Rational. IBM Informix

Version 6.0.0.1

IBM Informix 4GL to EGL Conversion Utility User's Guide

G251-2485-00

Rational. IBM Informix

Version 6.0.0.1

IBM Informix 4GL to EGL Conversion Utility User's Guide

G251-2485-00

Note!
Before using this information and the product it supports, read the information in "Notices" on page L-1.

First Edition (February 2005)

Contents

Introduction

In This Introduction

This introduction provides an overview of the information in this manual and describes the conventions it uses.

About This Manual

This manual provides information on how to convert IBM Informix 4GL (I4GL) applications into Enterprise Generation Language (EGL) applications. Used in Rational Application Developer (RAD) and Rational WebSphere Developer (RWD), EGL is a development technology that lets you quickly write full-function applications that run in a Java™ environment and on z/OS.

Types of Users

This manual is written for I4GL application developers who want to have the flexibility of creating web applications, want to extend the usability of their applications containing forms, want to use the new EGL reporting capability, want to use Message Queues with their EGL program, and want to use the full range of EGL functionality.

This manual assumes that you have I4GL programming experience, including experience with compiling I4GL code. In addition, if you use C code with your I4GL program, you must know C language programming.

Features of This Product

Information in this section covers both the I4GL to EGL Conversion Utility product and extended I4GL-like functionality within EGL.

The I4GL to EGL Conversion Utility converts your shared libraries and I4GL applications into EGL packages and programs. After the conversion of your shared libraries and I4GL applications, you can link your C functions with your converted EGL code.

EGL provides the following I4GL-like functionality:

Screen Forms. The Conversion Utility converts your existing I4GL screen forms into the EGL-equivalent Console User Interface.

Calling C functions. C functions can be invoked from an I4GL program. Similarly, EGL can invoke C functions.

Reports. The Conversion Utility converts your I4GL report files into equivalent EGL and JasperReports files.

Note: This product includes software developed by Teodor Danciu (http://jasperreports.sourceforge.net).

Product Installation

The I4GL to EGL Conversion Utility Version 6.0.0.1 is available for download through the Rational Product Updater. To install the Conversion Utility, you must have both the 6.0 version of your Rational product and the iFix004 version of your EGL plugin installed.

If you are reading this Guide before installing your Rational product and EGL plugin, you should review your Rational product Installation Guide first.The Installation Guide provides information on software and hardware requirements.

To install the I4GL to EGL Conversion Utility:

1. Log on to your system with the same user account used to install your Rational product.

   Note: On Windows operating systems, this account requires **Administer** privileges. On Linux operating systems, you must use the **root** user account.
2. Start the Rational Product Updater.
   - For Windows: Select **Start > Programs > IBM Rational > Rational Product Updater**.
   - For Linux: In the GNOME desktop environment, select the main menu, then select **Programming > Rational Product Updater**.
3. Click the **Find Optional Features** button. A progress indicator tells you the update sites are being searched.
4. Within the **Product** window, select the **Informix 4GL to EGL Conversion Utility** check box.
5. Click the **Install Features** button.
6. If the Rational Software Development Platform is running, you may be prompted to stop it. If so, click **OK** to close the dialog box. When you have stopped the Rational Software Development Platform, return to the Product Updater and again click **Install Features**.
7. A dialog box shows the license agreement for the Conversion Utility. Once you have read and accepted the license agreement, click **OK** to begin the installation.

When the installation is finished, the progress indicator closes and a message confirming the success of the installation appears in the top pane of the Product Updater. For more information about using the Rational Product Updater, see the *Rational Product Updater* and the *Finding and installing product updates* help topics in your Rational product.

Platforms Supported

To successfully convert your I4GL Programs to EGL Packages, you must have access to one of the following development environment platforms:

- Windows
- Linux

In addition, you must have sufficient system access and permissions to the source code and directory structure to allow the creation of shared libraries and new **.egl** source files.

Typographical Conventions

This section describes the typographical conventions that this manual uses. These conventions make it easier to gather information from this and other volumes in the documentation set.

Convention	Meaning
KEYWORD	All primary elements in a programming language statement (keywords) appear in uppercase letters in a serif font.
italics ***italics*** *italics*	Within text, new terms and emphasized words appear in italics. Within syntax and code examples, variable values that you are to specify appear in italics.
boldface ***boldface***	Names of program entities (such as classes, events, and tables), environment variables, file and pathnames, and interface elements (such as icons, menu items, and buttons) appear in boldface.
monospace *monospace*	Information that the product displays and information that you enter appear in a monospace typeface.
KEYSTROKE	Keys that you are to press appear in uppercase letters in a sans serif font.
>	This symbol indicates a menu item. For example, "Choose **Tools > Options**" means choose the **Options** item from the **Tools** menu.

Documentation

This section discusses the documentation you can use to assist with the I4GL conversion and to learn more about how to use EGL.

This *IBM Informix 4GL to EGL Conversion User's Guide* (User's Guide) leads I4GL users through the Conversion Path to EGL, and has the following chapters:

- This introduction
- Chapter 1, "Overview of the Conversion Process," which introduces the benefits and limitations of the I4GL to EGL conversion process and provides an overview of the pre-conversion, conversion, post-conversion, and reconversion stages.
- Chapter 2, "Preparing for Conversion," which describes the tasks users need to complete before using the Conversion Wizard. These tasks include identifying the components of your database schema and I4GL application, creating C shared libraries, and understanding your report font specifications.
- Chapter 3, "Conversion Tasks," which provides details on how to use the Conversion Wizard to extract your Informix database schema and convert your I4GL shared libraries and applications.

- Chapter 4, “Post-Conversion Tasks,” which describes the changes made by the Conversion Utility to your I4GL application, the artifacts generated during the conversion, how to correct conversion errors, and how to use the C shared libraries with your EGL application. This chapter also provides information on how to use the EGL Information Center online help system.
- Chapter 5, “Reconversion Process and Tasks,” which describes when and how to reconvert your I4GL shared libraries.
- Appendix A, “I4GL to EGL Syntax Mapping,” on page A-1, which provides the mapping between I4GL and EGL syntax.
- Appendix B, “I4GL Report Conversion Code Example,” on page B-1, which provides an example of I4GL report code and the comparable examples of the EGL driver functions.
- Appendix C, “I4GL Form Code to EGL Form Code Example,” on page C-1, which provides an example of I4GL form code and the comparable EGL code.
- Appendix D, “Configuration File Templates,” on page D-1, which provides template configuration files for Database Schema Extraction, Library, and Application projects.
- Appendix E, “Manifest File Examples,” on page E-1, which provides examples of manifest files for Database Schema Extraction, Library, and Application projects.
- Appendix F, “DTD Examples,” on page F-1, which provides examples of the DTD used for configuration and manifest files for Database Schema Extraction, Library, and Application projects.
- Appendix G, “Conversion Log Examples,” on page G-1, which provides examples of conversion logs in **.txt** format.
- Appendix H, “EGL Build Descriptor Example,” on page H-1, which provides examples of EGL build descriptor file generated by the Conversion Utility.
- Appendix I, “EGL Reserved Words,” on page I-1, which lists the EGL reserved words.
- A glossary of relevant terms and an Error Messages section follow the chapters.
- A Notices appendix describes IBM products, features, and services.
- An index directs you to areas of particular interest.

You should read this book in its entirety before attempting to convert your I4GL application and you should refer to this book when converting your I4GL applications.

Your Rational product includes a **readme.html** file. The **readme.html** file contains the latest information on any product limitations or changes to documentation for your Rational product. This file is located in the top directory of your Rational product.

In addition, the Conversion Utility includes a **readme004FGL.html** file. The **readme004FGL.html** file contains information on Conversion Utility limitations and changes to documentation since the completion of this User’s Guide. You should review the contents of the **readme004FGL.html** file before converting your I4GL applications. The file is located in the top directory of the Conversion Utility plugin.

When using the conversion wizard, you can activate the help for the specific panel you are using by pressing **F1**. Each panel of the Conversion Wizard has a dedicated help topic associated with it. Each help topic explains what information is required by the wizard panel. To activate the help topic, with your cursor on the wizard panel, press the **F1** key.

Information on how to use your converted EGL code is located in the following places:

- The Rational online information center. The information center is accessible from the main menu by selecting **Help > Rational Help**.
- The *EGL Reference Guide*. Provided in **.pdf** format, this guide is a compilation of all of the EGL information center topics, is indexed, and can be printed into hard-copy format. The filename for the guide is **eglref.pdf**, and the file is located in the top-level directory of your Rational product. The document is refreshed occasionally, and the most recent copy of the document is always located on the EGL website at www.ibm.com/developerworks/rational/library/egldoc.html.
- The EGL Tutorial teaches you how to build a simple dynamic Web site using EGL. The tutorial is accessible from the main menu by selecting **Help > Tutorials Gallery**. More information about the Tutorial is available in "The Information Center Help System and EGL Tutorial" on page 4-31.

IBM Welcomes Your Comments

We want to know about any corrections or clarifications that you would find useful in our manuals, which will help us improve future versions. Include the following information:

- The name and version of the manual that you are using
- Section and page number
- Your suggestions about the manual

Send your comments to us at the following email address:

docinf@us.ibm.com

This email address is reserved for reporting errors and omissions in our documentation. For immediate help with a technical problem, contact IBM Technical Support. The product home page for Rational Application Developer is www.ibm.com/software/awdtools/developer/application/index.html. For FAQs, lists of known problems and fixes, and other support information, visit the Support page on the product home page. You can also post questions through Rational Application Developer Forum.

We appreciate your suggestions.

Chapter 1. Overview of the Conversion Process

In This Chapter

This chapter introduces you to the benefits and limitations of the I4GL to EGL conversion process and provides an overview of the pre-conversion, conversion, post-conversion, and reconversion stages.

Introduction to the I4GL to EGL Conversion

The IBM Informix 4GL to EGL Conversion Utility (Conversion Utility) enables applications written in the Informix 4GL (I4GL) language to be converted to the Enterprise Generation Language (EGL). EGL lets you quickly write full-function applications that run in a Java™ environment. EGL gives you the ability to deliver enterprise data to browsers, even if you have minimal experience with Web technologies. You can use EGL to code a Web service that can be accessed by other Internet-based programs.

The Conversion Utility converts an I4GL program using a specific display device and connecting a specific IBM Informix Dynamic Server (IDS) database into an equivalent EGL program using the same specific display device and connecting to the same IDS database. The Conversion Utility converts I4GL language syntax and builds artifacts (like I4GL shared libraries) into equivalent EGL components. At runtime, the EGL build generates a Java executable program. EGL applications can work with IDS databases. Currently, EGL applications are not supported on IBM Informix XPS, Online, or SE databases.

The Conversion Utility offers you the ability to convert a program in one session, begin a conversion and finish when desired, or reconvert a shared library as necessary.

Conversion Benefits

I4GL users who convert their applications to EGL will be able to:

- Use the new EGL code exactly as I4GL was used, including the I4GL Forms interface (called Console User Interface in EGL). EGL code will also be able to call C functions and, with restrictions, call ESQL/C functions. Finally, EGL users will be able to generate I4GL-like reports.
- Use the new EGL code to create web applications.

I4GL to EGL Conversion Overview

A successful I4GL to EGL conversion has three successive stages, each of which is introduced below:

1. Pre-Conversion Stage
2. Conversion Utility Stage
3. Post-Conversion Stage

Note: Occasionally, you will need to reconvert your I4GL shared libraries. The Reconversion Stage is also introduced below.

Pre-Conversion Stage

Prior to initiating the conversion, you need to identify the components of your I4GL program and database schema and prepare your I4GL program files and libraries. This information is detailed in Chapter 2, "Preparing for Conversion," on page 2-1.

Conversion Stage

The conversion of I4GL applications is accomplished with the Conversion Utility Wizard. The Wizard screens prompt you to insert the information you organized during the pre-conversion stage. Information on how to use the Wizard is detailed in Chapter 3, "Conversion Tasks," on page 3-1.

Converting an I4GL program is a multi-tiered process as shown in Figure 1-2 on page 1-4. This figure summarizes the types of I4GL files that are converted, the order in which the files are converted, and the resultant converted file types; it also shows that **.c** files are not converted.

The components of your I4GL program must be extracted or converted in the following order:

- Database schema information
- I4GL shared library or libraries
- I4GL application

Note: C shared libraries do not need to be converted in order to be used by EGL, and they do not pass through the Wizard.

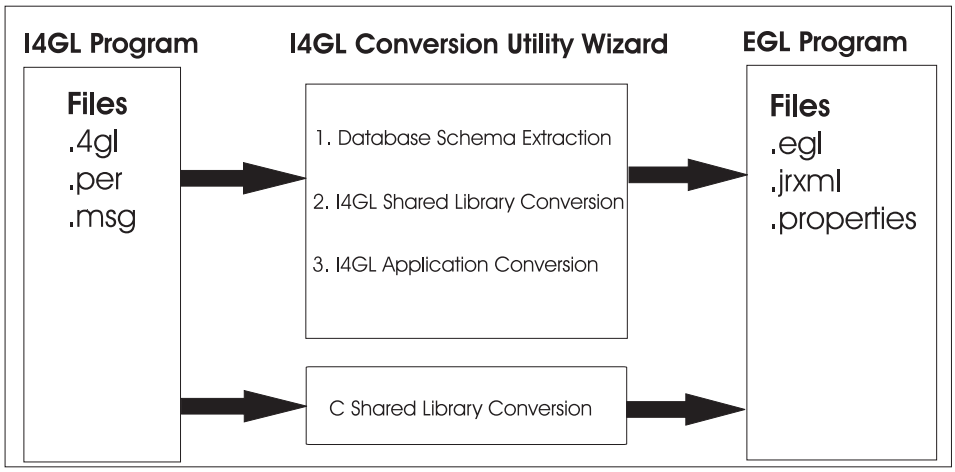

Figure 1-1. How an I4GL program converts to an EGL program

Post-Conversion Stage

A conversion log will identify whether or not your conversion was completely successful. If no errors are identified in the conversion log, the conversion was successful, and you can compile your EGL source files and use your converted application. If errors are identified in the conversion log, you must rectify those errors before using your EGL files. This information is detailed in Chapter 4, "Post-Conversion Tasks," on page 4-1.

Reconversion Stage

When shared libraries are converted in the wrong order, or reference the functions of another shared library, some function references assumed to be C functions are later defined in EGL. Any shared libraries generated with an incorrect assumption must be reconverted. In addition, shared libraries within applications containing multiple shared libraries may occasionally need to be re-converted. For more information on reconverting your shared libraries, see Chapter 5, "Reconversion Process and Tasks," on page 5-1.

Conversion Limitations

The I4GL to EGL Conversion Utility, Version 6.0.0.1 has the following limitations:

1. C functions that call I4GL functions are not supported.
2. I4GL programs that call ESQL/C are not supported.
3. Global variables defined in I4GL modules cannot be accessed from C source code.
4. Database connection sharing between ESQL/C or C shared libraries and migrated EGL Applications or Packages is not supported.
5. Dynamic 4GL is not supported.
6. You must use a third-party report design product to re-design converted reports or design new reports.
7. I4GL Reports converted to EGL can only be generated to Java; they cannot be generated to COBOL.
8. The performance of I4GL applications converted to EGL will be somewhat reduced.

9. Conversion of I4GL auxiliary products like Ace Reports and ISQL are not supported.
10. EGL does not support both global and local cursor scope in the same application or shared library. For example, you cannot convert a library with local scope and an application with global scope.
11. The I4GL Program database **syspgm4gl** is not supported during the conversion process.

Note: Task-oriented information about conversion limitations and their workarounds is detailed in "Conversion Limitations and Workarounds" on page 2-2.

C Interface Support and Limitations

Many I4GL users have I4GL programs that call C functions. With little or no modification, you will be able to use your existing C code in your I4GL converted code. For specific information on the tasks necessary to prepare your C code prior to using the Conversion Utility, see "Modifying C Code Used with Rapid Development System (RDS)" on page 2-5. For details on how to link the C shared library with the converted EGL application, see "Using C Shared Libraries with the EGL Program" on page 4-9. For details on how to use C functionality in EGL, see the online information center.

Report Support and Limitations

The Conversion Utility converts I4GL Reports files into EGL files for the drivers and ReportHandler, and into JasperReports files for the design values. With the creation of these new files, EGL users are able to use a report designer or text editor of their choice to design the layout of their reports.

Note: Your Rational product does not contain any third-party report design software. If you want third-party report design software, you will need to purchase this separately. For more information, see the *EGL reports overview* help topic.

Figure 1-2 shows how I4GL reports convert to EGL. First, you define the properties of the existing I4GL report program and enter it into the Conversion Utility Wizard. In turn, the Utility Wizard produces the following output: JasperReport XML design documents; EGL report driver functionality; and an EGL report handler.

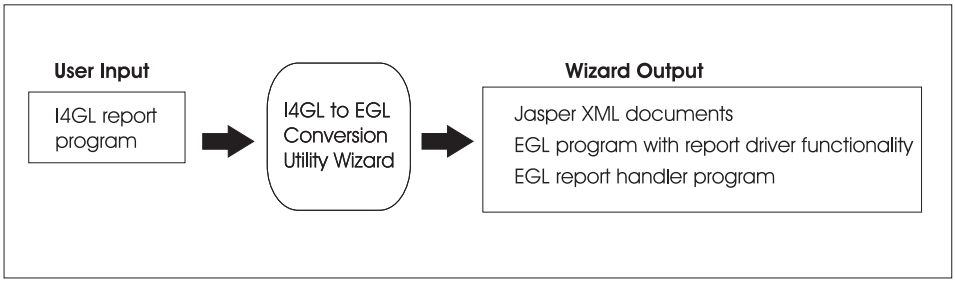

Figure 1-2. How I4GL Reports Converted to EGL

For information on how to prepare your I4GL reports for conversion to EGL, see "Conversion Limitations and Workarounds" on page 2-2. For information on how to understand your reports conversion, see "Understanding Report Conversion" on page 4-14

page 4-14. To review the syntax mapping between I4GL Reports, EGL and JasperReports, see Appendix A, "I4GL to EGL Syntax Mapping," on page A-1. For information on how to use the EGL Reports feature in EGL, see the *EGL reports overview* help topic.

Screen Forms Support

While the conversion of I4GL to EGL provides users with an opportunity to implement web-based applications with graphical user interfaces, the Conversion Utility also enables users to retain the appearance and functionality of their I4GL screen forms, or text-based user interfaces. During the conversion of your I4GL application, I4GL screen forms will be converted into EGL Console User Interfaces (CUI). The CUI will have exactly the same functionality as the screen forms. In addition to running your converted CUI, you can also develop new CUIs.

To review the syntax mapping between I4GL forms and the CUI, see "I4GL Forms to EGL Console User Interface" on page A-8. For details on how to use the Console User Interface in EGL applications, see the *Console user interface* help topic.

Chapter 2. Preparing for Conversion

In This Chapter

This chapter describes the tasks you need to complete before using the Conversion Wizard. These tasks include identifying the components of your database schema and I4GL application, creating C shared libraries, and understanding your report font specifications.

Overview of Pre-Conversion Tasks

For a successful conversion from I4GL to EGL, the Conversion Utility Wizard requires specific information about your I4GL application. The pre-conversion steps listed below will help you prepare your files for conversion and collect the information required by the Wizard.

The following steps are detailed in this chapter:

1. Verify the installation of the Conversion Utility.
2. Understand the conversion limitations.
3. Identify the I4GL project you want to convert, and record the names and source file locations of each of the I4GL applications within the project. Identify and record the names and locations of all dependent I4GL shared libraries and C shared libraries linked with the I4GL application. As necessary, you will need to list the shared libraries from already-converted EGL packages.
4. As needed, generate **.4gl** and **.per** files.
5. Verify that all I4GL source files compile successfully using the I4GL 7.32 compiler.
6. If any of your I4GL programs are compiled on non-English locales, identify the client locale.
7. Identify whether or not the I4GL build environment for a shared library contains both .**4gl** and **.c** source files. If it does, segregate the **.4gl** files and convert them as an I4GL shared library project. Compile the **.c** files into a shared library.

8. Identify user-defined message files and the code pages in which the messages are encoded.
9. Start an IBM Informix database instance, create the database schema used by I4GL modules, and record the following database connection information: Database name, Server name, Host name, Port Number, Client Locale and Database Locale. During conversion, you will also need to know your user name and password.
10. Identify a destination directory for your EGL files.
11. Move your I4GL source files into the I4GL staging directory on the machine on which you want to run the Conversion Utility.

Verify the I4GL Conversion Utility Installation

From your Rational product main screen, you can verify the existence of the Conversion Utility by selecting **File > New > Other** and then selecting the **Show All Wizards** option. If the option **Informix 4GL to EGL Conversion** is in the list, the Conversion Utility is installed.

If the **Informix 4GL to EGL Conversion** option is not in the list, you must install the Conversion Utility following the instructions provided in "Product Installation" on page viii.

By default, the Conversion Utility is disabled. To activate the Utility, from the Main menu select **Window > Preferences > Workbench > Capabilities > Informix 4GL to EGL Conversion**.

Conversion Limitations and Workarounds

The conversion limitations, with workarounds as possible, are listed below:

C Code Functionality

In the I4GL language, C code has two implementations:

- C functions can be called by I4GL programs, and then using push and pop external functions, the C functions can call an I4GL function.
- C programs can call I4GL functions by using the **fgl_start()**, **fgl_call()**, **fgl_exitfm()**, and **fgl_end()** macros.

In the EGL language, an EGL function can call a C function, but a C function cannot call an EGL function. Prior to converting your I4GL application to EGL, you must remove from your code all instances of C functions calling I4GL functions. In addition, you must remove the push functions used to pass values to the I4GL function and the pop functions used to retrieve values returned by the I4GL function.

Global variables defined in EGL cannot be used by the C code. Prior to conversion, you must modify your I4GL and C code to explicitly pass such global variables to the C code and then reset the values upon return from the C call.

Because EGL is converted into Java at runtime, an additional layer of JNI (Java Native Interface) is involved when calling C functions. Therefore, the performance of EGL applications calling C functions may be slower than the performance of I4GL applications calling C functions.

Reports

The Conversion Utility converts your I4GL report logic into two types of files:

- Multiple EGL (**.egl**) files for the ReportHandler and the ReportDriver functions
- A JasperReports (**.jrxml**) file for the report design

The Utility requires font format specifications to create the **.jrxml** file. During the conversion of your I4GL application, the Wizard **Cursor Scope and Report Font File** screen will prompt you to accept the default font specifications or to designate alternate font specifications.

The default font specifications are:

```
locale name = en_us
      name = Courier New
      size = 10
      height = 12
      width = 6
      pdfFont = Courier
      encoding = CP1252
```

In addition, the **FontInfo.xml** file located in the ***productinstallation*/etc/examples/FontSpecification** directory also lists these default values. To designate alternative non-default font settings when using the Wizard, you must edit the **FontInfo.xml** file to include your preferred font information.

The following is an example of the default **FontInfo.xml** file:

```
<locale name="en_us">
      <name>Courier New</name>
      <size>10</size>
      <height>12</height>
      <width>6</width>
      <pdfFont>Courier</pdfFont>
      <encoding>CP1252</encoding>
</locale>
```

Define the following **.xml** fields in your file:

- **locale name**: the language and country identified with the font. Each locale is limited to one font specification. However, you can include multiple different locale specifications in the same **.xml** file.
- **name**: the name of the font.
- **size**: the size of the font, which must be an integer literal, such as ″10″.
- **height**: the height of the font in pixels, which must be an integer literal, such as ″12″.

  The height value is used to calculate report elements requiring vertical positioning, including static text field, text field, band and page height, and top and bottom margin size.
- **width**: the width of the font in pixels, which must be an integer literal, such as ″6″. To ensure that your EGL report output produces output similar to your I4GL report, the font must have a fixed width.

The width value is used to calculate any report elements requiring horizontal positioning, including static text field, text field, page and column width, and left and right margin width.

- **pdfFont**: the name of the font to use if a report is exported to Adobe PDF format.
- **encoding**: the alphanumeric code that designates the code pages in Java.

After you edit **FontInfo.xml** to specify your font specifications, you should rename the file to a filename ending in **.xml**. If for any reason, your designated font specification is corrupted, the conversion tool will map the default information into your .**jrxml** file.

Note: The Conversion Utility does not attempt to verify the availability of your selected font name, font size, PDF Font or encoding on your system. These values will be transferred as specified from the font specification file to the JasperReport Design (**.jrxml**) file. Incorrect font information can result in EGL compilation or runtime errors.

