to a specific application or RT generation option transfers to a specific application</td>
<td>EZETXAPP</td>
<td>EFK2MTCL</td>
<td>Not supported</td>
</tr>
<tr>
<td>Application uses EZEDEST or EZEDESTP</td>
<td>EZETDEST</td>
<td>EFK2MTED</td>
<td>Not supported</td>
</tr>
</tbody>
</table>

Table C3. CSP file and database allocation placeholder templates

---

89
Appendix D. Converting CSP reserved words file

In CSP, the EZEWORD DD statement in the generation job specifies the library (PDS) and the member that contains reserved words that are not permitted in the generated COBOL programs. The default EZEWORDS member is shipped with CSP and contains COBOL, SQL, and CICS reserved words that cannot be used in the generated COBOL programs. You might have added additional words to this list. Compare the PDS member pointed to by the EZEWORD DD statement with CSPHighLevelQualifier.SEZESAMP(EZEWORDS) to determine if you made any modifications. CSPHighLevelQualifier is the high level qualifier used when you installed CSP/370AD. If you never made modifications to the EZEWORDS member, you can skip this section.

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

Recommendation:
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 words you added to EZEWORDS. 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’s workstation. This technique simplifies making a change to the reserved words file.
Appendix E. Code page conversions

Depending on whether you migrate from CSP/370AD or CSP/2AD, you might need to use one or both of these techniques.

Using the hptrules.nls file

You can use the following technique when migrating from either CSP/370AD or CSP/2AD if the not sign is the only special character that does not download correctly from the host or move correctly from OS/2 to a Windows operating system.

For CSP, the not sign is the ¬. When you download CSP/370AD External Source Format files, some download programs convert ¬ to ¼ or other characters. Similarly, when you transfer CSP/2AD External Source Format files from OS/2 to the Windows operating system, the code point for ¬ differs, causing the not sign to convert incorrectly. VAGen provides special conversion for the not sign using a file called hptrules.nls.

For VAGen on Smalltalk, the hptrules.nls file is in the following directory:

\VASmalltalkInstallationDirectory\nls

For VAGen on Java, the hptrules.nls file is in the following directory:

\VAJavaInstallationDirectory\ide\program\hptvgj45\nls

The hptrules.nls file enables you to specify national language information and can help with handling the not sign. One section of the file enables you to specify your three national language characters, the ^ for the standard VAGen not sign, and an alternate not sign. If you determine that the only special character you need to convert is the ¬ or ¼, you can follow these steps to do the conversion:

1. Shut down VisualAge Generator.
2. Edit hptrules.nls.
3. Read the comments in the file. You need to change the first section of the file (:nlsrules).
4. In the :nlsrules section, there are three columns: the locale, 5 special characters, and the default language code. Locate the line for your locale. The five special characters are:

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-3</td>
<td>The three national language characters (for example, $#@ for English-US)</td>
</tr>
<tr>
<td>4</td>
<td>The ^, which is the standard not sign for VisualAge Generator</td>
</tr>
<tr>
<td>5</td>
<td>An alternate not sign, which you can set to ¬, ¼, or whatever character your download or transfer converted the CSP not sign to. Using the same text editor for both files, copy the incorrectly converted not sign from the trial file you downloaded or transferred. Paste the character into this position of the hptrules.nls file for your language code.</td>
</tr>
</tbody>
</table>

5. Start VisualAge Generator.

Whenever you import an External Source Format file into VisualAge Generator (through the normal VAGen Import command or with the ESF to MSL function of the VAGen MSL Migration Assistance Tool), VisualAge Generator converts any occurrence of the alternate not sign specified in hptrules.nls to the ^ . This conversion only applies to occurrences in processes or statement groups. The constant delimiter for a map is not converted, but because the constant delimiter is stored with each map, conversion is not necessary.

Note: When you convert to EGL, the VAGen not sign is converted again from ^ to !.
Changing from OS/2 to a Windows Operating System

You can use the following technique when migrating from CSP/2AD for single-byte languages. If you are migrating a double-byte language (for example, Chinese, Japanese, or Korean), code page conversion is not required and you can skip this section.

Do not use this technique if you transferred your External Source Format file from OS/2 to the Windows operating system in such a way that the code page conversion was automatically performed. For example, if you transferred the file using FTP with the ASCII option, then the code page conversion should be done automatically. You still need to check that the special characters were converted correctly and then use this technique if there is a problem.

The code pages from some languages differ between OS/2 and Windows operating systems. For example, the code point for the certain characters such as the not sign differs in the English-US code pages for OS/2 and Windows operating systems. VAGen provides a code page conversion tool called hptcnv45.exe.

For VAGen on Smalltalk, the hptcnv45.exe file is in the following directory:

`VASmalltalkInstallationDirectory\bin`

For VAGen on Java, the hptcnv45.exe file is in the following directory:

`VAJavaInstallationDirectory\ide\program`

To run the code page conversion tool, follow these steps:
1. Make the External Source Format file available on the Windows operating system.
2. From a Command Prompt window, change to the directory in which hptcnv45.exe is located.
3. Convert the External Source Format file by running the following command:
   ```
   hptcnv45 esf-file-name conversion-table
   ```
   `esf-file-name` is the drive, path, and file name for the External Source Format file you want to convert. `conversion-table` is the name of a conversion table that translates from the OS/2 code page to the Windows operating system code page. Table E1 shows the conversion table names.

<table>
<thead>
<tr>
<th>Language</th>
<th>From OS/2 To Windows Operating System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arabic</td>
<td>ELAO2ARA</td>
</tr>
<tr>
<td>Brazilian Portuguese</td>
<td>ELAO2850</td>
</tr>
<tr>
<td>Danish</td>
<td>ELAO2850</td>
</tr>
<tr>
<td>Eastern European, Latin-2</td>
<td>ELAO2852</td>
</tr>
<tr>
<td>English, USA</td>
<td>ELAO2437</td>
</tr>
<tr>
<td>English, United Kingdom</td>
<td>ELAO2850</td>
</tr>
<tr>
<td>Finnish</td>
<td>ELAO2850</td>
</tr>
<tr>
<td>French</td>
<td>ELAO2850</td>
</tr>
<tr>
<td>German</td>
<td>ELAO2850</td>
</tr>
<tr>
<td>Greek</td>
<td>ELAO2GRE</td>
</tr>
<tr>
<td>Hebrew</td>
<td>ELAO2HEB</td>
</tr>
<tr>
<td>Italian</td>
<td>ELAO2850</td>
</tr>
<tr>
<td>Language</td>
<td>From OS/2 To Windows Operating System</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------------------------------</td>
</tr>
<tr>
<td>Norwegian</td>
<td>ELAO2850</td>
</tr>
<tr>
<td>Russian</td>
<td>ELAO2CYR</td>
</tr>
<tr>
<td>Spanish</td>
<td>ELAO2850</td>
</tr>
<tr>
<td>Swedish</td>
<td>ELAO2850</td>
</tr>
<tr>
<td>Swiss German</td>
<td>ELAO2850</td>
</tr>
<tr>
<td>Turkish</td>
<td>ELAO2TUR</td>
</tr>
</tbody>
</table>

The converted External Source Format file is named `esf-file-name.cnv`. The file is stored in the directory where `hptcnv45.exe` is located.

To see a short (English) description of the conversion tool, type the following command:

```
hptcnv45 ?
```

4. Use a comparison tool to do a byte-by-byte comparison of the External Source Format file and the converted file.

If the only character being converted is the not sign, then you might be able to use the technique described in the section “Using the hptrules.nls file” to avoid converting all your files. However, use caution if you skip the conversion step – some of your other files might have characters other than the not sign that require conversion.

Be especially alert for any characters being converted that should not be. For example, you might have coded:

```
MOVEA "special-character" TO HIGH-VALUES-CHAR;
```

as a way of setting the high end for a range of keys that you are searching in a file or database. In this case, you might not want the special-character to be converted or you might not like the hex value that results from the conversion. If this is the case, you must to modify your code. A better coding technique is:

```
MOVEA "special-hex-characters" TO HIGH-VALUES-HEX;
```

The field called `HIGH-VALUES-HEX` is substructured under the field called `HIGH-VALUES-CHAR`. This technique ensures that the same special-hex-characters are used in the External Source Format file and the corresponding converted file.