You should follow these font specification selection guidelines:

- Use a font that has both fixed width and fixed height.
- Use a similar font for the general report export options and the PDF export option.
- The font height and font width should reflect the number of pixels dimensions of the given font type and size.
- Verify that the encoding string accurately reflects the locale.

Identify Existing I4GL Components Project

Once you have selected the I4GL program to convert, you must:

1. Identify and record the I4GL source modules and locations for each of your I4GL applications.
2. Identify and record the file names and locations of each of the I4GL and C shared libraries linked to the I4GL applications
3. As applicable, you must list the names and location of the manifest file from already-converted I4GL shared libraries.

Note: You will be required to recompile the C shared libraries later during the Pre-Conversion stage.

Generate I4GL Source Files

As needed, generate **.4gl** and **.per** source files.

Any pre-processing required to generate the I4GL source files from the I4GL modules must be completed at this time.

Compile your I4GL Application

Compile your I4GL application code as appropriate and verify that it compiles successfully. If your I4GL application code does not compile successfully, your I4GL to EGL conversion will fail. Repair any code that does not compile successfully. Do not attempt an I4GL to EGL conversion until all I4GL application code compiles successfully.

Identify the Client Locale

The Conversion Wizard uses the English designation of **en_US.8859-1** as the default locale for message files. If your message files are not in English, you must identify the correct locale for your message files and have that information available to insert into the Wizard.

Identify and Separate the Shared Libraries

If your I4GL program has shared libraries, you must review each library to identify whether it was compiled from **.4gl**, **.c**, or both types of source files.

1. A shared library compiled from only **.4gl** source files does not require further conversion preparation. It should be converted as an I4GL shared library project.
2. A shared library compiled from only **.c** source files does not require further conversion preparation. It should be linked when creating the application level shared library.
3. A shared library compiled from both **.4gl** and **.c** source files requires the following modifications:
   a. The **.4gl** files must be segregated and converted as an I4GL shared library project.
   b. The **.c** files must be compiled into a shared library. This shared library should be linked when creating the application level shared library. For instructions on how to link the shared library, see "Using C Shared Libraries with the EGL Program" on page 4-9.

After your I4GL application is converted, you will create an application level shared library and link your C shared libraries to it. This application level shared library will be used with your new EGL application. For more information, see "Using C Shared Libraries with the EGL Program" on page 4-9.

Note: If your program uses pre-existing static libraries consisting of object files compiled from **.c** source files, these static libraries should also be linked when creating the application level shared library.

Modifying C Code Used with Rapid Development System (RDS)

Because RDS uses a customized runner to call C functions from I4GL programs, I4GL developers who use RDS must modify their C code as detailed below.

Note: A *customized runner* is an executable program that users create to run I4GL programs that call C functions.

The customized runner is created with the **cfglgo** command, as shown in the following syntax:

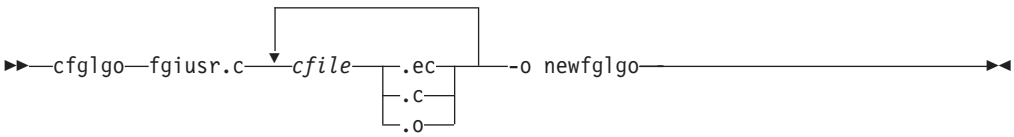

Table 2-1. **cfglgo** *elements*

Element	Description
fgiusr.c	the name of the file in which the C and ESQL/C functions are declared.
cfile .ec | .c | .o	the name of the source file containing the C or ESQL/C functions to be compiled and linked with the new runner, or the name of an object file previously compiled from a **.c** or an **.ec** file.
newfglgo	the name of the customized runner.

If you use RDS, you must compile all **.o**, **.c** or **.ec** files into a shared library. If any **.ec** files are compiled in the library, you must use ESQL/C shared libraries when compiling the library. This shared library should be linked when creating the application level shared library. For more information, see "Using C Shared Libraries with the EGL Program" on page 4-9.

For more information about the **cfglgo** command, see Chapter 1 of the *IBM Informix 4GL Reference Manual*.

Identify User-Defined Message Files

Identify all user-defined message files and the code pages in which the messages are encoded.

Identify Informix Database Schema Information

The first step in the conversion process is to extract the database schema for your conversion application.

1. Create an instance of the database for your I4GL program
2. Locate and open the **schema** file
3. Record the following information:
   - Database name
   - Server name
   - Host name
   - Port name
   - User name
   - Password
   - Database Locale

By default, the Conversion Utility ignores system tables. However, if the I4GL applications using the database refer to any Informix system tables, you have the option of the extracting the schema for all system tables in the database.

Identify an EGL Destination Directory

The Conversion Utility uses the EGL destination directory to organize the converted EGL source files, message files, and project-specific files required by the Rational IDE.

Based on the information provided to the Wizard, the Conversion Utility creates an EGL destination directory for you. For example, if your identified EGL destination directory is **C:\egl\src**, and your project name is ***MyProject***, within the EGL

destination directory, the Conversion Utility creates a new sub-directory structure of ***projectname*\EGLSource*projectname*** Following the example parameters above, the new directory structure is **C:\egl\src\MyProject\EGLSource\MyProject.**

The first instance of ***project name*** hosts the **.classpath**, **.eglPath**, and **.project** Rational IDE-specific files, and serves as the project contents directory for importing the converted project into the IDE.

The **EGLSource** sub-directory serves as the placeholder for all converted EGL source files, and contains the ***projectname*** sub-directory, which represents the root package name for all converted EGL files. The directory hierarchy of converted the EGL files is identical to the I4GL source directory hierarchy, and resides within the ***projectname*** sub-directory.

The Conversion Utility creates the follow directories within the EGL destination directory:

- ***projectname*\EGLSource*projectname***. Hosts all converted EGL source files.
- ***projectname*\MessageSource**. Hosts all converted EGL Message files.
- ***project name*\JavaSource**. Hosts all generated Java Source files from EGL source in IDE.

Note: If the same directory is identified for both your existing I4GL source files and your new EGL source files, the conversion utility creates the new directory structure before initiating the conversion.

The Conversion Utility stops the conversion if your system does not have write permissions or enough disk space in the EGL destination directory. For a successful conversion, you should have disk space at least equivalent to your existing I4GL directory.

Prepare the I4GL Source File Directory

Prior to launching the Conversion Wizard, you must ensure that all of your I4GL source files, shared libraries, and message files are located together in a directory. This I4GL directory can be located within your Rational product workspace or in a location entirely outside of your Rational product.

In addition, you can designate the same directory for both the storage of the I4GL program source files and as the destination directory for the EGL project files. With this option, the integrity of the I4GL source files is retained during conversion, and results in a directory of I4GL and EGL source files.

Note: Although the Conversion Utility allows you to locate all of your I4GL projects within, and to convert your I4GL projects from, one single directory, you should locate your I4GL projects within individual directories. This reduces the changes of complications arising from multiple similar files.

Chapter 3. Conversion Tasks

In This Chapter

This chapter provides details on how to use the Conversion Wizard to extract your Informix database schema and convert your I4GL shared libraries and applications. This chapter also describes how to use the Conversion Utility command line option, and introduces the Conversion Log.

Conversion Utility Stages

The I4GL Conversion Utility is a Wizard that prompts you for the information you were required to collect and identify in Chapter 2, “Preparing for Conversion.”

To complete the conversion, the components of your I4GL program must pass through the Wizard multiple times. In addition, your information must pass through the Wizard in the following order:

1. Informix Database Schema
2. I4GL Shared Libraries
3. I4GL Application

Since C shared libraries are not converted to EGL, they do not pass through the Wizard. For complete information on the processing of your C shared libraries, see “Using C Shared Libraries with the EGL Program” on page 4-9.

As mentioned in “Documentation” on page ix, each panel of the Wizard has a help topic associated with it. Access this help topic by pressing **F1**.

Informix Database Schema Extraction

The first step in the conversion process is to extract the project database schema.

Note: To limit the number of imports generated for your converted files, you should extract each database as a separate project. Each database in a Database Schema Extraction project will be a separate package, but all databases are listed in the single manifest file generated for the Database Schema Extraction project. The Conversion Utility inserts imports for all packages listed in the manifest file even if they are not referenced by the I4GL files.

To start the Utility and extract the Informix Database Schema:

1. From within the EGL Perspective in your Rational product, select **File > New > Other > Informix 4GL to EGL Conversion > Informix Database Schema Extraction**.
2. In the Database Schema Extraction screen, insert the following information:
   - **Project Name**. For example, **Mydbschema**.

     Note: The project names for database schema extraction and for shared library and application conversion should be different.
   - **New Project**. A database schema extraction is considered **New** if it does not have an existing configuration file. For **Location**, you can browse to locate the appropriate directory or you can create a new directory.
   - **Open Project**. A database schema extraction is considered **Open** if it has an existing configuration file. If you select this option, you must provide the location of the existing configuration file. You can browse to locate the file.
   - **Conversion Artifacts**. The Conversion Utility generates a number of artifacts related to the conversion, including Configuration, manifest and Conversion Log files. By default, the conversion artifacts will be located in the *EGLDestinationDirectory*/**ConversionArtifacts** directory. You can also designate to create the conversion artifacts in an external directory. You may browse to an appropriate directory, but cannot create a new directory at this time.
3. **Database Connection Details**. Add, delete, or edit information required to connect to your Informix database server instance, including
   - **Database**
   - **Server Name**
   - **Host Name**
   - **Port Number**
   - **User Name**
   - **Password**
   - **Client Locale**
   - **Database Locale**
   - **System Tables**. If the I4GL application requires extraction of the Informix system tables, select **Yes**.
4. **Conversion Project Details**. You can review the configuration file, an **.xml** document that coordinates all of the project details entered in the previous screens. At this stage of the conversion process, the manifest file and the Conversion Report screens are not populated with information.
5. Click **Finish** to initiate the database schema extraction.
6. Click **Yes** to confirm and start the database schema extraction.
7. The **Schema Extraction Status** screen confirms whether or not the extraction was successful. If the extraction was successful, you can review the now-populated manifest file, which contains column and table information for the database selected. The Wizard refers to the manifest file during the conversion of any I4GL shared library or application containing a DATABASE statement. Continue as appropriate to "I4GL Shared Libraries Conversion" on page 3-3 below or to "I4GL Application Conversion" on page 3-4.

   A conversion log file identifying the detailed status of the converted project is created.

   If the **Schema Extraction Status** message is not successful, you must repeat this Database Schema Extraction process from the beginning and correct any

erroneous information that you might have entered. For more information why the database schema extraction might be unsuccessful, see "Correcting Conversion Errors" on page 4-6.

8. Click **Cancel** to close the Wizard.

Conversion Utility Processing for Informix Database Schema Extraction

The Database Schema Extraction stage initiates the following:

1. The Conversion Utility creates an individual *EGL package* for each of the databases used in the I4GL program. Each of these EGL packages has an **.egl** file relating to the database. The file has an SQLRecord definition mapping to the table and dataitems associated with each column.
2. For each table in the database, a corresponding EGL source file is generated.
3. The collected database schema becomes a separate *EGL project* which is referred to from other EGL projects.
4. A configuration file and a manifest file are generated.
5. Only Informix data types supported within the IBM Informix 4GL 7.32 release are extracted. Any other data types are ignored.

I4GL Shared Libraries Conversion

If your I4GL application uses shared libraries, you must convert the shared libraries after extracting the Informix Database Schema and before converting the I4GL application. If your I4GL application has multiple shared libraries, you must use the Conversion Utility to convert each shared library separately. In essence, each shared library becomes a separate conversion project.

Because of the dependencies between the converted shared libraries and the I4GL application, the I4GL application can only be converted after all I4GL shared libraries have been converted.

Note: This I4GL Shared Library stage is not used for C Shared Libraries. For information on the post-conversion processing of your C shared libraries, see "Using C Shared Libraries with the EGL Program" on page 4-9.

To convert I4GL Shared Libraries:

1. From within the EGL Perspective in your Rational product, select **File > New > Other > Informix 4GL to EGL Conversion > I4GL Shared Library Conversion.**
2. In the **I4GL Shared Library Conversion Project** screen, insert the following information:
   - **Project Name**. For example, **Mysharedlibrary**.

     Note: The project names for database schema extraction and shared library and application conversion should all be different.
   - **Project Details**. Options include:
     - **New Project**. A shared library project is considered **New** if it does not have an existing configuration file. Selecting this option requires you to provide the I4GL Source Directory and the EGL Source Directory. You can browse to locate these directories.

- **Open Project**. A shared library project is considered **Open** if it has an existing configuration file. Selecting this option requires you to provide the location of the existing configuration file. You can browse to locate the file.
- **Reconversion Project**. A shared library project is considered a **Reconversion** if you previously completed an I4GL shared library project, but the I4GL application conversion determines that a reconversion of your shared library files is necessary. Selecting this option requires you to provide the location of the existing configuration file and manifest files. You can browse to locate the file.

  Note: For information on when reconversion is necessary, see "When to Reconvert Your I4GL Shared Libraries" on page 5-1.

- **Conversion Artifacts**. The Conversion Utility generates a number of artifacts related to the conversion, including configuration, manifest and conversion log files. By default, the Conversion Artifacts will be located in the *EGLDestinationDirectory*/**ConversionArtifacts** directory. You can designate to create the conversion artifacts in an external directory. You can browse to an appropriate directory, but you cannot create a new directory at this time.

3. Click **Finish** to launch the shared library project conversion.
4. Click **Yes** to confirm and start the shared library project conversion.
5. The **Shared Library Status** screen confirms whether or not the conversion was successful. If the conversion was successful, you can continue as appropriate with another I4GL Shared Library conversion or with an I4GL application conversion.

   If the **Shared Library Status** message is not successful, you must repeat this shared library conversion process from the beginning and correct any erroneous information that you might have entered. For more information on why the shared library conversion might have been unsuccessful, see "Correcting Conversion Errors" on page 4-6.

Conversion Utility Processing for I4GL Shared Libraries

At the I4GL Shared Libraries stage, the launch of the I4GL2EGL conversion utility initiates the following:

1. The Conversion Utility creates an individual EGL package for each of the I4GL shared libraries.
2. The entire group of EGL packages are generated into an EGL project which can then be referred to by other EGL applications.
3. Manifest files are generated for the EGL Project.

Note: I4GL shared libraries can also be converted using a command line mode. However, this option is recommended for reconversion efforts only. For instructions on how to use the command line mode, see "How to Reconvert Your I4GL Shared Libraries" on page 5-1.

I4GL Application Conversion

Your I4GL application can be converted after the Informix database schema has been extracted and all I4GL shared libraries have been converted.

To convert your I4GL application:

1. From within the EGL Perspective in your Rational product, select **File > New > Other > Informix 4GL to EGL Conversion > I4GL Application Conversion.**

2. In the **I4GL Application Conversion Project** screen, insert the following information:
   - **Project Name** For example, **Myapplication**.

     Note: The project names for database schema extraction and shared library and application conversion should all be different.

     – **New Project**. A project is considered **New** if it does not have an existing configuration file. Selecting this option requires you to provide the I4GL source directory and the EGL source directory. You can browse to locate these directories.

     – **Open Project**. A project is considered **Open** if it has an existing configuration file. Selecting this option requires you to provide the location of the existing configuration file. You can browse to locate the file.

     Note: Although you can reconvert your shared libraries, you cannot reconvert your I4GL application. If the conversion of your I4GL application fails, you must resolve the errors as noted in the Conversion Log and convert your I4GL application as a **New** or **Open** project.
   - **Conversion Artifacts**. The Conversion Utility generates a number of artifacts related to the conversion, including configuration, manifest and conversion log files. By default, the conversion artifacts are located in the *EGLDestinationDirectory*/**ConversionArtifacts** directory. You can also designate to create the conversion artifacts in an external directory. You can browse to an appropriate directory, but you cannot create a new directory at this time.
3. Click **Next** to continue.
4. The I4GL Conversion **Project Files** screen presents following information:
   - **Client Locale** displays the information you provided during the database schema extraction stage and is used to correctly convert message files. You can not change your locale at this time.
   - **Project Files** displays the folders, sub-folders and files located within the I4GL source directory you entered in the previous **I4GL Application Conversion Project** screen. You have the following options:
     – **Filter types**. You can limit the files selected in the **Project Files** screen. The **Select Types** dialog box provides a list of file types that are currently supported in your Rational product. The .**4gl**, .**per** and .**msg** files are selected by default. Enter message file extensions other than ***.msg** in the **Other Extensions** text box.
     – **Select all** will select all files within the root directory displayed on the screen.
     – **Deselect all** will deselect all folders, sub-folders and files within **Project Files**.
   - **Click** the boxes associated with the I4GL folders and files to be converted.
   - Click **Next** to continue.
5. The **Locale for Message Files** screen displays only if you selected message files in the previous **Project Files** screen. The **Locale for Message Files** screen displays your selected message files.
   - Click **Next** to continue.
6. The **Cursor Scope and Report Font File** screen confirms the status of the database cursor scope and the I4GL reports font file location. If you want to

accept the defaults, you can skip this page. If you do not want to accept the defaults, you must complete the following actions:

- The default for **Database Cursor Scope** is **local**. Select the **Set SQL cursor scope for conversion** check box if you want to change the cursor scope to **global**.
- The default for **Set font file location for report** is identified in "Reports" on page 2-3. Check the **Set font file location for report** box if you do not want to accept the default. In the **Font file** dialog box, enter or browse for the location of the font file.
- Click **Next** to continue.

7. The **Database Schema Details** screen confirms the existence of the database schema needed for the conversion and the manifest file containing the schema information. If your I4GL program does not contain a DATABASE statement, no action is necessary for this screen.

   This screen requires the following actions:

   - Click the **Dependent Database Schema** box if you successfully completed the database schema extraction stage.
   - Enter the name of the **Default Server**.
   - Click **Add** to display the **Add Manifest File from Dependent Project** box.
   - Enter the location of the database schema **Manifest File**. This file is located in the **ConversionArtifacts** directory that you created during the database schema extraction stage.
   - Select **OK** to continue.
   - The **Project Name** and the **Database Schema Extraction Manifest File** location now display on the screen. You can add to, delete or edit this information by clicking the **Add**, **Delete** or **Edit** buttons.
   - Click **Next** to continue.

8. The **Conversion Project Dependencies** screen identifies all EGL packages (former I4GL shared libraries) upon which the converted application depends.

   As appropriate, select the box for:

   - Dependent I4GL Shared Library Projects
     - Click **Add** to display the **Add Manifest File from Dependent Project** box.
     - Enter the location of the database schema **Manifest File**. It is located in the **ConversionArtifacts** directory that you created during the I4GL shared library conversion stage.
     - Select **OK** to continue.
     - The I4GL Shared Library Project Name and manifest file location now display on the screen. You can add to, delete or edit this information by clicking the **Add**, **Delete** or **Edit** buttons.

9. Click **Next**.
10. **Conversion Project Details**. If your project is a **New Project**, you can review the configuration file, an **.xml** document that coordinates all of the project details entered in the previous screens. At this stage of the conversion process, the manifest file and the Conversion Report screens are not populated with information. If your project is an **Open** or a **Reconversion Project**, this screen displays the populated manifest file and conversion report.
11. Click **Finish** to initiate the conversion.
12. Click **Yes** to launch the I4GL2EGL conversion program.

Conversion Utility Processing for I4GL Application Conversion

At the I4GL application stage, the launch of the I4GL2EGL conversion utility initiates the following:

1. The parser converts the selected I4GL application source files.
2. A manifest file containing function calls, function arguments, global variables and other information is created.
3. The converted I4GL shared libraries and the I4GL source code manifest files from dependent shared library projects are reviewed for consistency and possible errors.
4. Each message file is read and converted to the appropriate .**properties** file using the equivalent Java code page.
5. If the current project defines an EGL function with a name previously assumed to be an external function, the conversion program updates and modifies the dependent I4GL shared library project manifest file, terminates the conversion, and reports the error in the conversion error log. At this point, you must reconvert the I4GL shared library using the updated manifest file.
6. All **.4gl**, **.per**, **.msg**, and any customized error messages files are converted to EGL.
7. Any I4GL errors are identified, and the conversion program generates an **.err** file within the EGL destination directory and updates the log file with the error information.
8. If the conversion is successful, a default EGL build descriptor file is generated in *EGLdestinationdirectory/project name/***EGLSource**. This file is named ***project name***.**eglbld**. If a file by this name already exists in the EGL destination directory, the Conversion Utility does not create a new file to override the existing file.
9. If the program being converted has C library dependencies, the conversion utility identifies each C function call and generates a function table. The function table is named **NativeFuncTab.c** and is located in the **ConversionArtifacts/NativeLibrary** directory. For more information on the function table, see "Function Table" on page 4-9.
10. The conversion log is created and populated with conversion statistics and details.
11. The conversion log file automatically opens for viewing.
12. Click **Cancel** to terminate the Wizard.

Conversion Utility Command Line Mode

Instead of using the Conversion Utility Wizard to convert your I4GL applications to EGL, you can use the Conversion Utility command line mode. However, since using the command line mode for the initial conversion process requires you to manually create a configuration file, you might find it easier to use the command line mode primarily for shared library reconversion, when an existing configuration file can be used.

The command line option is invoked through a script named **e4gl**, which is located in the ***productinstallation*/bin** directory.

This script is found in the following directory:

- Linux:
  `/opt/IBM/Rational/SDP/6.0/bin`
- Windows:

```
C:\Program Files\IBM\Rational\SDP\6.0\bin
```

Prior to using the command line script, you must create a configuration file for your conversion project. Your configuration file must be in **.xml** format and follow exactly the Data Type Definition (DTD) provided in the sample configuration file. You should not create a configuration file without using the sample file as a template.

To create a configuration file:

1. Locate the sample configuration file in the following directory:
   `plugins/com.ibm.etools.i4gl.conversion_version/etc/dtd/Configuration`
2. Open the file in an **.xml** editor, rename the file, and save it to the preferred directory.
3. Your new file contains a DTD which provides intuitive **.xml** tags. Use these tags to insert your information into the file.
4. Save the file.

To use the command line option:

1. From the command line, enter the following, where *configurationfile* is the name of your newly-created configuration file:
   `e4gl configurationfile`

If your configuration file does not reside in the current working directory, you must qualify it with an absolute directory path.

For information on how to use the command line utility to convert an I4GL application, see "Command Line Reconversion" on page 5-2.

Note: If the script is moved from the default ***productinstallation***/bin directory, the RAD_HOME environment variable (which is used by the script to locate the conversion **.jar** files) must be set to point to the root directory of your Rational product installation.

The Conversion Log

At the end of the I4GL to EGL conversion, the conversion log opens for display. The log contains conversion details, including information about warnings, fatal errors, shared libraries and source file conversion status. The conversion log is primarily used to identify conversion errors. For more information on how to use the conversion log to resolve conversion errors, see "Correcting Conversion Errors" on page 4-6.

Chapter 4. Post-Conversion Tasks

In This Chapter

This chapter describes the changes made by the Conversion Utility to your I4GL application, the artifacts generated during the conversion, how to correct conversion errors, and how to use the C shared libraries with your EGL application. This chapter also provides information on how to use the EGL information center online help system.

Post-Conversion Tasks

Before using your converted EGL program, you should review or implement the following tasks and information as appropriate:

- "Changes Made During the Conversion."
- "Artifacts Generated During the Conversion" on page 4-3.
- "Source File Conversion Mapping" on page 4-5
- "Command Line Conversion: Importing Projects into the Workspace" on page 4-6.
- "Correcting Conversion Errors" on page 4-6
- "Using C Shared Libraries with the EGL Program" on page 4-9
- "Properties Files" on page 4-11
- "Validating and Compiling Converted EGL Files" on page 4-12
- "Generating EGL to Java" on page 4-13
- "Understanding Error Message Conversion" on page 4-14
- "Understanding Report Conversion" on page 4-14
- "Understanding your EGL Projects, Packages and Files" on page 4-27
- "The Information Center Help System and EGL Tutorial" on page 4-31

Changes Made During the Conversion

During the three stages of the I4GL to EGL conversion, the following changes occur:

- The Conversion Utility creates a new root directory structure to host the converted EGL source files in the EGL destination directory. A workspace resource folder was also created, which by default, has the following subdirectory: ***EGLDestinationDirectory*/*ProjectName*/EGLSource/*ProjectName***.
- If the converted application is dependent upon any C shared libraries, EGL native libraries and a function table are generated. For information about how to use the EGL native library and the function table to create the link between your C shared libraries and EGL program, see "Using C Shared Libraries with the EGL Program" on page 4-9.
- An EGL project build descriptor file is created and used to build the converted **.egl** files in the EGL Perspective.
- I4GL syntax constructs are mapped to EGL syntax. For a complete mapping between I4GL and EGL, see Appendix A, "I4GL to EGL Syntax Mapping," on page A-1.
- All **.4gl** source files are converted into **.egl** source files.
- All I4GL .**per** form specification files are converted into **.egl** Console User Interface specification files.