5. Use the converted External Source Format file as input to the VAGen MSL Migration Assistance Tool or VAGen Import.
Appendix F. Installing EMSRV

When you install VisualAge Smalltalk, you must install EMSRV as the library manager. If you use Windows XP, you must install the latest version of EMSRV. The version that you need is available on the following Web site:

ftp://ps.boulder.ibm.com/ps/products/visualagegen/VAJava/EMSRV71a/

Download both the .exe file and the .pdf file.

Note: You can also obtain the latest VAGen Version 4.5 Fix Pack from the following Web site:

To install EMSRV, you must be logged on as an Administrator. The following steps assume that you are using Windows XP:

1. Define a new user by clicking Start -> Control Panel -> User Accounts and then following these steps:
   a. Select Create a new account.
   b. Set the user name (for example, emsrv). Click Next.
   c. Select Computer administrator. Click Create account.
   d. Select the new account and change the password (for example, emsrvpw0)
   e. From the Control Panel window, select Administrative Tools -> Local Security Policy.
   f. Expand Local Policies and then select User Rights Assignment.
   g. In the right pane, right-click Act as part of the operating system; then click on Properties in the pop-up menu.
      - Add Administrators and emsrv to the list of users that have this user right.
   h. In the right pane, right-click Log on as a service; then click on Properties in the pop-up menu.
      - Add Administrators and emsrv to the list of users that have this user right.
   i. Close the Control Panel windows.

2. Install EMSRV as a service according to the directions in the EMSRV Installation and User's Guide (the PDF you downloaded from the EMSRV site).
Appendix G. Using this paper for other releases of CSP or VAGen

The information in this paper is specific to CSP Version 4.1. However, much of the information does apply to earlier releases of CSP and to VisualGen Version 2.2. For example, the following information applies to migrating from releases other than CSP Version 4.1:

- Organizing your source code.
- Running the migration tools and the manual steps between the tools.
- Setting up your EGL development environment.
- Creating the EGL resource association part.

The main differences are as follows:

- If migrating from CSP Version 3.2.1 or earlier, the following extra tasks are required:
  - Extra steps to prepare your source code for migration.
  - Extra steps as described for CSP Version 3.2.2 and CSP Version 3.3.
- If migrating from CSP Version 3.2.2 or CSP Version 3.3, the following extra tasks are required:
  - Creating your build descriptor, linkage table, bind, and link edit parts.
  - Setting up your build scripts.
  - Setting up your runtime JCL templates.
  - Setting up the EGL reserved words file.
  - Converting your CSP message file.
  - Additional runtime differences for interpretive code.
- If migrating from VisualAge Version 2.2, the following extra tasks are required:
  - Dealing with the migration of GUI parts, if any.
  - The values of some generation options differ.

Additional details are provided below.

CSP Version 3.2.1 or earlier

Refer to *Migrating Cross System Product Applications to VisualAge Generator*, Part 2 “Migrating from Cross System Product 3.2.1 or Earlier” for additional details of steps that might be required. The book is available at:


CSP Versions 3.2.1 and earlier do not provide the ability to export External Source Format files. You must perform the following additional steps to prepare your source code for migration:

1. Export internal format for your current release of CSP.
2. Import the internal format into CSP Version 3.2.2, Version 3.3, or Version 4.1.
3. Export the External Source Format, download it, and import it using the VAGen MSL Migration Assistance Tool as described in this paper.

In addition, CSP Version 3.2.1 and earlier were always interpretive runtime environments. Therefore, you must do all the additional steps described for CSP interpretive environments under “CSP Version 3.2.2 or Version 3.3”.

95
CSP Version 3.2.2 or Version 3.3

Refer to *Migrating Cross System Product Applications to VisualAge Generator*, Part 3 “Migrating from Cross System Product 3.2.2 or 3.3” for additional details of steps that might be required. The book is available at:


CSP Version 3.2.2 and Version 3.3 provide External Source Format. You can export the External Source Format, download it, and import it using the VAGen MSL Migration Assistance Tool as described in this paper.

Table G1 shows the runtime environments supported by CSP Version 3.3 and the corresponding EGL runtime environment, if any. Earlier versions of CSP supported a subset of the CSP Version 3.3 runtime environments, with the specific environments varying by version.

Table G1. CSP Version 3.3 runtime environments

<table>
<thead>
<tr>
<th>CSP Version 3.3 runtime environment</th>
<th>Interpretive or COBOL generation</th>
<th>EGL runtime environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>DPPX</td>
<td>Interpretive</td>
<td>Not supported</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Consider COBOL generation for iSeries</td>
</tr>
<tr>
<td>IBM DOS</td>
<td>Interpretive</td>
<td>Not supported</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Consider Java generation for a Windows operating system</td>
</tr>
<tr>
<td>IMS BMP</td>
<td>COBOL generation</td>
<td>COBOL generation for IMS BMP (1)</td>
</tr>
<tr>
<td>IMS/VS</td>
<td>COBOL generation</td>
<td>COBOL generation for IMS/VS (1)(2)</td>
</tr>
<tr>
<td>MVS BATCH</td>
<td>Interpretive or COBOL generation</td>
<td>COBOL generation for ZOSBATCH (1)</td>
</tr>
<tr>
<td>MVS CICS</td>
<td>Interpretive</td>
<td>COBOL generation for ZOSCICS (2)</td>
</tr>
<tr>
<td>MVS TSO</td>
<td>Interpretive</td>
<td>Not supported</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Consider COBOL generation for ZOSCICS</td>
</tr>
<tr>
<td>OS/2</td>
<td>Interpretive</td>
<td>Not supported</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Consider Java generation for a Windows operating system</td>
</tr>
<tr>
<td>OS/400®</td>
<td>Interpretive</td>
<td>COBOL generation for iSeries (2)(3)</td>
</tr>
<tr>
<td>VM BATCH</td>
<td>Interpretive</td>
<td>Not supported</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Consider COBOL generation for ZOSBATCH</td>
</tr>
<tr>
<td>VM CMS</td>
<td>Interpretive</td>
<td>Not supported</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Consider COBOL generation for ZOSCICS</td>
</tr>
<tr>
<td>VSE BATCH</td>
<td>Interpretive</td>
<td>Not supported</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(4)(5)</td>
</tr>
<tr>
<td>VSE CICS</td>
<td>Interpretive</td>
<td>Not supported</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(4)(5)</td>
</tr>
</tbody>
</table>

Notes:

1. COBOL generation for CSP Version 3.3 is provided by CSP/370RS Version 1.1.
2. It is possible to call programs in this environment from the EGL debugger.
3. COBOL generation for iSeries is supported by Rational Business Developer Version 7.1. Rational Business Developer Version 7.1 also includes the build server and runtime support for iSeries, but you must upload them from Rational Business Developer to your iSeries host.
4. COBOL generation for VSE was previously supported for Rational Application Developer Version 6 using the VisualAge Generator EGL Plug-in for VSE Version 1.0 (program number 5724-L93). However, the plug-in does not support COBOL generation for VSE using Rational
Business Developer Version 7.1. Consider migrating to COBOL generation for z/OS Batch and z/OS CICS.

5. Refer to Migrating Cross System Product Applications to VisualAge Generator, Chapter 4 “CSP/AD 3.2.2 or CSP/AD 3.3 to VisualAge Generator Developer”, section “Considerations for Migrating an MSL from VSE/ESA to the Workstation” for information about exporting your VSE MSL.

For both the interpretive runtime environments and the COBOL runtime environments in CSP Version 3.3 or earlier, the following tasks differ from what is described in this paper for CSP Version 4.1:

- You must convert the CSP message file to a VAGen table and then to an EGL DataTable. Refer to Migrating Cross System Product Applications to VisualAge Generator, Appendix F “Using the Message File Conversion Utility”, subsection “Using VisualAge Generator Message File Conversion Utility” for details of converting to a VAGen message table. Then include the VAGen message table with your other parts when you migrate from VisualAge Generator to EGL.

  Note:
  - The message file conversion utility is an EGL program named DZGMSGC.
  - For z/OS, the utility ships with the EGL server product in the PDS with the low-level qualifier SELALMD. Refer to the EGL Server Guide for sample z/OS runtime JCL.
  - For VSE, the utility ships with VisualAge Generator Server for MVS, VSE, and VM Version 1.2. The utility should be in the VAGen product runtime library. Refer to the VisualAge Generator Server Guide for MVS, VSE, and VM for sample VSE runtime JCL.
  - There is no CSP linkage table. Instead, only a default linkage is supported. You should not need a linkage table for generation. However, you might need a debug linkage table in the same situations as described in this paper for CSP Version 4.1. If you must create a linkage options part for generation in EGL, the comparable information in EGL is:
    - For CICS, use localCall, linkType="CICSLINK", parmForm="COMMPTR”.
    - For non-CICS, use localCall, linkType="DYNAMIC", parmForm="OSLINK”.
  - EZECNVCM and EZEDLTRM are merged into converseVar.commitOnConverse in EGL.