  Note: In EGL, all files are **.egl** files. For a complete mapping between I4GL and EGL files, see "Source File Conversion Mapping" on page 4-5.
- I4GL customized error message files are migrated to EGL. For information about those error message files, see "Understanding Error Message Conversion" on page 4-14.
- I4GL report logic is converted into equivalent EGL and JasperReports format. For information on your converted reports, see "Understanding Report Conversion" on page 4-14. To review a mapping between I4GL and EGL, see "4GL Report Execution Statements" on page A-11 and "I4GL Report Driver Statements" on page A-11.

Note: If your I4GL program did not contain report functionality, you can still add that functionality to your converted program. For information on how to create new reports in your EGL project, see the **EGL Report** help topics in the information center.

Artifacts Generated During the Conversion

The Conversion Utility generates two broad categories of conversion artifacts:

- Project-specific artifacts, such as configuration and manifest files.
- Error files, which are generated from the syntax errors in I4GL input files.

The error files are placed into the same directory used by the input I4GL file. By default, all other conversion artifacts are placed in the *EGLDestinationDirectory*/**ConversionArtifacts** directory. You may designate a different directory location during the database schema extraction or I4GL shared library conversion stage of the Conversion Utility.

The Conversion Artifacts directory contains the following sub-directories:

1. **/config**. Placeholder for the Conversion Utility to generate a configuration file for **New** projects.
2. **/manifest**. Placeholder for the Conversion Utility to generate a manifest file for the current conversion projects.
3. **/log**. Placeholder for the Conversion Utility to generate conversion log files for the current conversion project.
4. **/NativeLibary**. Placeholder for the Conversion Utility to generate the function table used to link C shared libraries to EGL.

The Conversion Utility generates the following artifacts:

Table 4-1. Artifacts (Documents) generated by the Conversion Utility

Artifact	File name	Description
Configuration file	***ProjectNameProjectType*Config.xml** For example, **MyDatabaseSchemaConfig.xml**	Contains project, database schema, directory, and file information. For more information, see "Configuration File" on page 4-4.
Conversion log	***ProjectNameProjectType*Log.txt** ***ProjectNameProjectType*Log.html**	Contains project information, including conversion errors and warnings, lists of dependent libraries, and the source file conversion status. This log is the primary source of information for unsuccessful conversions. Unless a fatal error stops the conversion process, this log is generated for both successful and unsuccessful conversions. For complete information, see "Correcting Conversion Errors" on page 4-6.
EGL Native Library	See "EGL Native Library" on page 4-9.	Contains the prototypes for the C functions encountered during the conversion of an I4GL shared library or application.
ERR file	***sourcefilename*.err** ***sourcefilenameForm*.err**	Identifies errors found during the conversion of I4GL source syntax to EGL.
Function Table	See "Function Table" on page 4-9.	Contains the names of all of the C functions encountered during the conversion of an I4GL application.

Table 4-1. Artifacts (Documents) generated by the Conversion Utility (continued)

Artifact	File name	Description
Manifest file	***ProjectNameProjectType*Manifest.xml**	Contains the project-specific information required to resolve dependencies. For a Database Schema Extraction project, the manifest file contains table/column information. For a Shared Library or I4GL Application project, the manifest file contains function calls and return types, global variables, form names, record types, package names, and other information. For more information, see "Manifest File" on page 4-5.

Note: The Conversion Utility generates file names as presented in the table above. If, during conversion, the Conversion Utility encounters these filenames in the Conversion Artifacts directory, it renames the existing file as ***filename*.bak.*num*** where *num* is a sequentially-increasing number starting with **1**. The Conversion Utility names the new file as described in Table 4-1 above.

Configuration File

During each of the three Conversion Utility stages, the utility compiles specific project, database schema, directory, and file information into the individual configuration file. Therefore, the configuration file stores all of the critical information about your conversion and should be referred to as necessary.

Each file is named after the utility stage that generates it. If, for example, you call your Database Schema Extraction project **MyDB**, the configuration file for that stage of the Conversion Utility is named **MyDBSchemaConfig.xml**.

If you prefer to convert your I4GL project outside the Rational environment, you can manually create a configuration file. A sample configuration file and the DTD are included in the product. If you manually create the configuration file, you must ensure that it conforms to the product configuration file DTD.

The sample configuration file is named **conversionsample.xml** and the DTD is named **conversionconfig.dtd**. Both are located in the following directory:

productinstallation/egl/eclipse/plugins/com.ibm.etools.i4gl.conversion_*version*/etc/dtd

Sample configuration files for the Database Schema Extraction project and the I4GL Application project are located in the following directory:

productinstallation/egl/eclipse/plugins/com.ibm.etools.i4gl.conversion_*version*/etc/examples/Configuration

Note: When you use the Conversion Utility Wizard, the Conversion Utility automatically generates a configuration file. The sample configuration files listed in the **etc** directory are template files that can be modified to your project specification. If you use the template files, you must ensure that you follow the DTD and create well-formed and valid XML documents. When a manually-edited configuration file that does not meet XML specifications is provided, the conversion terminates.

For an example of a configuration file template, see Appendix D, "Configuration File Templates." For an example of the DTD used for configuration and manifest files, see Appendix F, "DTD Examples."

Manifest File

A stage-specific manifest file is generated during all three conversion stages.

The manifest file generated during extraction of the database schema contains information about all of the tables, columns and data types of the selected database. This file is used by the conversion utility to resolve column names and data types between I4GL and EGL.

The manifest file generated during conversion of the I4GL shared library contains a list of I4GL and assumed C function calls used in the I4GL converting project. This file is used during the I4GL application conversion stage to validate all function calls used in dependent EGL Packages and I4GL source code.

Note: The Conversion Utility also generates a manifest file for I4GL application conversion projects. However, the application manifest file is generated for reference only, and provides a list of the technical details of the project.

For an example of a manifest file, see Appendix E, "Manifest File Examples."

Source File Conversion Mapping

During the conversion process, your I4GL program files were mapped to equivalent EGL project files. Table 4-2 provides the mapping between those files.

Table 4-2. How I4GL Files Convert into EGL Files

File and library type	I4GL file extension	EGL file extension
Source files	**.4gl**	**.egl**
Form specification files	**.per** For an example of how I4GL form files convert to EGL, see Appendix C, "I4GL Form Code to EGL Form Code Example."	**.egl** The constant string **Form** is added to all converted form specification file names; for example, an I4GL file named **someFile.per** is named **someFileForm.egl** in EGL.
Report files	**.4gl**	**.egl** **Note:** I4GL report design logic converts into a JasperReports **.jrxml** file. For more information on report file conversion, see "Understanding Report Conversion" on page 4-14.
Customized Error Message files	**.msg** or any other extension	**.properties**
Shared library created from I4GL source files	**.so** or other platform equivalent extension	EGL Packages
Build utility files	makefile	**.eglbld**, the EGL build descriptor file

Command Line Conversion: Importing Projects into the Workspace

The Conversion Utility Wizard automatically imports your converted project into the workspace within the **EGL Projects** category. If you used the command line option to convert your project, you must import your project into **EGL Projects** manually. You must import your Database Schema, Shared Library and Application projects individually.

To import your Database Schema, Shared Library, or Application project:

1. In the EGL perspective, right-click anywhere in the **Project Explorer** workspace.
2. From the menu, select **Import**.
3. From the **Import Wizard**, select **Existing Project into Workspace**. Select **Next**.
4. In the **Project Contents** text box of the **Import Project from File System** window, select **Browse**.
5. In the **Browse For Folder** window, select the EGL destination directory of your converted project. Select **OK** to exit the **Browse For Folder** window. Select **Finish** to exit the **Import Project from File System** window.

   Note: The Conversion Utility creates **.project**, **.eglPath**, and **.classpath** files in the EGL destination directory. These files are used by your Rational product to identify the project. If these files are missing, the project cannot be imported into the workspace.
6. Your selected directory now displays in the **Import Wizard** within **Project Contents**. The project name displays in **Project Name**.

   Note: If your Rational product does not locate the **.project** file in your selected directory, an error is returned. You must verify that your directory has that file before you can continue.
7. Click **Finish**. Your project should now be visible in the workspace.

Correcting Conversion Errors

The Conversion Utility generates a conversion log to help you understand the status of your conversion project and assist you in correcting any conversion errors. The Conversion Utility saves this conversion log to your designated artifacts directory **log** sub-folder. This file can be identified by the name ***ProjectNameProjectType*Log.html**, where ***ProjectType*** can be ***Schema***, ***Library*** or ***Application***. In addition to the **.html** file, a **.txt** file is generated.

Your conversion project can be classified as either PASSED or FAILED. The PASSED classification means that the conversion was successful and your EGL code is available for use.

The project is classified as FAILED when one or more EGL source files cannot be generated. Your project might fail for any of the following reasons:

- The Conversion Utility did not have permission to write to the identified EGL source directory or conversion artifacts directory.
- The XML configuration file or the dependent manifest file contain errors.
- For a Database Schema Extraction project, the JDBC driver terminates the process if you do not have connection permission to the database, provide an incorrect password, or if any other exception is noted.

- For a Shared Library or I4GL Application project, the conversion terminates if any source file listed in the configuration file does not have read permission or is missing.
- The I4GL syntax could not be converted into EGL syntax.

If your project fails during the validation stage, the conversion terminates. The validation stage includes validation of the XML configuration file, manifest file, source files, JDBC connection, and write and read permissions. If your project fails due to syntax errors in I4GL source files, the conversion continues, but the project is marked FAILED. If your project fails during any other stage, the conversion continues. In both of these cases, a conversion log is generated. A FAILED project must be converted from the beginning again.

After correcting all conversion errors, all FAILED projects must go through the conversion process again. If you use the Conversion Wizard for this process, you should select the **Open Project** option, which enables you to use your existing configuration file.

Note: Converting a FAILED project is not the same as reconverting a Shared Library. With a FAILED project, you must follow the three-stage Conversion Wizard path: Database Schema Extraction, Shared Libraries Conversion, and I4GL Application Conversion. When a shared library needs reconversion, you need only use the Shared Libraries path.

In addition, individual files can be classified as one of the following:

- PASSED. Confirms the file converted correctly; no user action necessary.
- ERROR. Identifies that the entire conversion failed and often explicitly identifies how it failed; you must reconvert the I4GL program using the Conversion Wizard.
- FIXME. Identifies that one or more stages of the conversion failed, but that conversion through the Conversion Wizard is not necessary. The information from this classification provides you with enough information to fix the problem with the converted files manually while in the EGL Perspective. Within your EGL code, each FIXME is identified with a FIXME tag, for example: **//FIXME:**. In addition, each FIXME and TODO includes a line and column reference to the original I4GL file.

  In addition to identifying a FIXME in the EGL code, each FIXME is listed within the EGL Tasks view. Use the information for each FIXME to correct your code.
- TODO. Identifies that the conversion passed, but with warnings; you can ignore the warnings or address each one manually while in the EGL Perspective. You should verify each of these warnings.

Conversion Log Contents

The conversion log contains the following information:

- **Conversion Status**. This section includes:
  - Project status
  - Conversion date
  - User name
  - Host
  - OS version
- **Project Details**. This section includes:
  - Project name

  - Directory information
  - Conversion type
  - Default Informix server instance
- **Conversion Artifacts**. This section includes the file names of the conversion artifacts.
- **I4GL Source File Conversion Summary**. This section lists the number of I4GL source files provided as input and the number of files converted into EGL as output. Occasionally, the number of EGL files exceeds the number of I4GL files.
- **Source File Conversion Details**. This section is present only for Shared Library and Application projects, and identifies the following:
  - Names of the input I4GL source files and the output EGL source files.
  - Conversion status of each file.
  - As appropriate, the location of the ERR file.
- **Exceptions**. This optional section identifies any conversion problems, including unsupported syntax warnings, fatal and non-fatal errors, and possible Conversion Utility internal functional failure exceptions. This section can record the reconversion requirements identified during your Application project manifest file consistency check.
- **Database Connection Details**. This section is present only for Database Schema Extraction projects, and identifies the following:
  - Server name
  - Database name
  - Table name
  - Corresponding EGL source file extracted for this table

Note: For both the database extraction and the source file conversion details section above, the **.html** version of the conversion log has hyperlinks to the input and output files and directories.

Once the conversion has completed, the log file is displayed in the **Report Tab**. If a default browser is located on the system, the Conversion Utility automatically opens the browser and the conversion log for viewing. You should review the contents of the log file soon after the conversion completes. You must understand the contents of the log file to correct any conversion errors or warnings.

If you use the command line mode, the conversion status is displayed in the console or shell window where the Conversion Utility is invoked. The log files are generated in the *EGLDestinationDirectory*/**ConversionArtifacts/log** directory. You must open this file for viewing.

Note: The conversion log also displays errors returned by your JDBC driver. If your driver provides an error description, that description is extracted from the driver and appears in the conversion log; if your driver does not provide an error description, only the error number provided by the JDBC driver displays in the conversion log.

In addition to the file information contained within the conversion log, the Conversion Utility also generates an ERR file to identify the I4GL syntax statements that did not convert successfully to EGL. With the ERR file information, you can correct the syntax errors in your I4GL code, and then reconvert your I4GL application. For **.4gl** files, the ERR file is named **sourcefile.err**. For **.per** files, the ERR file is named **sourcefileForm.err**.

For an example of a conversion log, see Appendix G, "Conversion Log Examples." To review a list of Conversion Utility error messages, see Error Messages.

Using C Shared Libraries with the EGL Program

If your I4GL program had shared libraries, one of your pre-conversion tasks was to review each library to identify whether it was compiled from **.4gl** source files, **.c** source files, or both. For a shared library compiled from both **.4gl** and **.c** source files, you segregated the **.c** files and compiled them into a C shared library. In addition, you might have identified some shared libraries that were originally compiled from only **.c** source files, or some pre-existing static libraries consisting of object files compiled from **.c** source files. This section explains how an application level shared library is created and linked with the variety of C libraries that you have identified.

During the conversion of an I4GL shared library or application, prototypes of the C functions encountered in that project were defined in an EGL native library created for that project. At the end of the I4GL application conversion, a function table identifying all the C functions was generated. These components are also explained in this section.

EGL Native Library

During the conversion of an I4GL shared library or application, prototypes of the C functions encountered in that conversion project are defined in an EGL native library created for that project. This library consists of function prototypes with no function body. The C functions are defined inside the C libraries, and the EGL native library is used to avoid validation errors in the EGL IDE. The name of the native library is **CExternals**, and it is created in the *EGLDestinationDirectory*/*ProjectName*/**EGLSource/CExternals** directory.

For complete information about the EGL native library, see the *Library part of type nativeLibrary* help topic.

Function Table

The function table is generated at the end of the I4GL application conversion stage. It is a C source file that includes the names of all C functions called from the EGL program. The function table is named **NativeFuncTab.c** and is located in the *EGLDestinationDirectory*/**ConversionArtifacts/NativeLibrary** directory.

In the following function table example, **c_fun1** and **c_fun2** are names of the C functions.

```
#include <stdio.h>
struct func_table {

      char *fun_name;
      int (*fptr)(int);
};

extern int c_fun1(int);
extern int c_fun2(int);
/* Similar prototypes for other functions */

struct func_table ftab[] =
            {
                 "c_fun1", c_fun1,
```

```
        "c_fun2", c_fun2,
        /* Similarly for other functions */
        "", NULL
    };
```

You must modify both the function table and the EGL application code if the following occur:

1. **The name of an existing C function is changed**. In addition to changing the function invocation statement in your EGL code, you must also change the following:
   - the name of the C function in the function table.
   - the function prototype must be defined in the EGL native library corresponding to each of the projects from which the function is invoked.
2. **A new C function is added**. Change both of the following:
   - the name of the new C function must be added to the function table
   - the function prototype defined in the EGL native library corresponding to each of the projects from which the function is involved.

Note: If two C functions share the same name, the actual function call depends upon the shared library linking sequence.

Creating the Application Level Shared Library

This section explains how to create an application level shared library and how to link it with the C libraries that have been identified or created during pre-conversion.

There are two parts to creating the application level shared library:

1. Download the EGL stack library and application object file from the IBM website to your computer.
2. Compile the function table and the appropriate platform-specific application object file into an application level shared library, and link this shared library with the appropriate platform-specific stack library and the C libraries identified or created during pre-conversion.

To download the EGL stack library and application object file:

1. The stack library is used to pass or return values between the EGL code and the C code. The pop and return APIs used in the C code are resolved in the stack library. The application object file acts as an interface between the EGL program and the C code. Note that both the stack library and the application object file are platform-specific components.
   a. Locate the EGL Support website.

      The URL for Rational Application Developer is www3.software.ibm.com/ibmdl/pub/software/rationalsdp/rad/60/redist

      The URL for Rational Web Developer is www3.software.ibm.com/ibmdl/pub/software/rationalsdp/rwd/60/redist

   b. Download the **EGLRuntimesV60IFix001.zip** file to your preferred directory.
   c. Unzip **EGLRuntimesV60IFix001.zip** to identify the following files:

      For the platform-specific stack libraries:
      - AIX 32-bit: **EGLRuntimes/Aix/bin/libstack.so**
      - AIX 64-bit: **EGLRuntimes/Aix64/bin/libstack.so**
      - Linux: **EGLRuntimes/Linux/bin/libstack.so**

- Win32:
  - **EGLRuntimes/Win32/bin/stack.dll**
  - **EGLRuntimes/Win32/bin/stack.lib**
- Solaris 32-bit: **EGLRuntimes/Solaris/bin/libstack.so**
- Solaris 64-bit: **EGLRuntimes/Solaris64/bin/libstack.so**
- HPUX 32-bit: **EGLRuntimes/HPUX/bin/libstack.sl**
- HPUX 64-bit: **EGLRuntimes/HPUX64/bin/libstack.sl**

For the platform-specific application object files:

- AIX: **EGLRuntimes/Aix/bin/application.o**
- AIX 64-bit: **EGLRuntimes/Aix64/bin/application.o**
- Linux: **EGLRuntimes/Linux/bin/application.o**
- Win32: **EGLRuntimes/Win32/bin/application.obj**
- Solaris 32-bit: **EGLRuntimes/Solaris/bin/application.o**
- Solaris 64-bit: **EGLRuntimes/Solaris64/bin/application.o**
- HPUX 32-bit: **EGLRuntimes/HPUX/bin/application.o**
- HPUX 64-bit: **EGLRuntimes/HPUX64/bin/application.o**

To create the application level shared library:

1. The following two artifacts must be compiled into the application level shared library and linked with the stack library and the libraries identified or created during pre-conversion:
   - the function table created by the Conversion Utility
   - the application object file
2. Compile the new shared library using the following example, where **NativeFuncTab.c** is the name of the function table, **lib1/lib2** and so forth are the names of the C libraries identified or created during pre-conversion and **lib_dir1/lib_dir2** and so forth are the respective directory locations of those libraries.
   - On AIX:

     ```
     cc -c NativeFuncTab.c
     ld -G -b32 -bexpall -bnoentry -brtl NativeFuncTab.o application.o
     -Lstack_lib_dir -lstack -Llib_dir1 -llib1 -Llib_dir2 -llib2 ... ... -o
     app_lib_name -lc
     ```

   - On Linux:

     ```
     cc -c NativeFuncTab.c
     gcc -shared NativeFuncTab.o application.o -Lstack_lib_dir -lstack
     -Llib_dir1 -llib1 -Llib_dir2 -llib2 ... ... -o app_lib_name
     ```

   - On Windows:

     ```
     cl /c NativeFuncTab.c
     link /DLL NativeFuncTab.obj application.obj /LIBPATH:stack_lib_dir
     /DEFAULTLIB:stack.lib /LIBPATH:lib_dir1 /DEFAULTLIB:lib1.lib
     /LIBPATH:lib_dir2 /DEFAULTLIB:lib2.lib ... ... /OUT:app_lib_name
     ```

When running the EGL application, this application level shared library is specified using the **vgj.defaultI4GLNativeLibrary** Java runtime property.

Properties Files

The properties of your converted EGL project are defined in one of three files, each of which have a **.properties** file extension. You must enter your I4GL environment variables into one of the following files as appropriate:

- **user.properties**, located in your home directory. This file identifies user-specific data like **user** and **password**.
- ***programname*.properties**, located in the CLASSPATH. Each converted I4GL project can have a file with this name. Properties and values in this file are customized to include values for all users.
- **rununit.properties**, located in the CLASSPATH. This file identifies information which is needed at runtime.

If a same-named property is set in any one of the files, the property in the file **user.properties** is used. The search order for properties is **user.properties**, ***programname*.properties**, and **rununit.properties**.

For a mapping of I4GL environment variables, EGL properties, and JDBC properties, see "Environment Variables" on page A-19.

The values bulleted below should be defined in at least one of the .**properties** files. The values in the example assume that the EGL application references a database named **stores7**, that it supports switching servers based on INFORMIXSERVER values, and that the server connection is enabled without a password through .**rhosts**, and that the application and the database server are not running on the same UNIX host.

- **INFORMIXSERVER**=myserver
- **DEFAULT_USER**=id
- **DEFAULT_PASSWORD**=pw
- **stores7@myserver**=jdbc:informix-sqli://host:port/database:INFORMIXSERVER=myserver;

If the application and the database server are running on the same host, **DEFAULT_USER** or **DEFAULT_PASSWORD** are not required.

If the application and the database server are operating on different UNIX hosts, you can specify the application host in the **.rhosts** file of the database server host. You do not need to specify the **DEFAULT_USER** and **DEFAULT_PASSWORD** values.

If the application does not require a value for **INFORMIXSERVER**, **INFORMIXSERVER** can be omitted as a separate value. If the application directly supplies userid and password values through the **CONNECT** statement, the **DEFAULT_USER** and **DEFAULT_PASSWORD** values can remain unspecified.

For more information on **.properties** files, see the *genProperties* and *Java runtime properties* help topics. For more information on specifying JDBC URLS for your IBM Informix DBMS server, review the IBM Informix JDBC documentation available at http://www.ibm.com/software/data/informix/pubs/library/.

Validating and Compiling Converted EGL Files

Your converted EGL source files must be validated and compiled. These steps are only required for I4GL Application and Shared Library projects and are not necessary for Database Schema Extraction projects. You must have the IBM Informix JDBC driver installed in your system before attempting this process.

Note: The Conversion Utility generates a template build descriptor file for each converted project. This file is located in the **EGLSource** directory and follows the naming convention of ***Projectname*.eglbld**.

To validate and compile the converted EGL source files:

1. From within the **Project Explorer** view, select and right-click *Projectname*.
2. From the menu provided, select **Properties**.
3. In the left box of the **Properties for** *Projectname* window, select **EGL Build Path**. In the tabbed **Projects** section, select all of listed dependent projects.
4. In the left box, select **EGL Default Build Descriptors**.
5. In the **Target system build descriptor** field of the **EGL Default Build Descriptors** section, select the down arrow. This displays all of the EGL build descriptor files available in the workspace.
6. Select your build descriptor, that is the file following the naming convention of ***Projectname*.eglbld**. Select **OK**. The automatic EGL build process initiates.
7. If your project requires database connectivity for testing, in the left box, select **Java Build Path**. Select the **Libraries** tab.
8. Select the **Add External JARs...** button. In the **JAR Selection** window, browse through the file system and locate the appropriate JDBC driver JAR files. Click **OK**. The selected JAR files appear in the **Libraries** tab.

Note: For an example of an EGL build descriptor, see Appendix H, "EGL Build Descriptor Example."

Generating EGL to Java

Before you generate your EGL files to Java, you must:

- Identify and resolve all TODO and FIXME messages. For more information, see"Correcting Conversion Errors" on page 4-6.
- Identify and resolve any validation errors. You cannot generate your EGL files to Java until all validation errors have been resolved.
- Create a **.properties** file for your JDBC connection information.
- As appropriate, add your database connection information to the ***Projectname*.eglbld** file.