- For additional language changes, refer to Migrating Cross System Product Applications to VisualAge Generator, Appendix A “Language Differences from the Cross System Product Set”. Also see “Automatic conversions during migration to VisualAge Generator” on page 33. Knowledge of the language differences might be useful if you need to refer to the VisualAge Generator to EGL Migration Guide, Appendix B “Relationship of VisualAge Generator and EGL Language Elements”.

For the interpretive runtime environments in CSP Version 3.3 and earlier, the following tasks differ from what is described in this paper for CSP Version 4.1:

- If your runtime environment is CICS, ensure that the EGL segmented property is set correctly based on the CSP execution mode. In CSP Version 3.3 or earlier, the execution mode for main transaction applications is specified at generation time. The execution mode is always NONSEGMENTED for application types other than main transactions. In EGL, the segmented property is specified on the program part at definition time and only applies to EGL main text programs (CSP main transaction applications).
- If you saved CSP generation options, the execution mode is stored in the CSP application member and appears in the External Source Format for the application on the :genopts tag in the EXECMODE attribute. Set the segmented property based on the EXECMODE attribute value in the following way:
  - SEGMENTED and EITHER convert to EGL segmented=yes.
• NONSEGMENTED and SINGLESEG convert to EGL segmented=no.
  The migration tool makes this change for you. However, you should review the values for
  the segmented property very carefully because the information stored in the CSP application
  member reflects the last time you specified to save the information – which is not necessarily
  the information you used the last time you generated the application.
• If you did not save CSP generation options, the VAGen MSL Migration Assistance Tool
  defaults the execution mode to NONSEGMENTED. The VAGen to EGL Migration Tool
  converts this to segmented=no. You should carefully review the value for the segmented
  property for each of the EGL main text programs.

Note:
  The VAGen to EGL Migration Tool only includes the segmented property for EGL main
  text programs. Therefore, you can use a File Search for this property to find all programs
  that are potentially affected by this problem.

• The generation options for interpretive are not related to the COBOL build descriptor options. To
  create your EGL build descriptor parts, refer to the EGL Generation Guide or online helps and
  review each build descriptor option to determine whether you should set it or not.
• The bind and link edit parts are not generated for interpretive runtime environments. You might
  need to create these parts based on whether the default bind and link edit commands created by
  EGL are acceptable or not.
• Preparation JCL procedures and preparation JCL templates are not used when generating for
  interpretive runtime environments. Therefore, refer to the EGL Server Guide and EGL
  Generation Guide for assistance in customizing the EGL build scripts.
• Runtime JCL templates are not used when generating for interpretive runtime environments.
  Therefore, refer to the EGL Server Guide and EGL Generation Guide for assistance in
  customizing the runtime JCL templates.
• Runtime JCL is quite different between interpretive processing and generated COBOL. You
  might want to generate new sample runtime JCL to use as a starting point. To generate the
  sample runtime JCL, specify the build descriptor option genRunFile="YES" when you generate
  your programs for the ZOSBATCH runtime environment.
• There is no reserved word file. You should not need to create one for EGL.
• You must generate all your programs, FormGroups, and DataTables in EGL. The EGL runtime
  server does not support interpretive execution.
• There are significant runtime differences between interpretive execution and generated COBOL.
  Refer to Migrating Cross System Product Applications to VisualAge Generator, Appendix B
  “Cross System Product Interpretive to COBOL” for details of the runtime differences.

For the COBOL runtime environments in CSP Version 3.3, the following tasks differ from what is
described in this paper for CSP Version 4.1:
• The chaining of generation options parts is different. Refer to Migrating Cross System Product
  Applications to VisualAge Generator, Chapter 7 “CSP/370RS COBOL Generation to VisualAge
  Generator Developer”, section “Generation Options” for a description of how the chaining
  worked for COBOL generation in CSP/370RS.
• The preparation JCL procedure names are the same as in CSP Version 4.1. However, the actual
  JCL in the procedures differs.
• The preparation JCL template names and runtime JCL template names changed between CSP
  Version 3.3 and VisualAge Generator. Refer to Migrating Cross System Product Applications to
  VisualAge Generator, Chapter 7 “CSP/370RS COBOL Generation to VisualAge Generator
  Developer”, section “Templates” for the correspondence between the CSP Version 3.3 and the
  VAGen template names.
The data sets that contain the outputs of generation and preparation use different data set naming conventions in CSP Version 3.3. The runtime environment is not included in the data set name in CSP Version 3.3.

If you previously used a DB2 message database, you should update any program-specific BIND commands to remove the message database. If you previously used a DL/I message database, you should update any PSBs to remove the message database PCB. Be sure to update both the IMS and the EGL PSB definitions.

You must regenerate all your programs, FormGroups, and DataTables with EGL. You cannot call or DXFR (EGL transfer to program statement) between programs generated with CSP/370RS Version 1.1 and programs generated with EGL. You can only XFER (EGL transfer to transaction statement) between programs generated with CSP/370RS Version 1.1 and programs generated with EGL if the programs and their associated transaction codes are running in different IMS regions. The programs generated with CSP/370RS Version 1.1 must run in a region that contains CSP/370RS Version 1.1 in the load library concatenation. The programs generated with EGL must run in a region contains the EGL server in the load library concatenation.

VisualGen Version 2.2

Refer to VisualAge Generator Migration Guide for details of additional steps that might be required. The book is available at:

VisualGen Version 2.2 uses MSLs similar to CSP. The information about migrating CSP MSLs to EGL applies to migrating VisualGen Version 2.2. MSLs except for differences noted below.

VisualGen Version 2.2 provides support for External Source Format. You can export the External Source Format, transfer it to a Windows operating system, and import it using the VAGen MSL Migration Assistance Tool as described in “Preparing to migrate your CSP/2AD source code”.

Table G2 shows the runtime environments supported by VisualGen Version 2.2 and the corresponding EGL runtime environment, if any. Earlier versions of VisualGen supported a subset of the VisualGen Version 2.2 runtime environments, with the specific environments varying by version.

<table>
<thead>
<tr>
<th>VisualGen Version 2.2 runtime environment</th>
<th>COBOL or C++ generation</th>
<th>EGL runtime environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIX</td>
<td>C++ generation</td>
<td>Not supported. Consider Java generation for AIX</td>
</tr>
<tr>
<td>CICS 6000</td>
<td>C++ generation</td>
<td>Not supported. Consider Java generation for AIX</td>
</tr>
<tr>
<td>CICS NT</td>
<td>C++ generation</td>
<td>Not supported. Consider Java generation for a Windows operating system</td>
</tr>
<tr>
<td>IMS BMP</td>
<td>COBOL generation</td>
<td>COBOL generation for IMS BMP</td>
</tr>
<tr>
<td>IMS/VS</td>
<td>COBOL generation</td>
<td>COBOL generation for IMS/VS (1)</td>
</tr>
<tr>
<td>MVS BATCH</td>
<td>COBOL generation</td>
<td>COBOL generation for ZOSBATCH</td>
</tr>
<tr>
<td>MVS CICS</td>
<td>COBOL generation</td>
<td>COBOL generation for ZOSCICS (1)</td>
</tr>
<tr>
<td>VisualGen Version 2.2 runtime environment</td>
<td>COBOL or C++ generation</td>
<td>EGL runtime environment</td>
</tr>
<tr>
<td>-----------------------------------------</td>
<td>-------------------------</td>
<td>-------------------------</td>
</tr>
<tr>
<td>MVS TSO</td>
<td>COBOL generation</td>
<td>Not supported</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Consider COBOL generation for ZOSCICS</td>
</tr>
<tr>
<td>OS/2</td>
<td>C++ generation</td>
<td>Not supported</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Consider Java generation for a Windows operating system</td>
</tr>
<tr>
<td>OS/2 CICS</td>
<td>COBOL generation</td>
<td>Not supported</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Consider Java generation for a Windows operating system</td>
</tr>
<tr>
<td>OS/400</td>
<td>COBOL generation</td>
<td>COBOL generation for iSeries (1) (2)</td>
</tr>
<tr>
<td>VM BATCH</td>
<td>COBOL generation</td>
<td>Not supported</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Consider COBOL generation for ZOSBATCH</td>
</tr>
<tr>
<td>VM CMS</td>
<td>COBOL generation</td>
<td>Not supported</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Consider COBOL generation for ZOSCICS</td>
</tr>
<tr>
<td>VSE BATCH</td>
<td>COBOL generation</td>
<td>Not supported</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(3)</td>
</tr>
<tr>
<td>VSE CICS</td>
<td>COBOL generation</td>
<td>Not supported</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(3)</td>
</tr>
<tr>
<td>Windows NT</td>
<td>C++ generation</td>
<td>Not supported</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Consider Java generation for a Windows operating system</td>
</tr>
</tbody>
</table>