To add your database connection information to the ***Projectname*.eglbld** file:

1. In the **Project Explorer** view, right-click on your *Projectname*.
2. Within the **EGLSource** directory, double-click on the ***Projectname*.eglbld** file.
3. The ***Projectname*.eglbld** file opens in a tabbed window. Check the **Show only specified options** box.
4. As appropriate, click on the existing option **Value** and enter your connection information for the following options:
   - sqlDB
   - sqlID
   - sqlPassword
   - sqlValidationConnectionURL
5. Close the window tab to accept the changes.

To generate EGL to Java:

1. In the **Project Explorer** view, right-click on your **Projectname**.

2. From the menu, select **Generate**.
3. The **Generating EGL Parts** window displays, and closes when generation completes.

To run the generated Java program with a database connection, you must create a **.property** file as described in "Properties Files" on page 4-11.

Your Java source files are located in the **JavaSource** folder in the project workspace. For more information on generating your EGL files, see the *Generating, preparing and running EGL output* help topic.

Understanding Error Message Conversion

The Conversion Utility converts I4GL error messages into Java properties files and places all of the converted message files into the following directory: ***EGLDestinationDirectory*/MessageSource**

A comparison between an I4GL error message file and the resultant Java properties format is seen in Table 4-3 below.

Table 4-3. Comparison of I4GL message to Java Properties file formats

I4GL message file format	EGL message file format
. message-number message-text	message-number=message-text

Message files in English (**en_us**) are converted into a Java .**properties** file with no language extension. Message files in other code pages generate a code page extension as part of the message file name and are appended with an extension of *_locale name*.**properties**. This extension differentiates locale-specific message files from each other. For example, a I4GL message file named **MyMessage.msg** file is converted to the following names:

In English: **MyMessage.properties**

In Chinese: **MyMessage_zh_TW.properties**

Understanding Report Conversion

EGL Reports use the functionality of JasperReports, an open source reporting library written in Java. Thus, during conversion to EGL, some I4GL code, functions, and files are changed.

During the conversion of your I4GL application, the following changes occurred to your I4GL report code:

- I4GL program files were converted into EGL program files. For example, the I4GL file named **myreport.4gl** is now the EGL file **myreport.egl**.
- A single I4GL REPORT function was converted into four EGL functions. For example, an I4GL report file named **myreport.4gl** with a **REPORT** function of **REPORT my4glReport(a, b, c)** was converted into an EGL report file named **myreport.egl** with the following functions calls:
  - my4glReport_START( )
  - my4glReport_OUTPUT(a, b, c)
  - my4glReport_FINISH( )
  - my4glReport_TERMINATE( )

Together, these four EGL functions act as a single report driver function in EGL.

- The business logic in an I4GL report was converted and moved into an EGL report handler file. The report handler is an EGL report component that provides the logic for handling events occurring during the filling of the report.

  If your I4GL report was named **my4glreport**, the EGL report handler file is named **my4glreport_handler.egl**.
- The presentation logic in an I4GL report is converted and moved into a JasperReports XML design file. This converted file contains the report layout template converted from your I4GL report.

  If your I4GL report is named **my4glreport**, the JasperReports XML file is named **my4glreport_XML.jrxml**. If your I4gl report contains multiple looping constructs containing output logic, each looping construct block is moved over to a subsequent sub-report. The filename of each subsequent report is identified with the word **SUB** and a sequence number. For example, the first subsequent report is named **my4glreport_XML_SUB1.jrxml.**

EGL Report Driver Functions

As mentioned above, during the I4GL to EGL conversion a single I4GL REPORT function is replaced with four EGL functions. Information on how each of those EGL functions map to I4GL syntax is included in Table 4-4 below.

Table 4-4. EGL Report Driver behavior

EGL Function	Corresponding I4GL statement	Behavior in EGL
*reportname*_**START( )**	START REPORT	While the converted EGL program is running, this function creates a temporary table in the default or active database of the EGL application. Parameters received by the I4GL report function are added as a column in the temporary table. In addition, with the exception of array variables, all global variables used by the I4GL report are added as a column to this temporary table.
*reportname*_**OUTPUT( )** *reportname*_**OUTPUT(a, b, c)**	OUTPUT TO REPORT	This function collects all of the arguments passed to it and any other global variables used by the I4GL report function for the report and adds their current value as a row to the temporary table. Typically, a number of calls must be made to pass all of the report data. **Note:** If the converting I4GL report does not receive any arguments or does not use global variables (other than array variables), this function inserts a placeholder row for the global variables into the temporary table.

Table 4-4. EGL Report Driver behavior (continued)

EGL Function	Corresponding I4GL statement	Behavior in EGL
reportname_FINISH( )	FINISH REPORT	This function does the following: • Initializes the report library • Passes a SELECT statement referring the temporary table to the EGL report library; the result set generated serves as the datasource for the JasperReports engine. • Uses the EGL API to fill the JasperReports. • Uses the EGL JasperReports text exporter to export the JasperReports to text format. After the report has been generated, this function drops the temporary table which was used to store the report data.
reportname_TERMINATE( )	TERMINATE REPORT	This function drops the temporary table and stop the report processing.

Note: When processing reports, the generated EGL report driver functions always assume that an active database is available. If an active database is not available, you must modify the report driver function code to use an EGL dynamic array of records to collect and pass the data to the JasperReports engine.

Since global array variables are not added to the temporary table, an I4GL report code using global array variables might not produce the expected results when converted to EGL. To achieve the expected results, you might need to modify your EGL report.

While I4GL reports were processed on a line by line basis, in EGL all report data were collected in a temporary table and then processed. Report data is only processed when all data is collected in the temporary table and **reportname_FINISH( )** is called. Thus, if there is a dependency between the data inside the report and I4GL report driving function, some reports do not produce correct results.

I4GL Report Sections

This section explains the conversion to EGL of the following four I4GL report sections:

- DEFINE
- OUTPUT
- ORDER BY
- FORMAT

I4GL DEFINE Section

In I4GL, the DEFINE section declares a data type for each formal argument in the REPORT prototype and for any additional local variables that can be referenced only from within the REPORT program block.

Parameter conversion: All parameters declared in the DEFINE section are converted to the XML design document as report fields, and then are declared as variables in the EGL report handler. In addition, parameters are converted to part

of the report driver functions as columns of the temporary table, so that their value at every iteration of an OUTPUT TO report call can be stored in the temporary table. The values stored in the temporary table are used as the data source for the report.

During conversion, the I4GL record type parameters are flattened to a single level in JasperReports fields. The following example is original I4GL code:

```
Define rec Record
             i, j Integer,
             k, l Integer,
             si Smallint,
End Record
```

This example is the equivalent code, flattened, in the XML design document:

```
rec_i
rec_j
rec_k
rec_l
rec_si
```

The Conversion Utility does not attempt to verify that the name of an I4GL record parameter is unique, that is, that another flattened or declared JasperReports field or variable does not have the same name. If, after conversion, you identify that two flattened or declared JasperReports fields or variables have the same name, you must change one of the names to ensure that each name is unique.

When converted to the EGL report handler, I4GL Report DEFINE section parameters are declared as normal variables. The EGL report handler uses the internal function **init** to process any business logic to initialize the locale variables with the report field values. Although in I4GL it was possible to assign values to the report fields or parameters received, it is not possible in EGL. Inconsistent expressions can result in a report with incorrect results. To achieve your preferred report results, you must manually modify your converted code.

The Conversion Utility converts I4GL variables to EGL Report Handler variables and columns on the temporary table. The Conversion Utility reviews each variable used by report statements and expressions, and treats global and non-array variables the same as report parameters. The Conversion Utility attempts to preserve the value of each variable at every iteration so that when EGL processes the datasource generated from the temporary table, the values received are the same as the I4GL values received. However, this method of passing values does not always conform to I4GL program logic and can produce incorrect results. To achieve your preferred report results, you must manually modify your converted code.

Note: With one exception, a global array variable used by an I4GL report remains global in nature. If your I4GL code has a global array variable with a value that changes with the iteration of the OUTPUT TO Report statement, the variable is not considered a global variable and it is not converted. To achieve your preferred report results, you must manually modify your converted code.

Local Variables Conversion: All DEFINE section local variables are converted to EGL report handler variables.

I4GL OUTPUT Section

In I4GL, the OUTPUT section specifies the destination and dimensions for output from the report and the page-eject sequence from the printer.

The original I4GL OUTPUT section clauses and how they convert to the XML design document are identified in Table 4-5 below.

Table 4-5. I4GL Output clause conversion

I4GL OUTPUT clause	Converts to	Conversion specifics
PAGE LENGTH	XML Design document	• If the PAGE LENGTH clause is present during conversion, the XML design document **pageHeight** page attribute is set to: `pageHeight = (Font height for the locale) * (value of PAGE LENGTH clause)` • If the PAGE LENGTH clause is not present during conversion, the value of PAGE LENGTH clause defaults to 66, the I4GL default page size. • Before setting the **pageHeight** value, the Conversion Utility verifies that the sum of the height of all the bands and **topMargin** and **bottomMargin** fit within the **pageHeight** value. As necessary, the Conversion Utility adjusts the **pageHeight** to ensure the correct fit.

BOTTOM MARGIN	XML Design document	• If the BOTTOM MARGIN clause is present during conversion, the XML design document **bottomMargin** page attribute is set to:
		`bottomMargin = (Font height for the locale) * (value of BOTTOM MARGIN)`
		• If the BOTTOM MARGIN clause is not present during conversion, the **bottomMargin** page attribute is set to:
		`bottomMargin = (Font height for the locale) * 3 (default I4GL value for BOTTOM MARGIN)`

LEFT MARGIN	XML Design document	• If the LEFT MARGIN clause is present during conversion, the JasperReports XML design **leftMargin** page attribute is set to:
		`leftMargin = (Font width for the locale) * (value of LEFT MARGIN)`
		• If the LEFT MARGIN clause is not present during conversion, the **leftMargin** page attribute is set to:
		`leftMargin = (Font width for the locale) * 5 (default I4GL value for LEFT MARGIN)`

RIGHT MARGIN	XML Design document	• If the RIGHT MARGIN clause is present during conversion, the XML design document **pageWidth** attribute is set to:
		`pageWidth = (Font width for the locale) * (value of right margin)`
		• If the RIGHT MARGIN clause is not present during conversion, the **pageWidth** attribute is set to:
		`pageWidth = (Font width for the locale) * 132 (default I4GL value for RIGHT MARGIN)`

Table 4-5. I4GL Output clause conversion (continued)

I4GL OUTPUT clause	Converts to	Conversion specifics

TOP MARGIN	XML Design document	• If the TOP MARGIN clause is present during conversion, the XML design document **topMargin** attribute is set to:
		`topMargin = (Font height for the locale) * (value of TOP MARGIN)`
		• If the TOP MARGIN clause is not present during conversion, the **topMargin** attribute is set to:
		`topMargin = (Font height for the locale) * 3 (default I4GL value for TOP MARGIN)`

REPORT TO	Does not convert directly	• If REPORT TO FILE *filename* or REPORT TO *filename* clauses are present in the I4GL report, the Conversion Utility saves the exported text output with the given *filename*.
TOP OF PAGE	Does not convert directly	In EGL, the character used by the EGL report text exporter.

I4GL ORDER BY Section

In I4GL, the ORDER BY section specifies a sort list for the input records and identifies the order in which groups are evaluated. The ORDER BY section converts to an XML design document (with the extension .**jrxml**) and to EGL Report driver functions.

Conversion of the ORDER BY to EGL provides:

- In the XML design document, group sections that follow the order established in the I4GL ORDER BY section.
- If ORDER EXTERNAL BY is not used in the I4GL Report, the temporary table maintains the order of ascending or descending columns as given, and records are retrieved from the table in that sorted order.

I4GL FORMAT Section

I4GL supports two types of FORMAT sections, both of which determine the appearance of the output from the report looks. The simplest report contains only the EVERY ROW keywords between the FORMAT and END REPORT keywords. A more complex I4GL report has FORMAT sections that contain control blocks (such as ON EVERY ROW or BEFORE GROUP OF) which contain statements to execute while the report is being processed. This section discusses both simple and complex I4GL reports and how their components convert to EGL and JasperReports.

Simple Reports: In I4GL, the FORMAT EVERY ROW statement produces a default report for all of the data passed to the I4GL Report function. No other format block statements are allowed with this statement.

In converting an I4GL simple report to the EGL equivalent, the Conversion Utility calculates the size of the report fields against the size of the field names, and uses whichever size is greater. If the size is less than the **columnWidth** attribute of the tag in the XML design document, the Conversion Utility converts the following to the XML design document:

- one page header section
- one detail section

In the page header section, a field name is placed one after the other using the size that is greater, field name or the field on the page header and the detail section.

If the size is greater than the **columnWidth** attribute in the XML design document, the Conversion Utility produces one detail section in the XML design document. The detail section shows report fields, and each field name is followed by the data in the field. The page size of a generated report is adjusted to fit the detail section.

I4GL simple report data values are converted to one of the following three EGL Report Handler function string values:

- **beforeDetailEvals**. This JasperReports default method is called before processing any detail band elements.
- **init**. This method copies all report field values to local variables.
- **getPrintString**. This method is used by the XML design document to retrieve report field values out of the print string array.

Complex Reports: In I4GL, control blocks define the structure of a report by specifying one or more statements to be executed when specific parts of the report are processed. If no data records are sent as output to the report, none of the statements in these blocks are executed. Essentially, I4GL complex reports have multiple subsections within the FORMAT section.

Although an I4GL report contains business and presentation logic in one central location, business and presentation logic are separate in EGL reports. The Conversion Utility converts I4GL business logic into an EGL report handler document and converts the I4GL presentation logic into an XML design document.

All of the I4GL FORMAT subsection data presentation statements are converted or moved to a corresponding JasperReports band, a generic report section. The mapping between the original I4GL FORMAT subsection data presentation statement and the converted band is provided in Table 4-6 below.

Table 4-6. Mapping of I4GL Data Presentation statements to JasperReports bands

Data presentation statement	Band
First Page Header	title
Page Header	pageHeader
Before Group Of *variable*	groupHeader (of a group section for the given *variable*)
On Every Row	detail
After Group Of *variable*	groupFooter (of a group section for the given *variable*)
Page Trailer	pageFooter
On Last Row	summary

The mapping between the original I4GL FORMAT subsection data and the EGL Report Handler method is provided in Table 4-7 below.

Note: The **JasperReports Method** column heading identifies if a method is also a default report event handling method.

Table 4-7. Mapping of I4GL FORMAT subsections to EGL and JasperReports

I4GL Report FORMAT sub-section	EGL Report Handler Method	JasperReports Method
First Page Header	firstPageHeader	No

Table 4-7. Mapping of I4GL FORMAT subsections to EGL and JasperReports (continued)

I4GL Report FORMAT sub-section	EGL Report Handler Method	JasperReports Method
Page Header	pageHeader	No
Before Group Of *variable*	beforeGroupOf *variable*	No
On Every Row	beforeDetailEval	Yes
After Group Of *variable*	afterGroupOf	No
Page Trailer	pageTrailer	No
On Last Row	onLastRow	No

In addition to the methods listed in the table above, three additional methods are defined in the EGL Report Handler:

- **init**. This Report Handler method is called from other methods to initialize the local report handler variables with report field values. In addition, this method initializes print flags and print strings associated with the designated band.
- **getPrintString**. This method is called from the XML design document to return values to be displayed in the text field.
- **getPrintFlag**. This method is called from the XML design document to determine the print status of a static text or text field. This method returns the values of 1 (true) or 0 (false).

The following I4GL Report statements do not convert to EGL or JasperReports; however, to maintain syntactically correct code, the Conversion Utility generates redundant code in the report handler for these statements:

- EXIT REPORT
- NEED
- PAUSE

The following I4GL elements convert to EGL on a conditional basis:

- **SKIP TO TOP OF PAGE**. This statement converts only if it occurs within the FORMAT section of a BEFORE GROUP OF sub-section. If **SKIP TO TOP OF PAGE** occurs in any other instance, the statement will not convert, and as a consequence, your report may produce data on an incorrect page.
- **LINENO**. EGL does not support conversion of the I4GL LINENO operator. However, the Conversion Utility does generate EGL report handler code which simulates the LINENO operation by incrementing the local report handler variable for every print statement executed through the business logic.

Note: If your converted report contains the elements listed above, you should review your converted code and correct it as necessary.

I4GL PRINT Statement: In I4GL, the PRINT statement produces output from a report definition. The Conversion Utility analyzes and collects data from all I4GL report PRINT statements and converts most of the report presentation layout to the XML design document. The Conversion Utility analyzes and collects the following PRINT statement data:

1. The number of PRINT statements in a report.
2. The number of looping constructs containing PRINT statements, and the number of PRINT statements in each looping construct.

3. The number of expressions in a PRINT statement and the attributes of each expression.
4. The size of each PRINT statement. Size is calculated by adding the size of all data returned by the PRINT statement expressions. If the size of the PRINT statement is larger than the **columnWidth**, the **pageWidth** and **columnWidth** are adjusted to fit all the print fields in the page.

Once the information above is collected, the Conversion Utility defines the following for the EGL Report Handler or the XML design document:

1. The design band height for every I4GL FORMAT sub-section.
2. The number of EGL sub-reports required, and the structure of the static text and text fields for each subreport.
3. The number of record arrays, and the structure necessary to hold subreport data in the EGL Report Handler.
4. The number of print flags and print strings required in the EGL Report Handler.
5. The number of text elements and whether they fit into a static text or a text field.
6. Whether the given text field is right justified, and therefore, a numeric, field, or left justified, and therefore, a non-numeric, field.

For every PRINT statement that does not occur in a loop, the Conversion Utility:

1. Adjusts the coordinates for the static text or the text field **.xml** tags and generates the correct coordinates in the XML design document.
2. Associates one array index in **egl4gl_printFlag** with the given PRINT statement.
3. Generates code in the XML design document to get the associated **egl4gl_printFlag** value returned from EGL report handler program for **the printWhenExpression** tag of every static text or text field in the XML design document to determine if that element should appear in the report.
4. Generates the EGL report handler code that sets the appropriate value for associated **egl4gl_printflag**.
5. Associates one array index in **egl4gl_printString** with every expression in the PRINT statement.
6. Adds code to the text fields in the XML design document to return string values to be placed in the report from EGL report handler for a given array index of **egl4gl_printString**.
7. Generates code in the EGL report handler to populate the **egl4gl_printString** array indexes with output string values taking into account PRINT attributes.
8. Increments the **egl4gl_lineNumber** variable by one; this increment parallels the I4GL LINENO operator.

I4GL PRINT Statement Expressions: I4GL PRINT statement expressions return one or more values that can be displayed as printable characters. The Conversion Utility analyzes each expression and converts it into either a staticText field or into one or more text fields in the XML design document. In addition, these print expressions are used to determine the vertical placement and the size or width of staticText or text fields in the XML design document. The width is determined by the following parameters:

- The attribute of the print expression.
- The data size of the variable making the PRINT expression.

Note: When the I4GL SPACE or SPACES AND COLUMN operator have an integer expression instead of an integer literal, the Conversion Utility sets the value of the expression to 1, the default. You might need to adjust the report design and code to achieve the correct report layout.

I4GL PRINT Statement in looping constructs: Like other PRINT statements, PRINT statements in looping constructs are converted to the equivalent number of print expressions, each of which is added to the XML design document subreport. In the EGL Report Handler, one record array is declared to accumulate all of the data generated for output. For every PRINT statement with a looping construct, the Conversion Utility:

1. Identifies all PRINT statement expressions.
2. Identifies the nesting structure of the loop.
3. Declares a record array structure which matches the nesting structure, print strings, and the print flags for the PRINT statement.
4. Adds code in the EGL report handler to declare a record array, and populates the record array fields with print string and print flag data.
5. Adds a subreport tag in the XML design document.
6. Generates another XML design document with one detail section containing all of the fields from the record array. This detail section represents the template of the PRINT statement as identified in the looping structure.

Note: The Conversion Utility attempts to convert the I4GL LABEL and GOTO looping structure statements into a looping structure. However, identification and conversion of the LABEL and GOTO looping structures is not always successful, and you might need to adjust the report design and code to achieve the correct results.

I4GL PRINT statement terminating with a semicolon: An I4GL PRINT statement terminating with a semicolon indicates that the next PRINT statement continues on the same line. The Conversion Utility identifies each PRINT statement terminating with a semi-colon and creates an XML design document with a similar layout. However, PRINT statements terminating with a semicolon are not successful when the following conditions occur:

1. An IF condition block is implemented before the next PRINT statement.
2. The PRINT statement is followed by a looping construct.
3. The PRINT statement is the last PRINT statement on the I4GL FORMAT section.

I4GL Report Operators: Details about the conversion of I4GL report operators is provided in Table 4-8 below.

Table 4-8. Conversion of I4GL Report Operators

I4GL Report Operator	Conversion specifics
CLIPPED	If a PRINT statement expression is followed by a PRINT statement expression with an attribute of CLIPPED, in the XML design document, the second PRINT statement is placed in the same text field as the first statement. If the Conversion Utility identifies a COLUMN print expression or an end of print statement, the placement of two print expressions into one text field terminates. The size of the text field into which the PRINT statement is placed is the sum of all PRINT fields placed in the text field.
WORDWRAP	To convert WORDWRAP, the Conversion Utility sets the XML design document textField tag **isStretchWithOverflow** attribute to *true*.

Table 4-8. Conversion of I4GL Report Operators (continued)

I4GL Report Operator	Conversion specifics
USING	When creating the output **egl4gl_printString** in the EGL report handler program, the EGL formatting functions are called to the return string, which is formatted similarly to the I4GL USING attribute.
SPACE or SPACES	The Conversion Utility analyzes the SPACE or SPACES operators as quoted blank strings, and converts the operators into a staticText field or a textField in the Jasper XML design document. If the number of SPACES is defined by a non-integer literal, the Conversion Utility assumes a SPACE of 1.
PAGENO	The Conversion Utility converts the PAGENO operator to the JasperReport PAGE_NUMBER variable. The EGL Report Handler maintains the methods to access the PAGE_NUMBER variable.
LINENO	There is no direct conversion mapping for the I4GL LINENO operator. Instead, converted I4GL reports use a user-defined variable **egl4gl_lineNumber** to simulate LINENO. **Note:** Because of the inability to maintain the correct value for LINENO, you might need to adjust the report code to achieve the expected report outcome.
FILE	This operator does not convert to EGL. To achieve the results generated by FILE, you must manually correct your converted code.

I4GL Aggregate Report Functions: In I4GL, the **SUM**, **MAX**, **MIN**, **COUNT**, **PERCENT**, and **AVG** aggregate report functions were used to summarize data from several records in a report. During the conversion of your I4GL report, each usage of an aggregate report function is converted to one or more XML design report variables and to one or more report handler functions. To ensure that the calculation can be handled in the report handler function, the XML design report variable calculation type is set to SYSTEM.

This section provides information on how each I4GL aggregate report function converts to EGL. For all of the examples below, in the I4GL statements provided, *variable* is a valid I4GL report identifier.

SUM: The I4GL **SUM** aggregate function generates one JasperReports XML design variable and one report handler function. For example, the single I4GL statement

```
PRINT SUM(variable)
```

converts to the following two elements in EGL:

1. the JasperReports XML design variable

   ```
   variable_SUM_number
   ```

2. the EGL report handler function, which calculates the JasperReports XML design variable

   ```
   Function update variable_SUM_number
   ```

In the EGL Report handler function above, *number* is a running number which distinguishes each occurrence of the I4GL **SUM** aggregate function, and is used to create a unique variable name.

Note: The variable name will change if it is a member of the record.

During conversion, code is generated to call a report design variable using the **getReportVariableValue()** report library API. Therefore, the **PRINT SUM(***variable***)** statement above appears in EGL as:

```
egl4gl_printString[1] = getReportVariableValue("<variable>_SUM_<number>");
```

MAX: The I4GL **MAX** aggregate function generates one XML design variable and one report handler function. For example, the single I4GL statement

```
PRINT MAX(variable)
```

converts to the following two elements in EGL:

1. the XML design variable

   ```
   variable_MAX_number
   ```

2. the EGL report handler function, which identifies the maximum value of the variable

   ```
   Function update variable_MAX_number
   ```

In the EGL report handler function above, *number* is a running number that distinguishes each occurrence the I4GL **MAX** aggregate function and is used to create a unique variable name.

Note: The variable name will change if it is a member of the record.

During conversion, code is generated to call a XML design report variable using the **getReportVariableValue()** report library API. Therefore, the I4GL **PRINT MAX(***variable*) statement above appears in EGL as:

```
egl4gl_printString[1] = getReportVariableValue("variable_MAX_number");
```

MIN: The I4GL **MIN** aggregate function generates one XML design variable and one report handler function. For example, the single I4GL statement

```
PRINT MIN(variable)
```

converts to the following two elements in EGL:

1. the JasperReports XML design variable
   - `variable_MIN_number`
2. the EGL report handler function, which identifies the minimum value of the JasperReports variable
   - `Function update variable_MIN_number`

In the EGL report handler function above, *number* is a running number that distinguishes each occurrence the I4GL **MIN** aggregate function, and is used to create a unique variable name.

Note: The variable name will change if it is a member of the record.

During conversion, code is generated to call a JasperReports XML design report variable using the **getReportVariableValue()** report library API. Therefore, the I4GL **PRINT MIN(***variable***)** statement above appears in EGL as:

```
egl4gl_printString[1] = getReportVariableValue("variable_MIN_number");
```

COUNT: The I4GL **COUNT** aggregate function generates one JasperReports XML design variable and one report handler function. For example, the single I4GL statement

```
PRINT COUNT(*)
```

converts to the following two elements in EGL:

1. the JasperReports XML design variable

- COUNT_*number*

2. the EGL report handler function, which identifies the maximum value of the JasperReports variable
   - Function update *variable*_COUNT_*number*

In the EGL report handler function above, *number* is a running number which distinguishes each occurrence the I4GL **COUNT** aggregate function, and is used to create a unique *variable* name.

Note: The variable name will change if it is a member of the record.

During conversion, code is generated to call an XML design report variable using the **getReportVariableValue()** report library API. Therefore, the I4GL **PRINT COUNT(*)** statement above, appears in EGL as:

```
egl4gl_printString[1] = getReportVariableValue("variable_COUNT_number");
```

PERCENT: The I4GL **PERCENT** aggregate function generates two JasperReports XML design variables and two report handler functions. For example, the single I4GL statement

```
PRINT PERCENT(variable)
```

converts to the following four elements in EGL:

1. two JasperReports XML design variables
   - *variable*_PERCENT_*number*_PART
   - *variable*_PERCENT_*number*_WHOLE
2. two EGL report handler functions, each of which identify the maximum value of the JasperReports variable

- Function update *variable*_PERCENT_*number*_PART
- Function update *variable*_PERCENT_*number*_WHOLE

For both XML design variables and the EGL report handler functions, the following applies:

- In Function update *variable*_PERCENT_*number*_PART
  - *variable* accumulates the value of all of the variables for which the conditional clause is satisfied.
- In Function update *variable*_PERCENT_*number*_WHOLE
  - *variable* accumulates the value of all the variables in the record set.

For both EGL report handler functions above, *number* is a running number which distinguishes each occurrence of the I4GL **PERCENT** aggregate function, and is used to create a unique *variable* name.

Note: The variable name will change if it is a member of the record.

During conversion, code is generated to call an XML design report variable using the **getReportVariableValue()** report library API. Therefore, the I4GL **PRINT PERCENT(*variable*)** statement above appears in EGL as:

```
egl4gl_printString[1] =
getReportVariableValue("variable_PERCENT_number_PART") /
getReportVariableValue("variable_PERCENT_number_WHOLE") * 100;
```

AVG: The I4GL **AVG** aggregate function generates two JasperReports XML design variables and two report handler functions. For example, the single I4GL statement

`PRINT AVG(variable)`

converts to the following four elements in EGL:

1. two JasperReports XML design variables
   - *variable*_AVG_*number*_SUM
   - *variable*_AVG_*number*_COUNT
2. two EGL report handler functions, each of which identify the maximum value of the JasperReports variable

- `Function update variable_AVG_number_SUM`
- `Function update variable_AVG_number_COUNT`

For both JasperReports XML design variables and the EGL report handler functions, the following applies:

- In `Function update variable_AVG_number_SUM`
  - *variable* accumulates the value of all of the variables for which the conditional clause is satisfied.
- In `Function update variable_AVG_number_COUNT`
  - *variable* accumulates the value of all the variables in the record set.

For both EGL report handler functions above, *number* is a running number which distinguishes each occurrence of the I4GL **AVG** aggregate function, and is used to create a unique *variable* name.

Note: The variable name will change if it is a member of the record.

During conversion, code is generated to call an XML design report variable using the **getReportVariableValue()** report library API. Therefore, the I4GL **PRINT AVG(***variable***)** statement above appears in EGL as:

```
egl4gl_printString[1] =
getReportVariableValue("variable_AVG_number_SUM") /
getReportVariableValue("variable_AVG_number_COUNT");
```

For information on how to implement your EGL reports or how to create new reports, see the *EGL Reports* topics in the information center.

Understanding your EGL Projects, Packages and Files

An EGL project contains zero to many source folders, each of which contains zero to many packages, each of which contains zero to many files. Each file contains zero to many parts.

Note: The following information is also presented in your Rational product information center, and provides cross-references to other information center topics.

EGL Project

An EGL project is characterized by a set of properties. In the context of an EGL project, EGL automatically performs validation and resolves part references when you perform certain tasks; for example, when you save an EGL file or build file. In addition, if you are working with page handler parts, EGL automatically generates output only if:

- You set the automatic build process after selecting these options: **Window > Preferences > Workbench > Perform build automatically on resource modification**.
- You established a default build descriptor as a preference or property

An EGL project is formed by selecting **EGL** or **EGL Web** as the project type when you create a new project. You assign properties while working through the steps of project creation. To begin modifying your choices after you have completed those steps, right-click the project name and when a context menu is displayed, click **Properties**.

The EGL properties are described in the following sections.

EGL source folder

One or more project folders that are the roots for the project's packages, each of which is a set of subdirectories. A source folder is useful for keeping EGL source separate from Java files and for keeping EGL source files out of the Web deployment directories. You should specify EGL source folders in all cases; but if a source folder is not specified, the only source folder is the project directory.

The value of this property is stored in a file named **.eglpath** in the project directory and is saved in the repository (if any) that you use to store EGL files.

The EGL project wizards each create one source folder named EGLSource.

EGL build path

The list of projects that are searched for any part that is not found in the current project.

The value of this property is stored in a file named **.eglpath** in the project directory and is saved in the repository (if any) that you use to store EGL files.

In the following example of an **.eglpath** file, EGLSource is a source folder in the current project, and **AnotherProject** is a project in the EGL path:

```
<?xml version="1.0" encoding="UTF-8"?>
  <eglpath>
    <eglpathentry kind="src" path="EGLSource"/>
    <eglpathentry kind="src" path="\AnotherProject"/>
  </eglpath>
```

The source folders for **AnotherProject** are determined from the **.eglpath** file in that project.

Default build descriptors

The build descriptors that allow you to generate output quickly, as described in **Generation in the workbench** information center topic.

Package

A package is a named collection of related source parts.

They are not in use when you create build parts.

By convention, you achieve uniqueness in package names by making the initial part of the package name an inversion of your organization's Internet domain name. For example, the IBM Corporation domain name is ibm.com, and the EGL packages begin with "com.ibm". By using this convention, you gain some

assurance that the names of Web programs developed by your organization will not duplicate the names of programs developed by another organization and can be installed on the same server without possibility of a name collision.

The folders of a given package are identified by the package name, which is a sequence of identifiers separated by periods (.), as in this example:

```
com.mycom.mypack
```

Each identifier corresponds to a subfolder under an EGL source folder. The directory structure for **com.mycom.mypack**, for example, is *\com\mycom\mypack*, and the source files are stored in the bottom-most folder; in this case, in **mypack**. If the workspace is *c:\myWorkspace*, if the project is *new.project*, and if the source folder is EGLSource, the path for that package is as follows:

```
c:\myWorkspace\new.project\EGLSource\com\mycom\mypack
```

The parts in an EGL file all belong to the same package. The file's package statement, if any, specifies the name of that package. If you do not specify a package statement, the parts are stored directly in the source folder and are said to be in the **default package**. You should always specify a package statement because files in the default package cannot be shared by parts in other packages or projects.

Two parts with the same identifier may not be defined in the same package.

Note: You should not use the same package name under different projects or different folders.

The package for generated Java output is the same as the EGL file package in most cases.

EGL Files

Each EGL file belongs to one of these categories:

Source file

An EGL source file (extension .**egl**) contains logic, data, and user interface parts and is written in EGL source format. Each of the following *generatable parts* can be transformed into a compilable unit:

- DataTable
- FormGroup
- Handler (the basis of a report handler)
- Library
- PageHandler
- Program

Other parts are called subparts.

An EGL source file can include zero to many subparts but can include no more than one generatable part. The generatable part (if any) must be at the top level of the file and must have the same name as the file.

Build File

An EGL build file (extension **.eglbld**) contains any number of build parts and is written in Extensible Markup Language (XML), in EGL build-file format. You can review the related DTD, which is in the following directory:

```
installationDir\egl\eclipse\plugins\com.ibm.etools.egl_version
```

Recommendations

This section gives recommendations for setting up your development projects.

For build descriptors

Project teams should appoint one person as a build-descriptor developer. The tasks for that person are as follows:

- Create the build descriptors for the source-code developers
- Put those build descriptors in a project separate from the source code projects; and make that separate project available in the repository or by some other means
- Ask the source-code developers to set the property **default build descriptors** in their projects, so that the property references the appropriate build descriptors
- If a small subset of the build descriptor options (such as for user ID and password) varies from one source-code developer to the next, ask each source-code developer to do as follows:
  - Code a personal build descriptor that uses the option **nextBuildDescriptor** to point to a group build descriptor
  - Ask the source-code developers to set the property **default build descriptors** in their files, folders, or packages, so that the property references the personal build descriptor. They do not specify the property at the project level because the project-level property is under repository control, along with other project information.

For additional information, see *Build descriptor part.*

For packages

For packages, recommendations are as follows:

- Do not use the same package name in different projects or source directories
- Do not use the default package

Part assignment

For parts, many of the recommendations refer to good practices, not hard requirements. Fulfill even the optional recommendations unless you have good reason to do otherwise:

- A *requirement* is that you put JSPs in the same project as their associated page handlers.
- If a subpart (like a record part) is used only by one program, library, or page handler, place the subpart in the same file as the part.
- If a part is referenced from different files in the same package, put that part in a separate file in the package.
- If a part is shared across packages in a single project, place that part in a separate package in that project.
- Put code for completely unrelated applications in different projects. The project is the unit for transferring code between your local directory structure and the repository. Design project structure so that developers can minimize the amount of code they must have loaded into their development system.
- Name projects, packages, and files in a way that reflects the use of the parts they contain.
- If your process emphasizes code ownership by a developer, do not assign parts for different owners to the same file.

- Assign parts to packages with a clear understanding of the purpose of the package; and group those parts by the closeness of the relationship between them. The following distinction is important:
  - Moving a part from file to file in the same package does not require that you change import statements in other files.
  - Moving a part from one package to another might require an import statement to be added or changed in every file that references the moved part.

The Information Center Help System and EGL Tutorial

The information center contains extensive documentation on how to use both EGL and the full range of product features. Access the information center from the main menu by selecting **Help > Rational Help**.

The EGL Tutorial teaches you how to build a simple dynamic Web site using EGL. The tutorial is accessible from the main menu by selecting **Help > Tutorials Gallery** and then selecting **Do and Learn** from the left pane of the gallery. The tutorial assists you in learning how to:

- Set up and configure an EGL project
- Create EGL source code
- Create two simple Web pages that access data in a relational database
- Pass a parameter from one Web page to another
- Configure a Web application server and run an application on that server

Exercises in the tutorial include:

- Setting up EGL
- Creating and configuring the EGL project
- Starting and configuring WebSphere Application Server v6.0
- Creating EGL data parts
- Creating an EGL library
- Creating a Web page
- Adding data to the page
- Linking to another page
- Creating an update page

Chapter 5. Reconversion Process and Tasks

In This Chapter

This chapter describes when and how to reconvert your I4GL shared libraries.

When to Reconvert Your I4GL Shared Libraries

During the conversion of your I4GL shared libraries, a library-specific manifest file is generated in the following directory: *EGLDestinationDirectory*/**ConversionArtifacts/manifest**. This manifest file has the following naming convention: ***Projectname Projecttype*Manifest.xml**. For example, for a Shared Library project named **MyLibrary**, the manifest file is named **MyLibraryLibraryManifest.xml**. The manifest file identifies the functions, global variables, and forms used in each library.

During an I4GL application conversion, the Conversion Utility compares the function call references in the source files with the manifest file from the dependent shared libraries. If the function call references in the source files and the function references in the dependent manifest file are not consistent, both must be reconciled. The dependent manifest file must be updated with the correct function call references, and the shared library project must be reconverted using the updated manifest file.

How to Reconvert Your I4GL Shared Libraries

During an I4GL application conversion, the Conversion Utility reconciles the dependent shared library manifest file. This manifest file should be used for the dependent shared library reconversion.

When Conversion Utility writes this reconciled manifest file, it backs up the original dependent manifest file with the filename of ***filename*.bak.*num***, and replaces the given manifest file with the reconciled manifest file. You must ensure that the correct manifest file is used for the dependent shared library reconversion.

You can reconvert your shared libraries in two modes:

- Through the conversion wizard
- Using the command line

Note: Since using the command line mode for the initial conversion process requires users to manually create a configuration file, you might find it easier to use the command line option only for reconversion, when an existing configuration file can be used.

Conversion Wizard Reconversion

To reconvert your shared library:

1. From within the EGL Perspective in your Rational product, select **File > New > Other > Informix 4GL to EGL Conversion > Shared Library Conversion Wizard.**
2. In the **I4GL Shared Library Conversion Project** screen, insert the following information:
   - **Project Details**.
     - **Reconversion Project**. Select this option and provide the location of the existing configuration file and the reconciled manifest file. You can browse to locate the file.
   - **Conversion Artifacts**. The Conversion Utility generates a number of artifacts related to the conversion, including configuration, manifest and conversion log files. By default, the conversion artifacts are located in the EGL destination directory. You can also designate to create the conversion artifacts in an external directory. You can browse to an appropriate directory, but cannot create a new directory at this time.
3. **Conversion Project Details**. Review the project details, including the contents of the configuration file and manifest file.
4. Click **Finish** to launch the reconversion.

Command Line Reconversion

You can reconvert your shared library from the command line.

To use the command line to reconvert a shared library:

1. Open a command line
2. At the prompt, enter the following, where *configurationfile* is the name of the configuration file used for the earlier conversion and *manifestfile* is the name of the updated manifest file created by the Conversion Utility:
   a. For Windows:

      ```
      e4gl.bat configurationfile -reconversion manifestfile
      ```

   b. For Linux:

      ```
      e4gl.sh configurationfile -reconversion manifestfile
      ```

Note: For information on how to use the command line utility to convert an I4GL application, see "Conversion Utility Command Line Mode" on page 3-7.

Reasons and Workarounds for Unsuccessful Reconversions

The shared library reconversion may fail for the reasons described in Table 5-1 below.

Table 5-1. Reasons for Shared Library Reconversion Failure

Reason for Reconversion Failure	Workaround
Configuration file corruption	Edit and correct the configuration file manually, or use the Conversion Utility Wizard to regenerate a new, uncorrupted configuration file.

Table 5-1. Reasons for Shared Library Reconversion Failure (continued)

Reason for Reconversion Failure	**Workaround**
Manifest file corruption	Use the Conversion Utility Wizard to convert the shared library as a **New Project** and produce a new manifest file. Use the new manifest file to convert the application project, and then use the Application project reconciled manifest file to reconvert the shared library.
Inadequate disk space	Create enough disk space.
Absence of file system write permissions	Obtain write permissions for the file system.
Inability to read the listed I4GL source files	Verify the name and path of each source file.

Appendix A. I4GL to EGL Syntax Mapping

In This Appendix

This appendix identifes the correspondence between I4GL and EGL syntax constructs, and includes the following tables:

Data Types

All EGL primitive types are described and their behaviors explained in the *Primitive Types* online help topic.

Note: RECORD definitions are appended to the end of the generated file and get names based on the file and function containing the definition and the first variable of that type.

Table A-1. How 4GL data types map to EGL primitive types

I4GL	EGL
ARRAY OF: DEFINE *x* array[10] of integer DEFINE *myrec* ARRAY[10,2] of RECORD INT *x*, INT *y* END RECORD	x int[10]; Myrec recordtype_filename_myrec[10][2]; Record recordtype_filename_myrec type SqlRecord x Int; y Int; End
BYTE	BLOB
BIGINT	BIGINT
CHAR (*size*) CHAR	UNICODE (SIZE) UNICODE (1)
CHARACTER	UNICODE (1)
DATE	DATE
DATETIME YEAR TO FRACTION(3) DATETIME YEAR TO HOUR	Timestamp ("yyyyMMddhhmmssfff"); Timestamp ("yyyyMMddhh");
DEC	DECIMAL
DECIMAL(*p, s*)	DECIMAL(p, s)
DECIMAL(*p*) NUMERIC(*p*)	DECIMAL(p) for ANSI database DOUBLE for non-ANSI database FLOAT for non-ANSI database • ANSI database determined from value found in the Database Schema Manifest File.
DOUBLE PRECISION	FLOAT
DYNAMIC ARRAY DEFINE myA DYNAMIC ARRAY WITH 3 DIMENSIONS of xxxx	myA xxx[][][];
FLOAT	FLOAT
INT INTEGER	INT INT
INT8	BIGINT
INTERVAL YEAR(9) TO MONTH INTERVAL DAY(7) TO FRACTION(3)	Interval("yyyyyyyyyMM" /* YEAR(9) TO MONTH */) Interval("ddddddddhhmmssfff")
MONEY	MONEY
NCHAR(*size*)	UNICODE(size)
NVARCHAR(*size*)	String **Note:** The maximum size of the VARCHAR variable is not represented in EGL.
REAL	SMALLFLOAT

Table A-1. How 4GL data types map to EGL primitive types (continued)

I4GL	EGL
RECORD DEFINE *Var* LIKE *table.code*; DEFINE *Recvar* LIKE *table*.*;	/* RECORD is defined by the EGL file during the database schema extraction project. */ Package IfmxDatabaseSchema.Svrname.Dbname; Dataitem like_tabname_code UNICODE(10) { ... properties ... }; Dataitem like_tabname_quantity INT { ... properties ... }; Record rec_like_tabname type SQLRecord { tablenames=["table"]} code like_tabname_code { column="code"} quantity like_tabname_quantity { columnname="code"} end Import IfmxDatabaseSchema.Svrname.Dbname.*; Var like_table_code; Recvar rec_like_table;
SMALLFLOAT	SMALLFLOAT
SMALLINT	SMALLINT
TEXT	CLOB
VARCHAR(*size*)	String **Note:** The maximum size of the VARCHAR variable is not represented in EGL.

Special Data Casting

Table A-2. Special Data Casting

I4GL	EGL
DEFINE *i* INTEGER, *d* DATE; LET *i* = *d*; LET *d* = *I*;	*i = DateToDays(d)*; *d = DaysToDate(i)*;
DEFINE *c* CHAR(80), *d* DATE; LET *c* = *d*;	*C = d*; /* system creates formatted string for d based on locale */
DEFINE *ds* DATETIME YEAR TO SEC; DISPLAY "now is", *ds*;	*displayLineMode ("now is "+ds)*; /* system creates formatted string for ds based on locale */
DEFINE *inv* INTERVAL DAY TO SEC; DISPLAY "remaining time is", *inv*;	*displayLineMode ("remaining time is "+inv)*; /* system creates formatted string for inv based on locale */

Definition and Declaration Statements

Table A-3. How 4GL Definition and Declaration Statements map to EGL

I4GL	EGL
DEFINE *x* INTEGER	x int

Table A-3. How 4GL Definition and Declaration Statements map to EGL (continued)

I4GL	EGL
FUNCTION *fn1*(*a*,*b*) DEFINE *a* INTEGER DEFINE *b* INTEGER DEFINE *c* INTEGER // ... RETURN c END FUNCTION	function fn1 (a INT in, b INT in) returning (INT) c int; // ... return (c); end //function
Multiple returns: FUNCTION *fn1*(*a*,*b*) DEFINE *a* INTEGER DEFINE *b* INTEGER DEFINE *c* INTEGER DEFINE *d* INTEGER RETURN *c*,*d* END FUNCTION	 function fn1 (a INT in, b INT in $_retvar1 INT out, $_retvar2 INT out) c int; d int; $_retvar1 = c $_retvar2 = d; return; end//Function
Record.*returns: CALL *fn1()* RETURNING *r.**; ... FUNCTION *fn1()*	 function fun1(/* returning */ $_retvar_1 rec_like_tab INOUT) move func_data to $_retvar_1; return; end
Record.mem1 THRU Record.mem2: FUNCTION *fn1*(*r.a* THRU *r.c*) DEFINE *r* LIKE *tabname*.*; END FUNCTION	// Expand member list to individual values FUNCTION fun1 (a INT IN, b UNICODE(20) IN, c INT IN) ... ... END // FUNCTION
GLOBALS ... END GLOBALS DEFINE *fvar* INT; GLOBALS DEFINE *gvar* INT; END GLOBALS GLOBALS *filename*	Variables defined at the top of the file but not in the GLOBALS section will be private to the file. Variables defined in the GLOBALS section can be referenced by other files and packages. private fvar int; gvar int; import packageOfFilename.*; // library declaration use filename;
LABEL LABEL xxx:	xxx: • Will be used along with GOTO statement
MAIN	FUNCTION $_*filename*_main()
REPORT	External tool

Storage Manipulation Statements

Table A-4. How 4GL Storage Manipulation Statements map to EGL

I4GL	EGL
DEFINE *var* BLOB, *var2* TEXT; LOCATE *var, var, var2* IN MEMORY LOCATE *var, var2* IN FILE LOCATE *var, var2* IN FILE *"filename"*; LOCATE *var, var2* IN FILE *filevar*;	Var Blob; Var2 Clob; //LOCATE var, var, var IN MEMORY; attachBlobToTempFile*(var)*; attachBlobToTempFile*(var2);* attachBlobToFile*(var,"filename")*; attachBlobToFile*(var2, "filename")*; attachBlobToFile*(var, filevar)*; attachBlobToFile*(var2 filevar)*;

FREE *preparedStatement*
FREE *Cursor*
FREE *bytevar*
FREE *textvar*
	FreeSql (preparedStatement);
	// free cursor - not required.
	FreeBlob(blobvar);
	FreeClob(clobvar);

INITIALIZE	SET
LET *x* = 10;	x = 10;
VALIDATE VALIDATE LIKE	Not supported.

Program Flow Control Statements

Table A-5. How 4GL Program Flow Control Statements map to EGL

I4GL	EGL
/* In 14GL foreach does open, fetch and close. */ DECLARE *cursorname* FOR *stmt*: FOREACH *cursorname* USING *a, b* INTO *x,y,z* WITH REOPTIMIZE END FOREACH	FOREACH The EGL foreach statement does the fetch and close: //DECLARE cursorname for stmt Open cursorname using a, b Foreach (from cursorname into x,y,z) /* REOPTIMIZE not supported */ End

CALL *fn1*(*a,b*) RETURNING c;
 	
LET c = *fn1*(*a,b*)
 	
CALL *fn1*(*a,b*) RETURNING *a*
 	
CALL *fn1*(*a,b*) RETURNING *a,b*
 	
CALL *fn1*(*rec*.*) RETURNING r*ec2*.*;
	c = fn1(a,b);
	
	c = fn1(a,b);
	
	a = fn1(a,b);
	
	fn1(a,b, /* returning */ a,b);
	
	fn1(rec, /* returning */rec2);

CASE	SWITCH CASE ... CASE ... CASE
CONTINUE	CONTINUE

Table A-5. How 4GL Program Flow Control Statements map to EGL (continued)

I4GL	EGL

DATABASE *mydb@SERVER2*;
 	
DEFINE *rec* RECORD LIKE *table*.*
 	
/*14GL internally puts a $database statement in the main function in the **.ec** file. This generates in MIN*/
	//DATABASE mydb@SERVER2
	import IfmxSchema.server2.mydb.*;
	
	rec IfmxSchema.server2.mydb.rec_like_table;
	
	DefineDatabaseAlias("DEFAULT",getProperty ("mydb@SERVER2"));
	connect("DEFAULT",
	getProperty("DEFAULT_USERID"),
	getProperty("DEFAULT_PASSWORD"), type1, explicit,
	repeatableRead, noAutoCommit);
	// ANSI DB settings

EXIT CASE EXIT CONSTRUCT EXIT DISPLAY EXIT FOR EXIT FOREACH EXIT INPUT EXIT MENU EXIT PROGRAM EXIT REPORT EXIT WHILE	Exit Exit openUI Exit openUI Exit for Exit foreach Exit openUI Exit openUI EXIT PROGRAM Call *reportName_TERMINATE()*: Exit while
FOR	FOR
GLOBALS *filename*	//GLOBALS "filename" This statement is ignored. The required Imports and Use statements are generated based on the Database Schema and Library manifests included in the Application conversion project and the individual library references to those external projects.
GOTO	GOTO
IF	IF

IF <*cond*> THEN
ELSE
END IF
	if (<cond>)
	else
	end

OUTPUT TO REPORT	Indirectly via external tool
RETURN; RETURN *a*; RETURN *rec*.*; RETURN *a, b*; RETURN *rec.col_a* THRU *rec.col_c*;	RETURN: RETURN (a); Move rec to $_retvar1; RETURN; $_retvar1 = a; $_retvar2 = b; return; function ($_retvar_1 *type* OUT, $_retvar_2 *type* OUT, $_retvar_3 *type* OUT) ... $_retvar_1 = rec.col_a; $_retvar_2 = rec.col_b; $_retvar_3 = rec.col_c; return; end

Table A-5. How 4GL Program Flow Control Statements map to EGL (continued)

I4GL	EGL
RUN *"cmd"* LINE RUN *"cmd"* FORM RUN *"cmd"* WITHOUT WAITING RUN .. RETURNING *x*;	CallCmd("cmd", Line); CallCmd("cmd", Form); StartCmd("cmd", Line); CallCmd("cmd",Line); x = sysVar.returnCode;
SQL ... END SQL	execute #sql {...}
WHILE *cond* ... END WHILE	while (*cond*) ... end

Note: Mappings for the FINISH REPORT, OUTPUT TO REPORT, START REPORT and TERMINATE REPORT 4GL Program Flow Control statements are included in the "4GL Report Execution Statements" on page A-11.