Notes:
1. It is possible to call programs in this environment from the EGL debugger.
2. COBOL generation for iSeries is supported by Rational Business Developer Version 7.1. Rational Business Developer Version 7.1 also includes the build server and runtime support for iSeries, but you must upload them from Rational Business Developer to your iSeries host.
3. COBOL generation for VSE was previously supported for Rational Application Developer Version 6 using the VisualAge Generator EGL Plug-in for VSE Version 1.0 (program number 5724-L93). However, the plug-in does not support COBOL generation for VSE using Rational Business Developer Version 7.1. Consider generating COBOL for z/OS Batch and z/OS CICS.

For both the COBOL and C++ runtime environments in VisualGen Version 2.2, the following tasks differ from what is described in this paper for CSP Version 4.1:

- You can migrate VisualGen GUIs to VisualAge Generator Version 4.5 on Smalltalk. Each VisualGen GUI becomes a Smalltalk view. However, migration of the Smalltalk views to EGL is not supported. Refer to VisualAge Generator Migration Guide, Chapter 24 “General migration considerations for VAGen 2.x and Cross System Product to Smalltalk”, section “Migrating GUIs” for details on migrating GUIs to VAGen on Smalltalk. The associates of a VisualGen GUI for the purposes of the VAGen MSL Migration Assistance Tool are: records, tables, processes, statement groups, and other embedded GUIs and their associates. An external GUI and its associates are not considered to be associates.

- The control parts for VisualGen Version 2.2 are very similar to those used for VisualAge Generator Version 4.5. Therefore, it is easiest to convert your control parts to VisualAge Generator and from there to EGL. Refer to the VisualAge Generator Migration Guide, Chapter 32 “Completing your migration on Smalltalk”, section “Defining control information” for details. The main differences are for generation options and include the following options:
  - /DESTNAME, and /COBOL are obsolete. Eliminate these options from your generation options parts.
  - /OPTIONS, /LINKAGE, RESOURCE, /LINKEDIT, and /BIND point to directories and files in VisualGen Version 2.2, but point to parts in VisualAge Generator Version 4.5 and EGL.
Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user’s responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not 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.

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
IBM Corporation
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.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

**Trademarks**
The following terms are trademarks of the IBM Corporation in the United States or other countries:
AIX
CICS
CICS OS/2
ClearCase
DB2
IBM
IMS
iSeries
MVS
OS/2
OS/400
RACF
Rational
SAA
TeamConnection
VisualAge
Solaris, Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries or both.

Microsoft, Windows, Windows NT, Windows XP and the Windows logo are trademarks or registered trademarks of Microsoft Corporation in the United States, other countries, or both.

Linux is a registered trademark of Linus Torvalds.

Rational and ClearCase are trademarks of International Business Machines Corporation and Rational Software Corporation in the United States, other countries, or both.

Other company, product or service names may be trademarks or service marks of others.

### Glossary

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>cross-part migration</td>
<td>The situation in which the migration of one part depends on other parts. In some cases, information moves from one part to another related part.</td>
</tr>
<tr>
<td>CSP</td>
<td>Cross System Product</td>
</tr>
<tr>
<td>EGL</td>
<td>Enterprise Generation Language</td>
</tr>
<tr>
<td>ESF</td>
<td>External Source Format</td>
</tr>
<tr>
<td>MSL</td>
<td>Member Specification Library</td>
</tr>
<tr>
<td>PDS</td>
<td>Partitioned data set</td>
</tr>
<tr>
<td>VAGen</td>
<td>VisualAge Generator</td>
</tr>
</tbody>
</table>