Compiler Directives

Table A-6. How 4GL Compiler Directives map to EGL

I4GL	EGL
DEFER INTERRUPT DEFER QUIT	consoleLib.deferInterrupt=YES; consoleLib.deferQuit=YES;
GLOBALS *filename*	Import packageOfFilename.*; Use *filename*; // library name.
SQL ... END SQL	EXECUTE #sql{ ... } **Note:** Program variables must be identified by the colon ":" prefix.
WHENEVER SQLERROR CONTINUE; SQL ... END SQL SQL ... END SQL	//WHENEVER SQLERROR CONTINUE; Try EXECUTE #sql{ ... } onException end; Try EXECUTE #sql{ ... } onException end;
WHENEVER SQLERROR CALL *XYZ*; SQL ... END SQL SQL ... END SQL	//WHENEVER SQLERROR CALL; try EXECUTE #sql{ ... } onException /* ERROR */ xzy() end try EXECUTE #sql{ ... } onException /* ERROR */ xzy() end
WHENEVER SQLERROR GOTO :*ABC*; SQL ... END SQL SQL ... END SQL	//WHENEVER SQLERROR GOTO; try execute #sql{ ... }; execute #sql(... }; onException /* ERROR */ gotoABC; end

Table A-6. How 4GL Compiler Directives map to EGL (continued)

I4GL	EGL
WHENEVER SQLERROR GOTO :*ABC* WHENEVER WARNING STOP SQL ... END SQL SQL ... END SQL	/*whenever sqlerror goto :ABC*/ /*whenever warning stop */; try execute #sql{ ... }; if (SQLCODE > 0 && SQLCODE !=100) then exit program; end /* WARNING */ onException /* ERROR */ gotoABC; end try execute #sql{ ...}; if (SQLCODE > 0 && SQLCODE !=100) then exit program; end /* WARNING */ onException /* ERROR */ gotoABC; end
WHENEVER ERROR CALL; SQL ... END SQL INPUT ... END INPUT	//WHENEVER ERROR CALL abc; try Execute #sql{ ... } onException call *abc()*; end try openUI ... End onException call *abc()*; end

WHENEVER SQLWARNING CALL *abc*;
SQL ... END SQL
 	
SQL ... END SQL
	//WHENEVER SQLWARNING CALL abc;
	try Execute #sql{ ... } onException end
	if sqlcode > 0 then
	call *abc()*;
	end
	try Execute #sql{ ... } onException end
	if sqlcode > 0 then
	call *abc()*;
	end

I4GL Forms to EGL Console User Interface

Table A-7. How 4GL Form Statements map to EGL Console User Interface Statements

I4GL	EGL
CLEAR SCREEN	activateWindow (*consoleLib.screen*); --or-- clearActiveWindow();
CLEAR WINDOW SCREEN	clearWindow(*consolelib.screen*);
CLEAR WINDOW *windowName*	clearWindow([*windowObject*]); --or-- clearWindowByName({*windowName*})
CLEAR FORM	clearActiveForm();
CLEAR *listOfFieldNames*	clearFields() --or-- clearConsoleFields(*fieldName* {, *fieldName*});
CLOSE FORM	NA

Table A-7. How 4GL Form Statements map to EGL Console User Interface Statements (continued)

I4GL	EGL

CLOSE WINDOW *windowName*	closeWindow(*windowObject*);
	--or--
	closeWindowByName(*windowName*);

CONSTRUCT	openUI {*isConstruct*=yes} *formObject* end
CURRENT WINDOW IS *windowName*	activateWindow(*windowObject*); --or-- activateWindowByName(*windowName)*
CURRENT WINDOW IS SCREEN	activateWindow(*consoleLib.screen*)
DEFER INTERRUPT	*c*onsoleLib.deferInterrupt=yes;
DEFER QUIT	consoleLib.deferQuit=yes;
DISPLAY	displayLineMode()
DISPLAY AT x,y;	displayAtPosition()
DISPLAY AT x;	displayAtLine()*:*
DISPLAY *a, b* TO *field1, field2*	openUI {*displayOnly*=yes, *bindingbyName*=no} a*ctiveForm*.field1, a*ctiveForm*.field2 bind *a,b* end;
DISPLAY BY NAME *fname, lname*;	openUI {*displayOnly*=yes, *bindingbyName*=yes} *activeForm* bind *fname*, *lname* end;
DISPLAY BY NAME *Re*.*	openUI {*displayOnly*=yes, *bindingbyName*=yes} *activeForm* bind <each rec element>;
DISPLAY ARRAY	openUI {*displayOnly*=yes} *formArrayDictionary* bind *programRecordArray* end
DISPLAY FORM	clearForm() openUI {*displayOnly*=yes} *consoleForm* end
ERROR	displayError (*message*);
INPUT	openUI *consoleForm* bind *program Variables* end --or-- openUI *consoleField*List bind program *Variables* end
INPUT ARRAY	openUI *activeForm.arrayDictionary* bind *programRecordArray* end

MENU	openUI new Menu {
	labelText="Menu1",
	menuItems=[new MenuItem {accelerators={"F1"],
	name="Cmd1",
	labelText="Command1"
	}
	//Repeat for other commands --separate each with a comma
	]
	}
	OnEvent
	(MENU_ACTION:"Cmd1")
	...egl statements...
	end;

MESSAGE	displayMessage (*message*);
NEXT FIELD *fieldName*	gotoFieldByName(*fieldIdentifier*)
OPEN FORM *formName* from *fileName*	formNameConsoleformType;

Table A-7. How 4GL Form Statements map to EGL Console User Interface Statements (continued)

I4GL	EGL

OPEN WINDOW *windowName*	the Window {name="theWindow", size=[rr,cc].
	Position=[zz,yy]};
	openWindow (*theWindow*);
	--or--
	openWindowByName("theWindow");
	
	Migration Mapping:
	openWindow(new Window{
	name="theWindow", size=[rr,cc],
	position=[xx,yy]});

OPTIONS	Use consoleLib.property

PROMPT	openUI new Prompt
	{message="Do you want to continue?", isChar=yes} bind
	userAnswer end;
	
	--or--
	
	myPrompt Prompt
	{message="What is your name?"};
	openUI myPrompt bind usersName
	end;
	
	For Line Mode operations, use promptLineMode():
	
	userAnswer Char(1);
	userAnswer=promptLineMode
	("Continue (y/n)?");

SCROLL	scrollDownPage() -or- scrollDownLines(*integerCount*) -or- scrollUpPage() -or- scrollUpLines (*integerCount*)
ARR_COUNT()	currentArrayCount()
ARR_CURR()	currentArrayDataLine()
SCR_LINE()	currentArrayScreenLine()
FGL_DRAWBOX()	drawBox() --or-- drawBoxWithColor()
FGL_GETKEY()	getKey()
FGL_KEYVAL()	getKeyCode()
NEXT OPTION *fieldName*	gotoMenuItem() --or-- gotoMenuItemByName()
INFIELD()	isCurrentField() --or-- isFieldModifiedByName()
FIELD_TOUCHED()	isFieldModified() --or-- isFieldModifiedByName()
FGL_LASTKEY	lastKeyTyped()
NEXT_FIELD	nextField()
PREVIOUS_FIELD	previousField()

Table A-7. How 4GL Form Statements map to EGL Console User Interface Statements (continued)

I4GL	EGL
FGL_SETCURRLINE	setArrayLine()
FGL_SCR_SIZE()	syslib.size(*screenArray*)
SHOWHELP()	showHelp()

4GL Report Execution Statements

The Conversion Utility converts I4GL Reports files into both EGL (**.egl**) and JasperReport (**.jrxml**) files. For specific information on how I4GL Report syntax converts, see "Understanding Report Conversion" on page 4-14.

The following I4GL Report statements do not convert to EGL or JasperReports; however, to maintain syntactically correct code, the Conversion Utility will generate redundant code in the report handler for these statements:

- EXIT REPORT
- NEED
- PAUSE

Table A-8. How 4GL Report Execution Statements map to EGL/JasperReport

I4GL	EGL / JasperReport
PRINT	See "I4GL FORMAT Section" on page 4-19.
SKIP	Indirectly mapped via external tool.

I4GL Report Driver Statements

I4GL report driver statements map to EGL functions. For an example of I4GL report code and the comparable examples of the EGL driver functions, see Appendix B, "I4GL Report Conversion Code Example."

Table A-9. How 4GL Report Driver Statements map to EGL

I4GL	EGL
FINISH REPORT *reportname*	*reportname*_FINISH( )
OUTPUT TO REPORT *reportname* OUTPUT TO REPORT *reportname* (*a, b, c*)	*reportname*_OUTPUT ( ) *reportname*_OUTPUT (a, b, c)
START REPORT *reportname report options*	*reportname*_START( ) **Note:** START REPORT report options do not map to EGL. In addition, the EGL function call has no arguments and no return value.
TERMINATE REPORT *reportname*	*reportname_TERMINATE* ( )

Built-in 4GL Functions, Variables, and Constants

Table A-10. How 4GL Built-in functions, variables, and constants map to EGL

I4GL	EGL
ARG_VAL (*int-expr*)	getCmdLineArg(*int-expr*)
ARR_COUNT ()	currentArrayCount()

Table A-10. How 4GL Built-in functions, variables, and constants map to EGL (continued)

I4GL	EGL
ASCII (*int-expr*)	integerAsChar(*int-expr*)
COLUMN *int-expr* COLUMN *int-const*, PAGENO, LINENO	No direct mapping. Indirectly mapped via external tool.
CURSOR_NAME (*"Identifier"*)	Not supported.
DATE(*char*)	dateValue(*char*)
Date(*integer*)	dateValue(*integer*)
DAY(*datevalOrDatetime*)	dayOf(*dateexpr);*
DOWNSHIFT (*char-expr*)	lowercase*(charexpr)*
ERR_PRINT (*int-expr*)	displayError*(err_get("number"));*
ERR_QUIT (*int-expr*)	displayError(err_get("number")); exit program;
ERRORLOG (*int-expr*)	errorLog()
EXTEND (*value*, *qualifier*)	extend (*value*, *qualifierpattern*)
FALSE	0/*FALSE*/
FGL_DRAWBOX (*nlines*, *ncols*, *begy*, *begx*, *color*)	drawBox(*int, int, int, int*) drawBoxWithColor(*int, int, int, int, colorKind*)
FGL_GETENV (*char-expr*)	getproperty(*charexpr*)
FGL_GETKEY () // INPUT KEYSTROKE	getKey()
FGL_KEYVAL (*char-expr*)	getKeyCode(*String*)
FGL_LASTKEY() //Doesn't wait,	lastKeyTyped()
FGL_SCR_SIZE (*arrayname*) FGL_SCR_SIZE *("arrayname"*);	syslib.size(*activeForm*[*arrayname*]*);* syslib.size(*activeForm."arrayname"*);
FGL_SETCURRLINE()	setArrayLine()
Intval UNITS *dateTImeUnit*	IntervalValueWithPattern(*intval*, *"UnitsString"*);
LENGTH *(char-expr)*	StrLeng*(charexpr*)
LET *a* = ERR_GET *(*SQLCODE*) // Get Message* *LET a* = ERR_GET *(int-expr) // Get Message*	err_get(*SQLCODE*); err_get(*int-expr*);
MONTH (*datevalOrDatetime*)	monthOf(*dateexpr*);
MDY (*intmonth, intdays, intyear*)	MDY(*intmonth, intdays, intyear)*
NEXT FIELD NEXT FIELD *fieldname*	nextField() gotoFieldByName (*"fieldname"*);
NEXT FIELD PREVIOUS	previousField()
NEXT OPTION *name*	gotoMenutItemByName(*"name"*);
NUM_ARGS ()	getCmdLineArgCount()
ORD *(char-expr)*	characterAsInteger(*char-expr*)
SCR_LINE*()*	currentArrayScreenLine()
SET_COUNT *(int-expr)*	setCurrentArrayCount()
SHOWHELP *(int-expr)*	showHelp(*"int-expr"*);
STARTLOG *("filename.filetype")*	startLog()
TIME	timeStamp(*"hhmmss"*);

Table A-10. How 4GL Built-in functions, variables, and constants map to EGL (continued)

I4GL	EGL
TIME (DateTimeValue)	currentTime();
TODAY	currentDate();
TRUE	I/*TRUE*/
UPSHIFT *(char-expr)*	uppercase(*char-expr*)
WEEKDAY *(datevalOrDatetime)*	weekdayOf(*datexpr*);
YEAR *(datevalOrDatetime)*	yearOf(*dateexpr*);

Built-in and External SQL Functions and Procedures

Table A-11. How 4GL Built-in and External SQL Functions and Procedures map to EGL

I4GL	EGL
CREATE FUNCTION	Execute #sql{ create function ... }
CREATE FUNCTION FROM	Not supported
CREATE PROCEDURE FROM	Not supported
CREATE ROUTINE FROM	Not supported
EXECUTE FUNCTION	Execute #sql{ execute function ... }
EXECUTE PROCEDURE	Execute #sql{ execute procedure ... }

Operators

Keyword-Based Operators

Table A-12. How 4GL Keyword-Based Operators map to EGL

I4GL	EGL
ASCII *int-expr*	integerAsCharacter(*int-expr*)
AND	&&
Value BETWEEN *expr1* AND *expr2*	*Value >= expr1 && Value <= expr2*
char-expr CLIPPED	Clip(*char_expr*)
CURRENT CURRENT *qualifier*	currentTimeStamp() extend (*currentTimeStamp*(), *"yyyMMddhhmmss"*;
FIELD_TOUCHED (*field-list*)	isFieldModified(*consoleField)* --or-- isFieldModifiedByName(*String*)
GET_FLDBUF (*field*) GET_FDLDBUF (*field-list*)	activeForm.field individual assignments with corresponding field.Value
INFIELD (*field*)	isCurrentField(*ConsoleField*) isCurrentFieldByName(*String*)
INT_FLAG	interruptRequested
Xyz IS NULL *Xyz* IS NOT NULL	Xyz IS NULL Xyz NOT NULL
LENGTH (*char-expr*)	StrLen (char-expr)

Table A-12. How 4GL Keyword-Based Operators map to EGL (continued)

I4GL	EGL
LIKE *x* NOT LIKE *y*	like !(*x* like *y*)
LINENO	Indirectly supported by external tool
MATCHES *expr* *x* NOT MATCHES *y*	matches !(*x* matches *y*)
int-expr MOD *int-expr*	Int-expr % int-expr
NOT	!
NOT FOUND	100/*NOTFOUND*/
OR	||
PAGENO	Indirectly supported by external tool
QUIT_FLAG	quitRequested
int-expr SPACE	Spaces(*int-expr*)
int-expr SPACES	Spaces(*int-expr*)
STATUS	SQLCODE
TIME	See "Built-in 4GL Functions, Variables, and Constants" on page A-11.
TODAY	See "Built-in 4GL Functions, Variables, and Constants" on page A-11.
int-expr UNITS *time-keyword*	See "Built-in 4GL Functions, Variables, and Constants" on page A-11.
int-expr USING *format-string* *char-expr* USING *format-string* datetime USING *format-string*	Format(*expression*, *formatString*) formatNumber(intval, "######") formatDate(mydate, "mm/dd/yy") format(*char-exp*, *formatString*) formatDate(datetime, *formatString*)
char-expr WORDWRAP	Handled by JasperReports

Operators Represented by Non-Alphabetic Symbols

Table A-13. How Operators represented by Non-Alphabetic Symbols map to EGL

Alphabetic	4GL	EGL
Addition	+	+
Comments	-- Single-line comment # Single-line comment { multiple-line comment }	// Single line comment /* multiple-line comment */
Division	/	/
Exponentiation	**	**
Greater than	>	>

Table A-13. How Operators represented by Non-Alphabetic Symbols map to EGL (continued)

Alphabetic	4GL	EGL
Greater than or equal to	>=	>=
Less than	<	<
Less than or equal to	<=	<=
Membership	.	.
Modulus	MOD	%
Multiplication	*	*
Not equal to	!= or <>	!=
Sub string	[first,last]	[first:last]
Subtraction	-	-
Unary negative	-	-
Unary positive	+	+

SQL Cursor Manipulation Statements

Table A-14. How 4GL Cursor Manipulation Statements map to EGL

I4GL	EGL
CLOSE *cursorname*	try close *cursorname*; onException end
DECLARE *c* CURSOR FOR *stmt*; DECLARE C CURSOR FOR SELECT * FROM SYSTABLES;	/* declare c cursor for stmt */ try PREPARE $_STMT_C FROM "SELECT * FROM SYSTABLES"; onException end **Note:** The Scroll and Hold attributes from DECLARE are now specified when the cursor is opened.

// WHENEVER SQLERROR STOP;
 	
FETCH *c*;
FETCH NEXT *C*;
FETCH PREVIOUS *C*;
FETCH RELATIVE 1 *C*;
FETCH RELATIVE -1 *C*;
FETCH FIRST *C*;
FETCH LAST *C*;
	/* FYI: WHENEVER SQLERROR STOP */
	/
	get next from c;
	get next from C;
	get previous from C
	get relative(1) from C;
	get relative(-1) from C;
	get first from C;
	get last from C;

FLUSH	NO-OP
FREE	freeSQL
OPEN	OPEN OPEN cursor WITH *statementid*; OPEN cursor WITH *statementid* USING *param1, param2*; OPEN cursor SCROLL WITH *statementid*;
OPEN ... WITH REOPTIMIZE	OPEN /* REOPTIMIZE not supported */
PREPARE	PREPARE
PUT	EXECUTE
SET AUTOFREE	NO-OP
SQL ... END	execute #sql { ... }

SQL Data Definition Statements

Table A-15. How 4GL SQL Data Definition Statements map to EGL

I4GL	EGL
ALTER INDEX	Execute #sql{ ... }
ALTER FRAGMENT	Execute #sql{alter fragment ... }
ALTER TABLE	Execute #sql{ ... }
CLOSE DATABASE	Execute #sql{ close database }
CREATE DATABASE	Execute #sql{ ... }
CREATE EXTERNAL TABLE	Execute #sql{ ... }
CREATE INDEX	Execute #sql{ ... }
CREATE PROCEDURE FORM	Execute #sql{ ... }
CREATE ROLE	Execute #sql{ ... }
CREATE SCHEMA	Execute #sql{ ... }
CREATE SYNONYM	Execute #sql{ ... }
CREATE TABLE	Execute #sql{ ... }
CREATE TRIGGER	Execute #sql{ ... }
CREATE VIEW	Execute #sql{ ... }
CONNECT DATABASE	defineDatabaseHandle(getProperty(dbname)); connect(getProperty(″DEFAULT_USER″), getProperty(″DEFAULT_PASSWORD″), explicit, autoCommit);
DROP DATABASE	Execute #sql{ ... }
DROP INDEX	Execute #sql{ ... }
DROP PROCEDURE	Execute #sql{ ... }
DROP ROLE	Execute #sql{ ... }
DROP SYNONYM	Execute #sql{ ... }
DROP TABLE	Execute #sql{ ... }
DROP TRIGGER	Execute #sql{ ... }
DROP VIEW	Execute #sql{ ... }
RENAME COLUMN	Execute #sql{ ... }
RENAME DATABASE	Execute #sql{ ... }
RENAME TABLE	Execute #sql{ ... }

SQL Data Manipulation Statements

Table A-16. How 4GL Data Manipulation Statements map to EGL

I4GL	EGL
INSERT	Execute #sql{ INSERT ... }
DELETE	Execute #sql{ DELETE ... }
LOAD	loadTable(*filename*, sql, delimiter);
OUTPUT	Indirectly by external tool

Table A-16. How 4GL Data Manipulation Statements map to EGL (continued)

I4GL	EGL
SELECT	Execute #sql{ select } into ...
UNLOAD	unloadTable(*filename*, sql, delimiter);
UPDATE	Execute #sql{ UPDATE ... }

SQL Dynamic Management Statements

Table A-17. How 4GL SQL Dynamic Management Statements map to EGL

I4GL	EGL
EXECUTE	EXECUTE statement
EXECUTE IMMEDIATE	Execute #SQL{ ... }
FREE	FREE
PREPARE	PREPARE
SET DEFERRED_PREPARE	NO-OP

SQL Query Optimization Statements

Table A-18. How 4GL SQL Query Optimization Statements map to EGL

I4GL	EGL
SET OPTIMIZATION	Execute #sql{set optimization ... }
SET EXPLAIN	Execute #sql{set explain ... }
SET PDQPRIORITY	Execute #sql{ ... }
SET RESIDENCY	Execute #sql{ ... }
SET SCHEDUE LEVEL	Execute #sql{ ... }
UPDATE STATISTICS	Execute #sql{ ... }

SQL Data Access Statements

Table A-19. How 4GL SQL Data Access Statements map to EGL

I4GL	EGL
GRANT	Execute #sql{ ... }
GRANT FRAGMENT	Execute #sql{ ... }
LOCK TABLE	Execute #sql{ ... }
REVOKE	Execute #sql{ ... }
REVOKE FRAGMENT	Execute #sql{ ... }
SET ISOLATION	Execute #sql{ ... };
SET LOCK MODE	Execute #sql{ ... }
SET ROLE	Execute #sql{ ... }
SET SESSION	Execute #sql{ ... }
SET TRANSACTION	Execute #sql{ ... }
UNLOCK TABLE	Execute #sql{ ... }

SQL Data Integrity Statements

Table A-20. How 4GL SQL Data Integrity Statements map to EGL

I4GL	EGL
BEGIN WORK	*beginDatabaseTransaction();*
COMMIT WORK	*commit()*
ROLLBACK WORK	*rollback()*
SET Database Object Mode	Execute #sql{... }
SET LOG	Execute #sql{... }
SET PLOAD FILE	Execute #sql{... }
SET TRANSACTION MODE	Execute #sql{... }
START VIOLATIONS TABLE	Execute #sql{... }
STOP VIOLATIONS TABLE	Execute #sql{... }

SQL Stored Procedure Statements

Table A-21. How SQL Stored Procedure Statements map to EGL

I4GL	EGL
EXECUTE PROCEDURE	Execute #sql{ ... }
SET DEBUG FILE TO	Execute #sql{ ... }

SQL Client/Server Connection Statements

Table A-22. How 4GL Connection Statements map to EGL

I4GL	EGL
SET CONNECTION *'conname'*;	setCurrentDatabase(*"conname"*);
SET CONNECTION *convar*;	setCurrentDatabase (*convar*)
SET CONNECTION *dbname*;	setCurrentDatabase(*"dbname"*)
SET CONNECTION DEFAULT;	setCurrentDatabase ("DEFAULT");
SET CONNECTION ... DORMANT	// set connection ... dormant

Table A-22. How 4GL Connection Statements map to EGL (continued)

I4GL	EGL
CONNECT TO *db@server*	setDatabaseHandle(getProperty("db@server")); connect(...);
CONNECT TO *db* AS *'conname'*	setDatabaseHandle(getProperty("db@server")); connect(...)
CONNECT TO DEFAULT	try try disconnect(); onException end try DefineDatabaseAlias(getProperty("@myserver")); getProperty("DEFAULT_USERID"), getProperty("DEFAULT_PASSWORD"), type1, explicit, repeatableRead, noAutoCommit); onException end onException end

DISCONNECT CURRENT
DISCONNECT DEFAULT
DISCONNECT ALL
DISCONNECT *'conname'*
DISCONNECT *convar*
	disconnect()
	disconnect("*DEFAULT*");
	disconnectAll();
	disconnect("*conname*");
	disconnect(*convar*);

SQL Optical Subsystems Statements

Table A-23. How SQL Optical Subsystems Statements map to EGL

I4GL	EGL
ALTER OPTICAL CLUSTER	Execute #sql{ ... }
CREATE OPTICAL CLUSTER	Execute #sql{ ... }
DROP OPTICAL CLUSTER	Execute #sql{ ... }
RELEASE	Execute #sql{ ... }
RESERVE	Execute #sql{ ... }
SET MOUNTING TIMINOUT	Execute #sql{ ... }

Environment Variables

Table A-24. How I4GL Environment Variables map to EGL and JDBC Properties

I4GL Environment Variables	EGL Properties	JDBC Properties
C4GLFLAGS	NO-OP	NO-OP
C4GLNOPARAMCHK	NO-OP	NO-OP
CC	NO-OP	NO-OP
COLUMNS	NO-OP	NO-OP
CLIENT_LOCALE	CLIENT_LOCALE	CLIENT_LOCALE
COLLCHAR	NO-OP	NO-OP
DBANSIWARN	NO-OP	NO-OP
DBCENTURY	NO-OP	DBCENTURY

Table A-24. How I4GL Environment Variables map to EGL and JDBC Properties (continued)

I4GL Environment Variables	EGL Properties	JDBC Properties
DBDATE	defaultDateFormat	DBDATE
DBDELIMITER	defaultDbDelimiterFormat	NO-OP
DBEDIT	NO-OP	NO-OP
DBESCWT	NO-OP	NO-OP
DBFORM	NO-OP	NO-OP
DBFORMAT	NO-OP	NO-OP
DBLANG	NO-OP	NO-OP
DBMONEY	defaultMoneyFormat defaultNumericFormat	NO-OP
DBPATH	NO-OP	NO-OP
DBPRINT	NO-OP	NO-OP
DBREMOTECMD	NO-OP	NO-OP
DBSPACETEMP	NO-OP	DBSPACETEMP
DBSRC	NO-OP	NO-OP
DBTEMP	NO-OP	DBTEMP
DBTIME	NO-OP	NO-OP
DBUPSPACE	NO-OP	DBUPSPACE
DBAPICODE	NO-OP	NO-OP
DB_LOCALE	DB_LOCALE	DB_LOCALE
DBNLS	NO-OP	NO-OP
ENVIGNORE	NO-OP	NO-OP
FET_BUF_SIZE	NO-OP	FET_BUF_SIZE
FGLPCFLAGS	NO-OP	NO-OP
FGLSKIPNXTPG	NO-OP	NO-OP
GL_DATE	defaultDateFormat	NO-OP
GL_DATETIME	defaultTimeStampFormat	NO-OP
INFORMIXC	NO-OP	NO-OP
INFORMIXONRETRY	NO-OP	NO-OP
INFORMIXCONTINUE	NO-OP	NO-OP
INFORMIXDIR	NO-OP	NO-OP
INFORMIXSERVER	INFORMIXSERVER	INFORMIXSERVER
INFORMIXSHMBASE	NO-OP	NO-OP
INFORMIXTERM	NO-OP	NO-OP
IXOLDFLDSCOPE	NO-OP	NO-OP
LANG	NO-OP	NO-OP
LINES	NO-OP	NO-OP
ONCONFIG	NO-OP	NO-OP
PATH	NO-OP	NO-OP
PDQPRIORITY	NO-OP	PDQPRIORITY
PROGRAM_DESIGN_DBS	NO-OP	NO-OP

Table A-24. How I4GL Environment Variables map to EGL and JDBC Properties (continued)

I4GL Environment Variables	EGL Properties	JDBC Properties
PSORT_DBTEMP	NO-OP	PSORT_DBTEMP
PSORT_NPROCS	NO-OP	PSORT_NPROCS
SQLEXEC	NO-OP	NO-OP
SQLRM	NO-OP	NO-OP
SQLRMDIR	NO-OP	NO-OP
SUPOUTPIPEMSG	NO-OP	NO-OP
SERVER_LOCALE	NO-OP	NO-OP
TERM	NO-OP	NO-OP
TERMCAP	NO-OP	NO-OP
TERMINFO	NO-OP	NO-OP

Appendix B. I4GL Report Conversion Code Example

In This Appendix

This appendix provides an example of I4GL report code and the comparable examples of the EGL driver functions.

4GL Report Code

```
REPORT r_invoice (c, x)
                DEFINE c RECORD
             customer_num int,
             fname        char(15),
             lname        char(15),
             company      char(20),
             address1     char(20),
             address2     char(20),
             city         char(15),
             state        char(2),
             zipcode      char(5),
             phone        char(18),
    END RECORD

   DEFINE x RECORD
       order_num INT,
       order_date         date,
       ship_instruct      Char(40),
       backlog            char,
       po_num             char(10),
       ship_date date,
       ship_weight        decimal(8,2),
       ship_charge        money(6,2)
       item_num  smallint,
       stock_num smallint,
       manu_code char(3),
       quantity  smallint,
       total_price         money(8,2),
       description         char(15),
       unit_price          money(6,2),
       unit                char(4),
       unit_descr          char(15),
       manu_name  char(10)
   END  RECORD
...
...
...
END  REPORT
```

EGL Driver Functions Generated from 4GL Code

```
Function r_invoice_START()
    execute #sql {
        create temp table r_invoice_report_table (
             c_customer_num int ,
             c_fname char (15),
             c_lname char (15) ,
             c_company char (20) ,
             c_address1 char (20) ,
             c_address2 char (20) ,
             c_city char (15) ,
             c_state char (2) ,
```

```
            c_zipcode char (5) ,
            c_phone char (18) ,
            x_order_num int ,
            x_order_date date ,
            x_ship_instruct char (40) ,
            x_backlog char ,
            x_po_num char (10) ,
            x_ship_date date ,
            x_ship_weight decimal (8, 2),
            x_ship_charge_money (6, 2),
            x_item_num smallint ,
            x_stock_num smallint ,
            x_manu_code char (3) ,
            x_quantity smallint ,
            x_total_price money (6, 2),
            x_description char (15) ,
            x_unit_price money (6, 2),
            x_unit char (4) ,
            x_unit_descr char (15) ,
            x_manu_name char (10)
        )
    }
end
Function r_invoice_OUTPUT (c recordtype_v4_c IN, x recordtype_v4_x IN)

   Prepare insert_stmt from "insert into r_invoice_report_table +
       "values(?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,)";
   Execute insert_stmt using (
           c.customer_num,
           c.fname,
           c.lname,
           c.company,
           c.address1,
           c.address2,
           c.city,
           c.state,
           c.zipcode,
           c.phone,
           x.order_num,
           x.order_date,
           x.ship_instruct,
           x.backlog,
           x.po_num,
           x.ship_date,
           x.ship_weight,
           x.ship_charge,
           x.item_num,
           x.stock_num,
           x.manu_code,
           x.quantity,
           x.total_price,
           x.description,
           x.unit_price,
           x.unit,
           x.unit_descr,
           x.manu_name
     )

End
Function r_invoice_FINISH()
    egl4glReport     Report;
    egl4glReportData ReportData;

    egl4glReport.reportDesignFile      = "r_invoice_XML.jasper";
    egl4glReport.reportDestinationFile = "r_invoice.jrprint";
    egl4glReport.reportExportFile      = "r_invoice.txt";
```

```
    egl4glReportData.sqlStatement = "Select * From" +
                                     " r_invoice_report_table";

    egl4glReport.reportData = egl4glReportData;

   ReportLib.fillReport(egl4glReport, DATASOURCE_SQL_STATEMENT );
   ReportLib.exportReport(egl4glReport, EXPORT_TEXT);

    Execute #SQL  {
         Drop Table r_invoice_report_table
    }
End
Function r_invoice_TERMINATE()
     Execute #SQL  {
         Drop Table r_invoice_report_table
     }
End

...
...
     Record recordtype_v4_c type BasicRecord
            customer_num INT;
            fname            UNICODE(15);
            lname            UNICODE(15);
            company          UNICODE(20);
            address1         UNICODE(20);
            address2         UNICODE(20);
            city             UNICODE(15);
            state            UNICODE(2);
            zipcode          UNICODE(5);
            phone            UNICODE(18);
END


     RECORD recordtype_v4_x type BasicRecord
            order_num INT;
            order_date      DATE;
            ship_instruct   UNICODE(40);
            ship_instruct   UNICODE(10;
            backlog         UNICODE(1);
            po_num          UNICODE(10);
            ship_date DATE;
            ship_weight     DECIMAL;
            ship_charge     DECIMAL(6,2);
            item_num  smallint;
            stock_num smallint;
            manu_code UNICODE(3);
            quantity  smallint;
            total_price     DECIMAL(8,2);
            description     UNICODE(15);
            unit_price      DECIMAL(6, 2):
            unit            UNICODE(4);
            unit_descr      UNICODE(15);
            manu_name  UNICODE(10);
END
```

Appendix C. I4GL Form Code to EGL Form Code Example

In This Appendix

This appendix provides an example of I4GL form code and the generated EGL form code.

4GL Form Code

```
------------------------      customer.4gl    ----------------------------
GLOBALS
 DEFINE p_customer RECORD
    customer_num int,
    fname        char(15),
    lname        char(15),
    company      char(20),
    address1     char(20),
    address2     char(20),
    city         char(15),
    state        char(2),
    zipcode      char(5),
    phone        char(18)
 END RECORD
END GLOBALS

MAIN

  DISPLAY "Starting form Customer "
  SLEEP 2

  CALL input_cust()

END MAIN

FUNCTION input_cust()

    OPEN FORM customer FROM "customer"
    DISPLAY FORM customer
    ATTRIBUTE(BLUE)

    DISPLAY "Press ESC to enter new customer data" AT 1,1
    INPUT BY NAME p_customer.*
        AFTER FIELD state
            DISPLAY "In field state "
          DISPLAY "Press ESC to enter new customer data", "" AT 1,1
    ON KEY (F1, CONTROL-F)
            DISPLAY "Control-F pressed"
    ON KEY (F2, CONTROL-Y)
      LET int_flag = TRUE
      EXIT INPUT
    END INPUT
    IF int_flag
    THEN
     LET int_flag = FALSE
        RETURN(FALSE)
    END IF

    DISPLAY BY NAME p_customer.fname ATTRIBUTE(MAGENTA)

END FUNCTION
```

```
----------------------    customer.per ------------------------
DATABASE FORMONLY
SCREEN
{


                          Customer Form

        Number      :[f000       ]
        Owner Name  :[f001           ][f002           ]
        Company     :[f003                ]
        Address     :[f004                ]
                     [f005                ]
        City        :[f006           ] State:[a0] Zipcode:[f007 ]
        Telephone   :[f008              ]

}


ATTRIBUTES
f000 = FORMONLY.customer_num TYPE INT, NOENTRY;
f001 = FORMONLY.fname TYPE CHAR, UPSHIFT ;
f002 = FORMONLY.lname TYPE CHAR, COLOR=RED;
f003 = FORMONLY.company TYPE CHAR, COMMENTS="Company name",
COLOR=MAGENTA;
f004 = FORMONLY.address1 TYPE CHAR, AUTONEXT, COLOR=MAGENTA;
f005 = FORMONLY.address2 TYPE CHAR, COLOR=MAGENTA;
f006 = FORMONLY.city TYPE CHAR, REQUIRED;
a0 = FORMONLY.state TYPE CHAR, UPSHIFT;
f007 = FORMONLY.zipcode TYPE CHAR;
f008 = FORMONLY.phone TYPE CHAR, PICTURE = "###-###-#### XXXXX";

INSTRUCTIONS
SCREEN RECORD customer (FORMONLY.customer_num THRU FORMONLY.fname)
```

EGL Code

```
------------------------  customer.egl --------------------------------
Package test;
/*Program*/
Library customer{localSQLScope=YES}

  p_customer recordtype_p_customer;

FUNCTION $_customer_i4glmain()

  displayLineMode( "Starting Customer form ");
  wait( 2);
  input_cust();

END

FUNCTION input_cust()
returns (INT)
    $_FORM_customer customerForm{ name="customer" };
    ConsoleLib.CurrentDisplayAttrs{ color=BLUE };
    displayFormByName( "customer");
    displayAtPosition( "Press ESC to enter new customer data"
        , 1,1);
    OpenUI{setInitial=YES, bindingByName=YES} activeForm bind
                p_customer.customer_num, p_customer.fname,
                p_customer.lname, p_customer.company,
                p_customer.address1, p_customer.address2,
                p_customer.city, p_customer.state,
                p_customer.zipcode, p_customer.phone
        onEvent(AFTER_FIELD:"state")
            displayLineMode( "In field state ");
```

```
        displayAtPosition( "Press ESC to enter new customer data"
                           +  "" , 1,1);
        onEvent(ON_KEY:"F1","CONTROL_F")
            displayLineMode( "Control-F pressed");
        onEvent(ON_KEY:"F2","CONTROL_Y")
            interruptRequested = YES;
            EXIT OpenUI;
    end;
    IF ( interruptRequested != NO)
        interruptRequested = NO;
        RETURN((0/*FALSE*/));
    END
    ConsoleLib.CurrentDisplayAttrs{ color=MAGENTA };
    OpenUI { displayOnly=YES, bindingByName=YES } activeForm bind
      p_customer.fname  End
END

END // Program

record recordtype_p_customer type SqlRecord
    customer_num INT;
    fname UNICODE(15);
    lname UNICODE(15);
    company UNICODE(20);
    address1 UNICODE(20);
    address2 UNICODE(20);
    city UNICODE(15);
    state UNICODE(2);
    zipcode UNICODE(5);
    phone UNICODE(18);
 END


-------------------------  customer_program.egl ----------------------
Package test;

Program customer_program
use customer;
Function main()
   $_customer_i4glmain();
end
end /* Program */


------------------------- customerForm.egl ---------------------------
Package test;

Record customerForm type ConsoleForm { formSize = [13,80],
                                       showBrackets = yes}

   *ConsoleField { position = [3,28],  value = "Customer Form" };
   *ConsoleField { position = [5,9],   value = "Number      :" };
   *ConsoleField { position = [6,9],   value = "Owner Name  :" };
   *ConsoleField { position = [7,9],   value = "Company     :" };
   *ConsoleField { position = [8,9],   value = "Address     :" };
   *ConsoleField { position = [10,9],  value = "City        :" };
   *ConsoleField { position = [10,40], value = "State:" };
   *ConsoleField { position = [10,51], value = "Zipcode:" };
   *ConsoleField { position = [11,9],  value = "Telephone   :" };

   customer_num  ConsoleField     { position = [5,23], fieldLen = 11,
     dataType = "int",  protect = yes };
   fname         ConsoleField     { position = [6,23], fieldLen = 15,
     dataType = "unicode",  caseFormat = upper };
   lname         ConsoleField     { position = [6,40], fieldLen = 15,
      dataType = "unicode",  color = RED };
   company       ConsoleField     { position = [7,23], fieldLen = 20,
```

```
      dataType = "unicode",  comment = "Company name",  color = MAGENTA };
   address1      ConsoleField     { position = [8,23], fieldLen = 20,
      dataType = "unicode",  autonext = yes,  color = MAGENTA };
   address2      ConsoleField     { position = [9,23], fieldLen = 20,
      dataType = "unicode",  color = MAGENTA };
   city          ConsoleField     { position = [10,23],fieldLen = 15,
      dataType = "unicode",  inputRequired = yes };
   state         ConsoleField     { position = [10,47],fieldLen = 2,
      dataType = "unicode",  caseFormat = upper };
   zipcode       ConsoleField     { position = [10,60],fieldLen = 5,
      dataType = "unicode" };
   phone         ConsoleField     { position = [11,23],fieldLen = 18,
      dataType = "unicode",  pattern = "###-###-#### XXXXX" };

   customer Dictionary { customer_num=customer_num, fname=fname,
                         lname=lname, company=company,
                         address1=address1, address2=address2,
                         city=city, state=state, zipcode=zipcode,
                         phone=phone
                       };

end
```

Appendix D. Configuration File Templates

In This Appendix

This appendix provides template configuration files for Database Schema Extraction, Shared Library, and I4GL Application projects.

Note: If you are creating a configuration file manually, do not use any characters that would prevent the XML Parser from correctly differentiating values from the XML reserved characters. To prevent the XML parser from a possible misinterpretation, you should include the values for the XML element in the **<![CDATA[value]>** tag .

Database Schema Extraction Project Configuration File Template

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Internal DTD for Configuration file -->
<!DOCTYPE conversion [
   <!ELEMENT conversion (rootdir,dbconnection*)>
    <!ATTLIST conversion project CDATA #REQUIRED >
      <!ATTLIST conversion type CDATA #FIXED "schema">
   <!ELEMENT rootdir (egldir,artifactsdir?)>
   <!ELEMENT egldir (#PCDATA)>
   <!ELEMENT artifactsdir (#PCDATA)>
   <!ELEMENT dbconnection (database,server,host,port,user,password)+>
      <!ATTLIST dbconnection extractSystemTables  (yes|no) "no" >
      <!ATTLIST dbconnection client_locale CDATA #IMPLIED >
      <!ATTLIST dbconnection db_locale  CDATA #IMPLIED >
   <!ELEMENT database (#PCDATA)>
   <!ELEMENT server (#PCDATA)>
   <!ELEMENT host (#PCDATA)>
   <!ELEMENT port (#PCDATA)>
   <!ELEMENT user (#PCDATA)>
   <!ELEMENT password ANY>
]>

<conversion  project="stores7"   type="schema">

 <rootdir>
    <egldir>C:\egl\src\stores7</egldir>
    <artifactsdir>C:\tmp\stores7\ConversionArtifacts</artifactsdir>
 </rootdir>

    <dbconnection extractSystemTables="no" client_locale="en_US.8859-1"
 db_locale="en_US.8859-1">
        <database><![CDATA[stores7]]></database>
        <server><![CDATA[myserver]]></server>
        <host>mymachine.location.company.com</host>
        <port>1999</port>
        <user>jdoe</user>
        <password><![CDATA[password]]></password>
    </dbconnection>
</conversion >
```

Shared Library Project Configuration File Template

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Internal DTD for Configuration file -->
<!DOCTYPE conversion [
   <!ELEMENT conversion ( rootdir, manifestfiles*, fglfiles?, formfiles?,
```

```
   msgfiles* )>
        <!ATTLIST conversion project CDATA #REQUIRED >
      <!ATTLIST conversion type CDATA  #FIXED "library" >
    <!ATTLIST conversion locale CDATA #IMPLIED >
      <!ATTLIST conversion cursor (local | global) #IMPLIED>
    <!ATTLIST conversion defaultserver CDATA #IMPLIED>
   <!ELEMENT rootdir (fgldir?,egldir,artifactsdir?)>
   <!ELEMENT fgldir (#PCDATA)>
   <!ELEMENT egldir (#PCDATA)>
   <!ELEMENT artifactsdir (#PCDATA)>
   <!ELEMENT manifestfiles (file)+>
    <!ATTLIST manifestfiles type ( schema | library) #REQUIRED>
   <!ELEMENT fglfiles (file)+>
   <!ELEMENT formfiles (file)+>
   <!ELEMENT fontconfigfile (file)>
   <!ELEMENT file (#PCDATA)>
   <!ELEMENT msgfiles (file)+>
    <!ATTLIST msgfiles locale CDATA #IMPLIED >
]>

<conversion project="SharedLibraryProject"type="library"
locale="en_US.8859-1" cursor="local" defaultserver="myserver">
  <rootdir>
    <fgldir><![CDATA[C:\i4gl\SharedLibraryProject\fgl]]></fgldir>
    <egldir>C:\egl\SharedLibraryProject</egldir>
    <artifactsdir</artifactsdir>
  </rootdir>

  <manifestfiles type="schema">
    <file>C:\temp\stores7\ConversionArtifacts\manifest\
 Stores7Manifest.xml
</file>
  </manifestfiles>

  <fglfiles>
    <file><![CDATA[fgl\d4_cust.4gl]]></file>
    <file><![CDATA[fgl\d4_demo.4gl]]></file>
    <file><![CDATA[fgl\d4_globals.4gl]]></file>
  </fglfiles>

   <!-- report font file, optional -->
  <fontconfigfile>
    <file><![CDATA[C:\Documents and Settings\Administrator\myfont.xml]]>
</file>
  </fontconfigfile>

</conversion >
```

Application Project Configuration File Template

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Internal DTD for Configuration file -->
<!DOCTYPE conversion [
   <!ELEMENT conversion ( rootdir, manifestfiles*, fglfiles?, formfiles?,
 msgfiles* )>
        <!ATTLIST conversion project CDATA #REQUIRED >
      <!ATTLIST conversion type CDATA  #FIXED "application" >
    <!ATTLIST conversion locale CDATA #IMPLIED >
      <!ATTLIST conversion cursor (local | global) #IMPLIED>
    <!ATTLIST conversion defaultserver CDATA #IMPLIED>
   <!ELEMENT rootdir (fgldir?,egldir,artifactsdir?)>
   <!ELEMENT fgldir (#PCDATA)>
   <!ELEMENT egldir (#PCDATA)>
   <!ELEMENT artifactsdir (#PCDATA)>
   <!ELEMENT manifestfiles (file)+>
    <!ATTLIST manifestfiles type ( schema | library) #REQUIRED>
   <!ELEMENT fglfiles (file)+>
```

```
   <!ELEMENT formfiles (file)+>
   <!ELEMENT fontconfigfile (file)>
   <!ELEMENT file (#PCDATA)>
   <!ELEMENT msgfiles (file)+>
    <!ATTLIST msgfiles locale CDATA #IMPLIED >
]>
<conversion  project="i4gldemo"   type="application"  locale="en_US.8859-1"
 defaultserver="myserver">

 <rootdir>
    <fgldir>C:\i4gl\src\i4gldemo</fgldir>
    <egldir>C:\egl\src\i4gldemo</egldir>
    <artifactsdir></artifactsdir>
 </rootdir>


 <manifestfiles type="schema">
    <file>C:/converted_projects/stores7/ConversionArtifacts/manifest/
Stores7Manifest.xml</file>
</manifestfiles>

<manifestfiles type="library">
    <file> C:\egl\SharedLibraryProject/ConversionArtifacts/manifest/
Stores7Manifest.xml</file>
</manifestfiles>


    <fglfiles>
        <file>d4_cust.4gl</file>
        <file>d4_demo.4gl</file>
        <file>d4_globals.4gl</file>
        <file>d4_load.4gl</file>
        <file>d4_main.4gl</file>
        <file>d4_orders.4gl</file>
        <file>d4_report.4gl</file>
        <file>d4_stock.4gl</file>
    </fglfiles>

    <formfiles>
        <file>forms\cust.per</file>
        <file>forms\custcur.per</file>
        <file>forms\custform.per</file>
        <file>forms\customer.per</file>
        <file>forms\ordcur.per</file>
        <file>forms\order.per</file>
        <file>forms\orderform.per</file>
        <file>forms\p_ordcur.per</file>
        <file>forms\state_list.per</file>
        <file>forms\stock1.per</file>
        <file>forms\stock_sel.per</file>
    </formfiles>

      <!-- example for converting UJIS message files -->
      <msgfiles locale="en_US.8859-1">
        <file>C:/i4gl/src/msg/en/0333/orders.msg</file>
        <file>C:/i4gl/src/msg/en/0333/customer.msg</file>
    </msgfiles>

    <msgfiles locale="ja_jp.UJIS">
        <file>C:/i4gl/src/msg/jp/ujis/orders.msg</file>
        <file>C:/i4gl/src/msg/jp/ujis/customer.msg</file>
    </msgfiles>
</conversion>
```

Appendix E. Manifest File Examples

In This Appendix

This appendix provides template configuration files for Database Schema Extraction, Shared Library, and I4GL Application projects.

Note: The examples below use the **stores7** database.

Database Schema Extraction Project Manifest File Example

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE manifest [
    <!ELEMENT manifest (package*)>
        <!ATTLIST manifest project CDATA #REQUIRED >
        <!ATTLIST manifest type CDATA #FIXED "schema">
        <!ATTLIST manifest version CDATA #REQUIRED >
    <!ELEMENT package (table+)>
        <!ATTLIST package
            name CDATA #REQUIRED
            server CDATA #REQUIRED
            database CDATA #REQUIRED
            mode (ANSI) #IMPLIED>
    <!ELEMENT table (column+)>
        <!ATTLIST table
                  name CDATA #REQUIRED
            egltype CDATA #REQUIRED
                  owner CDATA #IMPLIED>
    <!ELEMENT column EMPTY>
        <!ATTLIST column
            name CDATA #REQUIRED
            dataitem CDATA #REQUIRED
            fgltype CDATA #REQUIRED
                  egltype CDATA #REQUIRED
            size CDATA #IMPLIED
            start CDATA #IMPLIED
            end CDATA #IMPLIED
            precision CDATA #IMPLIED
            scale CDATA #IMPLIED>
]>

<!--
    Database schema manifest file generated by I4GL to EGL Conversion Utility
    Project Name : stores7
    Extracted on : Fri Jan 28 13:46:07 CST 2005
-->

<manifest project ="stores7" type="schema" version="1.0.0">
    <!--  Database     : stores7  -->
    <package name="stores7.myserver.stores7" server="myserver"
database="stores7" >
        <table name ="call_type" egltype ="rec_like_call_type">
            <column name ="call_code"
dataitem ="dataitem_like_call_type_call_code"
 fgltype ="char" egltype ="unicode(1)" size ="1" />
            <column name ="code_descr"
dataitem ="dataitem_like_call_type_code_descr"
 fgltype ="char" egltype ="unicode(30)" size ="30" />
        </table>

        <table name ="catalog" egltype ="rec_like_catalog">
            <column name ="catalog_num"
```

```
dataitem ="dataitem_like_catalog_catalog_num"
 fgltype ="serial" egltype ="int" />
            <column name ="stock_num"
dataitem ="dataitem_like_catalog_stock_num"
 fgltype ="smallint" egltype ="smallint" />
            <column name ="manu_code"
dataitem ="dataitem_like_catalog_manu_code"
 fgltype ="char" egltype ="unicode(3)" size ="3" />
            <column name ="cat_descr"
dataitem ="dataitem_like_catalog_cat_descr"
 fgltype ="text" egltype ="clob" />
            <column name ="cat_picture"
dataitem ="dataitem_like_catalog_cat_picture"
 fgltype ="byte" egltype ="blob" />
            <column name ="cat_advert"
dataitem ="dataitem_like_catalog_cat_advert"
 fgltype ="varchar" egltype ="string" size ="255" />
        </table>

        <table name ="cust_calls" egltype ="rec_like_cust_calls">
            <column name ="customer_num" dataitem =
"dataitem_like_cust_calls_customer_num" fgltype ="int"
egltype ="int" />
            <column name ="call_dtime"
dataitem ="dataitem_like_cust_calls_call_dtime"
 fgltype ="datetime year to minute" egltype ="timestamp
(&quot;yyyyMMddhhmm&quot;)" start ="year"
end ="minute" />
            <column name ="user_id"
dataitem ="dataitem_like_cust_calls_user_id"
 fgltype ="char" egltype ="unicode(32)" size ="32" />
            <column name ="call_code"
dataitem ="dataitem_like_cust_calls_call_code"
 fgltype ="char" egltype ="unicode(1)" size ="1" />
            <column name ="call_descr"
dataitem ="dataitem_like_cust_calls_call_descr"
 fgltype ="char" egltype ="unicode(240)" size ="240" />
            <column name ="res_dtime"
dataitem ="dataitem_like_cust_calls_res_dtime"
 fgltype ="datetime year to minute" egltype ="timestamp
(&quot;yyyyMMddhhmm&quot;)" start ="year"
 end ="minute" />
            <column name ="res_descr"
dataitem ="dataitem_like_cust_calls_res_descr"
 fgltype ="char" egltype ="unicode(240)" size ="240" />
        </table>

        <table name ="customer" egltype ="rec_like_customer">
            <column name ="customer_num"
dataitem ="dataitem_like_customer_customer_num"
fgltype ="serial" egltype ="int" />
            <column name ="fname"
dataitem ="dataitem_like_customer_fname"
  fgltype ="char" egltype ="unicode(15)" size ="15" />
            <column name ="lname"
dataitem ="dataitem_like_customer_lname"
 fgltype ="char" egltype ="unicode(15)" size ="15" />
            <column name ="company"
dataitem ="dataitem_like_customer_company"
fgltype ="char" egltype ="unicode(20)" size ="20" />
            <column name ="address1"
dataitem ="dataitem_like_customer_address1"
 fgltype ="char" egltype ="unicode(20)" size ="20" />
            <column name ="address2"
dataitem ="dataitem_like_customer_address2"
 fgltype ="char" egltype ="unicode(20)" size ="20" />
            <column name ="city"
```

```
dataitem ="dataitem_like_customer_city" fgltype ="char"
  egltype ="unicode(15)" size ="15" />
            <column name ="state"
dataitem ="dataitem_like_customer_state" fgltype ="char"
  egltype ="unicode(2)" size ="2" />
            <column name ="zipcode"
dataitem ="dataitem_like_customer_zipcode"
 fgltype ="char" egltype ="unicode(5)" size ="5" />
            <column name ="phone"
dataitem ="dataitem_like_customer_phone"
 fgltype ="char" egltype ="unicode(18)" size ="18" />
        </table>

        <table name ="items" egltype ="rec_like_items">
            <column name ="item_num"
dataitem ="dataitem_like_items_item_num"
 fgltype ="smallint" egltype ="smallint" />
            <column name ="order_num"
dataitem ="dataitem_like_items_order_num"
 fgltype ="int" egltype ="int" />
            <column name ="stock_num"
dataitem ="dataitem_like_items_stock_num"
 fgltype ="smallint" egltype ="smallint" />
            <column name ="manu_code"
dataitem ="dataitem_like_items_manu_code"
 fgltype ="char" egltype ="unicode(3)" size ="3" />
            <column name ="quantity"
dataitem ="dataitem_like_items_quantity"
 fgltype ="smallint" egltype ="smallint" />
            <column name ="total_price"
dataitem ="dataitem_like_items_total_price"
 fgltype ="money" egltype ="money(8,2)"
scale ="2" precision ="8" />
        </table>

        <table name ="manufact" egltype ="rec_like_manufact">
            <column name ="manu_code"
dataitem ="dataitem_like_manufact_manu_code"
 fgltype ="char" egltype ="unicode(3)" size ="3" />
            <column name ="manu_name"
dataitem ="dataitem_like_manufact_manu_name"
 fgltype ="char" egltype ="unicode(15)" size ="15" />
            <column name ="lead_time"
dataitem ="dataitem_like_manufact_lead_time"
 fgltype ="interval day(3) to day"
egltype ="interval(&quot;ddd&quot;)" precision ="3"
start ="day" end ="day" />
        </table>

        <table name ="msgs" egltype ="rec_like_msgs">
            <column name ="lang" dataitem ="dataitem_like_msgs_lang"
 fgltype ="char" egltype ="unicode(32)" size ="32" />
            <column name ="number" dataitem ="dataitem_like_msgs_number"
 fgltype ="int" egltype ="int" />
            <column name ="message" dataitem ="dataitem_like_msgs_message"
 fgltype ="nchar" egltype ="unicode(255)" size ="255" />
        </table>

        <table name ="orders" egltype ="rec_like_orders">
            <column name ="order_num"
dataitem ="dataitem_like_orders_order_num"
 fgltype ="serial" egltype ="int" />
            <column name ="order_date"
dataitem ="dataitem_like_orders_order_date"
 fgltype ="date" egltype ="date" />
            <column name ="customer_num"
dataitem ="dataitem_like_orders_customer_num"
```

```
 fgltype ="int" egltype ="int" />
            <column name ="ship_instruct"
dataitem ="dataitem_like_orders_ship_instruct"
 fgltype ="char" egltype ="unicode(40)" size ="40" />
            <column name ="backlog"
 dataitem ="dataitem_like_orders_backlog"
 fgltype ="char" egltype ="unicode(1)" size ="1" />
            <column name ="po_num"
 dataitem ="dataitem_like_orders_po_num"
 fgltype ="char" egltype ="unicode(10)" size ="10" />
            <column name ="ship_date"
 dataitem ="dataitem_like_orders_ship_date"
 fgltype ="date" egltype ="date" />
            <column name ="ship_weight"
dataitem ="dataitem_like_orders_ship_weight"
 fgltype ="decimal" egltype ="decimal(8,2)" scale ="2"
precision ="8" />
            <column name ="ship_charge"
dataitem ="dataitem_like_orders_ship_charge"
 fgltype ="money" egltype ="money(6,2)" scale ="2"
precision ="6" />
            <column name ="paid_date"
dataitem ="dataitem_like_orders_paid_date"
 fgltype ="date" egltype ="date" />
        </table>

        <table name ="state" egltype ="rec_like_state">
            <column name ="code" dataitem ="dataitem_like_state_code"
fgltype ="char" egltype ="unicode(2)" size ="2" />
            <column name ="sname" dataitem ="dataitem_like_state_sname"
fgltype ="char" egltype ="unicode(15)" size ="15" />
        </table>

        <table name ="stock" egltype ="rec_like_stock">
            <column name ="stock_num"
dataitem ="dataitem_like_stock_stock_num"
 fgltype ="smallint" egltype ="smallint" />
            <column name ="manu_code"
dataitem ="dataitem_like_stock_manu_code"
 fgltype ="char" egltype ="unicode(3)" size ="3" />
            <column name ="description"
dataitem ="dataitem_like_stock_description"
 fgltype ="char" egltype ="unicode(15)" size ="15" />
            <column name ="unit_price"
dataitem ="dataitem_like_stock_unit_price"
 fgltype ="money" egltype ="money(6,2)" scale ="2"
precision ="6" />
            <column name ="unit"
dataitem ="dataitem_like_stock_unit"
fgltype ="char" egltype ="unicode(4)" size ="4" />
            <column name ="unit_descr"
dataitem ="dataitem_like_stock_unit_descr"
 fgltype ="char" egltype ="unicode(15)" size ="15" />
        </table>

        <table name ="syscolval" egltype ="rec_like_syscolval">
            <column name ="tabname"
dataitem ="dataitem_like_syscolval_tabname"
 fgltype ="char" egltype ="unicode(18)" size ="18" />
            <column name ="colname"
dataitem ="dataitem_like_syscolval_colname"
 fgltype ="char" egltype ="unicode(18)" size ="18" />
            <column name ="attrname"
dataitem ="dataitem_like_syscolval_attrname"
 fgltype ="char" egltype ="unicode(10)" size ="10" />
            <column name ="attrval"
dataitem ="dataitem_like_syscolval_attrval"
```

```
 fgltype ="char" egltype ="unicode(64)" size ="64" />
        </table>
    </package>
</manifest>
```

Shared Library or Application Project Manifest File Example

This example shows a shared library manifest file. Shared library and application projects have identical manifest files, with the following exception:

- in the shared library project manifest file, the attribute **type** equals **library**.
- in the application project manifest file, the attribute **type** equals **application**.

```
<?xml version="1.0" encoding="utf-8"?>
<!--
    Manifest file generated by I4GL to EGL Conversion Utility
    Project Name  :I4gldemoSharedLibrary
    Generated  on :Fri Jan 28 14:01:36 CST 2005
-->

<!-- DTD for Manifest file -->
<!DOCTYPE manifest [
<!ELEMENT manifest (package)>
   <!ATTLIST manifest project CDATA #REQUIRED>
   <!ATTLIST manifest type CDATA #FIXED "library">
<!ELEMENT package (rectype*, variables*,
function*,cfunc*,cursor*,
preparedStatements*)>
   <!ATTLIST package name CDATA #REQUIRED>
<!ELEMENT function (parameter*,return*)>
   <!ATTLIST function name CDATA #REQUIRED>
   <!ATTLIST function package CDATA #IMPLIED>
   <!ATTLIST function library CDATA #REQUIRED>
   <!ATTLIST function type CDATA #REQUIRED>
<!ELEMENT parameter EMPTY>
   <!ATTLIST parameter name CDATA #REQUIRED>
   <!ATTLIST parameter egltype CDATA #REQUIRED>
   <!ATTLIST parameter fgltype CDATA #IMPLIED>
   <!ATTLIST parameter size CDATA #IMPLIED>
   <!ATTLIST parameter precision CDATA #IMPLIED>
   <!ATTLIST parameter scale CDATA #IMPLIED>
   <!ATTLIST parameter start CDATA #IMPLIED>
   <!ATTLIST parameter end CDATA #IMPLIED>
   <!ATTLIST parameter isrectype (t|f) "f">
   <!ATTLIST parameter library CDATA #REQUIRED>
<!ELEMENT return EMPTY>
   <!ATTLIST return name CDATA #REQUIRED>
   <!ATTLIST return egltype CDATA #REQUIRED>
   <!ATTLIST return fgltype CDATA #IMPLIED>
   <!ATTLIST return size CDATA #IMPLIED>
   <!ATTLIST return precision CDATA #IMPLIED>
   <!ATTLIST return scale CDATA #IMPLIED>
   <!ATTLIST return start CDATA #IMPLIED>
   <!ATTLIST return end CDATA #IMPLIED>
   <!ATTLIST return isrectype (t|f) "f" >
   <!ATTLIST return library CDATA #REQUIRED>
<!ELEMENT cfunc (dependentPackage*) >
   <!ATTLIST cfunc name CDATA #REQUIRED>
   <!ATTLIST cfunc package CDATA #REQUIRED>
   <!ATTLIST cfunc library CDATA #REQUIRED>
   <!ATTLIST cfunc argcount CDATA #REQUIRED>
   <!ATTLIST cfunc retcount CDATA #REQUIRED>
<!ELEMENT dependentPackage EMPTY>
   <!ATTLIST dependentPackage package CDATA #REQUIRED>
<!ELEMENT cursor EMPTY>
   <!ATTLIST cursor name CDATA #REQUIRED>
   <!ATTLIST cursor ishold (t|f) "f" >
```

```
   <!ATTLIST cursor isscrolling (t|f) "f" >
   <!ATTLIST cursor library CDATA #REQUIRED>
<!ELEMENT variables (variable*)>
<!ELEMENT variable EMPTY>
   <!ATTLIST variable name CDATA #REQUIRED>
   <!ATTLIST variable egltype CDATA #REQUIRED>
   <!ATTLIST variable fgltype CDATA #IMPLIED>
   <!ATTLIST variable size CDATA #IMPLIED>
   <!ATTLIST variable precision CDATA #IMPLIED>
   <!ATTLIST variable scale CDATA #IMPLIED>
   <!ATTLIST variable start CDATA #IMPLIED>
   <!ATTLIST variable end CDATA #IMPLIED>
   <!ATTLIST variable isrectype (t|f) "f" >
   <!ATTLIST variable library CDATA #REQUIRED>
<!ELEMENT rectype (field*)>
   <!ATTLIST rectype name CDATA #REQUIRED>
   <!ATTLIST rectype library CDATA #REQUIRED>
<!ELEMENT field  EMPTY>
   <!ATTLIST field name CDATA #REQUIRED>
   <!ATTLIST field egltype CDATA #REQUIRED>
   <!ATTLIST field fgltype CDATA #IMPLIED>
   <!ATTLIST field size CDATA #IMPLIED>
   <!ATTLIST field precision CDATA #IMPLIED>
   <!ATTLIST field scale CDATA #IMPLIED>
   <!ATTLIST field start CDATA #IMPLIED>
   <!ATTLIST field end CDATA #IMPLIED>
   <!ATTLIST field isrectype (t|f) "f" >
   <!ATTLIST field library CDATA #REQUIRED>
<!ELEMENT preparedStatements (statement*)>
<!ELEMENT statement EMPTY>
   <!ATTLIST statement name CDATA #REQUIRED>
   <!ATTLIST statement library CDATA #REQUIRED>
]>

<manifest project="I4gldemoSharedLibrary" type="library" >
<package name="I4gldemoSharedLibrary">
<!-- Record Declarations-->
<rectype name="recordtype_d4_orders_invoice_x_invoice"
library="d4_orders">
    <field name="order_num" library="d4_orders" egltype="int"
 fgltype="serial" isrectype="f" />
    <field name="order_date" library="d4_orders" egltype="date"
 fgltype="date" isrectype="f" />
    <field name="ship_instruct" library="d4_orders"
egltype="unicode(40)"
  fgltype="char" size="40" isrectype="f" />
    <field name="backlog" library="d4_orders" egltype="unicode(1)"
 fgltype="char" size="1" isrectype="f" />
    <field name="po_num" library="d4_orders" egltype="unicode(10)"
 fgltype="char" size="10" isrectype="f" />
    <field name="ship_date" library="d4_orders" egltype="date"
 fgltype="date" isrectype="f" />
    <field name="ship_weight" library="d4_orders"
egltype="decimal(8,2)" fgltype="decimal" precision="8"
scale="2" isrectype="f" />
    <field name="ship_charge" library="d4_orders"
egltype="money(6,2)" fgltype="money" precision="6"
scale="2" isrectype="f" />
    <field name="item_num" library="d4_orders" egltype="smallint"
 fgltype="smallint" isrectype="f" />
    <field name="stock_num" library="d4_orders" egltype="smallint"
fgltype="smallint" isrectype="f" />
    <field name="manu_code" library="d4_orders" egltype="unicode(3)"
 fgltype="char" size="3" isrectype="f" />
    <field name="quantity" library="d4_orders" egltype="smallint"
fgltype="smallint" isrectype="f" />
    <field name="total_price" library="d4_orders" egltype="money(8,2)"
```

```
fgltype="money" precision="8" scale="2" isrectype="f" />
    <field name="description" library="d4_orders" egltype="unicode(15)"
fgltype="char" size="15" isrectype="f" />
    <field name="unit_price" library="d4_orders" egltype="money(6,2)"
fgltype="money" precision="6" scale="2" isrectype="f" />
    <field name="unit" library="d4_orders" egltype="unicode(4)"
fgltype="char" size="4" isrectype="f" />
    <field name="unit_descr" library="d4_orders" egltype="unicode(15)"
fgltype="char" size="15" isrectype="f" />
    <field name="manu_name" library="d4_orders" egltype="unicode(15)"
fgltype="char" size="15" isrectype="f" />
</rectype>
<rectype name="recordtype_d4_orders_x" library="d4_orders">
    <field name="order_num" library="d4_orders" egltype="int"
fgltype="serial" isrectype="f" />
    <field name="order_date" library="d4_orders" egltype="date"
 fgltype="date" isrectype="f" />
    <field name="ship_instruct" library="d4_orders"
egltype="unicode(40)" fgltype="char" size="40" isrectype="f" />
    <field name="backlog" library="d4_orders" egltype="unicode(1)"
 fgltype="char" size="1" isrectype="f" />
    <field name="po_num" library="d4_orders" egltype="unicode(10)"
 fgltype="char" size="10" isrectype="f" />
    <field name="ship_date" library="d4_orders" egltype="date"
fgltype="date" isrectype="f" />
    <field name="ship_weight" library="d4_orders" egltype="decimal(8,2)"
fgltype="decimal" precision="8" scale="2" isrectype="f" />
    <field name="ship_charge" library="d4_orders" egltype="money(6,2)"
fgltype="money" precision="6" scale="2" isrectype="f" />
    <field name="item_num" library="d4_orders" egltype="smallint"
fgltype="smallint" isrectype="f" />
    <field name="stock_num" library="d4_orders" egltype="smallint"
fgltype="smallint" isrectype="f" />
    <field name="manu_code" library="d4_orders" egltype="unicode(3)"
 fgltype="char" size="3" isrectype="f" />
    <field name="quantity" library="d4_orders" egltype="smallint"
fgltype="smallint" isrectype="f" />
    <field name="total_price" library="d4_orders" egltype="money(8,2)"
fgltype="money" precision="8" scale="2" isrectype="f" />
    <field name="description" library="d4_orders" egltype="unicode(15)"
fgltype="char" size="15" isrectype="f" />
    <field name="unit_price" library="d4_orders" egltype="money(6,2)"
fgltype="money" precision="6" scale="2" isrectype="f" />
    <field name="unit" library="d4_orders" egltype="unicode(4)"
fgltype="char" size="4" isrectype="f" />
    <field name="unit_descr" library="d4_orders" egltype="unicode(15)"
fgltype="char" size="15" isrectype="f" />
    <field name="manu_name" library="d4_orders" egltype="unicode(15)"
fgltype="char" size="15" isrectype="f" />
</rectype>
<rectype name="recordtype_d4_report_print_ar_r" library="d4_report">
    <field name="customer_num" library="d4_report" egltype="int"
 fgltype="serial" isrectype="f" />
    <field name="fname" library="d4_report" egltype="unicode(15)"
 fgltype="char" size="15" isrectype="f" />
    <field name="lname" library="d4_report" egltype="unicode(15)"
 fgltype="char" size="15" isrectype="f" />
    <field name="company" library="d4_report" egltype="unicode(20)"
 fgltype="char" size="20" isrectype="f" />
    <field name="order_num" library="d4_report" egltype="int"
 fgltype="serial" isrectype="f" />
    <field name="order_date" library="d4_report" egltype="date"
 fgltype="date" isrectype="f" />
    <field name="ship_date" library="d4_report" egltype="date"
 fgltype="date" isrectype="f" />
    <field name="paid_date" library="d4_report" egltype="date"
 fgltype="date" isrectype="f" />
```

```
    <field name="total_price" library="d4_report" egltype="money(8,2)"
fgltype="money" precision="8" scale="2" isrectype="f" />
</rectype>
<rectype name="recordtype_d4_report_r" library="d4_report">
    <field name="customer_num" library="d4_report" egltype="int"
 fgltype="serial" isrectype="f" />
    <field name="fname" library="d4_report" egltype="unicode(15)"
 fgltype="char" size="15" isrectype="f" />
    <field name="lname" library="d4_report" egltype="unicode(15)"
 fgltype="char" size="15" isrectype="f" />
    <field name="company" library="d4_report" egltype="unicode(20)"
 fgltype="char" size="20" isrectype="f" />
    <field name="order_num" library="d4_report" egltype="int"
 fgltype="serial" isrectype="f" />
    <field name="order_date" library="d4_report" egltype="date"
 fgltype="date" isrectype="f" />
    <field name="ship_date" library="d4_report" egltype="date"
 fgltype="date" isrectype="f" />
    <field name="paid_date" library="d4_report" egltype="date"
 fgltype="date" isrectype="f" />
    <field name="total_price" library="d4_report" egltype="money(8,2)"
fgltype="money" precision="8" scale="2" isrectype="f" />
</rectype>
<rectype name="recordtype_p_items" library="d4_globals">
    <field name="item_num" library="d4_globals" egltype="smallint"
fgltype="smallint" isrectype="f" />
    <field name="stock_num" library="d4_globals" egltype="smallint"
fgltype="smallint" isrectype="f" />
    <field name="manu_code" library="d4_globals" egltype="unicode(3)"
fgltype="char" size="3" isrectype="f" />
    <field name="description" library="d4_globals" egltype="unicode(15)"
fgltype="char" size="15" isrectype="f" />
    <field name="quantity" library="d4_globals" egltype="smallint"
fgltype="smallint" isrectype="f" />
    <field name="unit_price" library="d4_globals" egltype="money(6,2)"
fgltype="money" precision="6" scale="2" isrectype="f" />
    <field name="total_price" library="d4_globals" egltype="money(8,2)"
fgltype="money" precision="8" scale="2" isrectype="f" />
</rectype>
<rectype name="recordtype_p_orders" library="d4_globals">
    <field name="order_num" library="d4_globals" egltype="int"
 fgltype="serial" isrectype="f" />
    <field name="order_date" library="d4_globals" egltype="date"
 fgltype="date" isrectype="f" />
    <field name="po_num" library="d4_globals" egltype="unicode(10)"
 fgltype="char" size="10" isrectype="f" />
    <field name="ship_instruct" library="d4_globals"
egltype="unicode(40)" fgltype="char" size="40" isrectype="f" />
</rectype>
<rectype name="recordtype_p_stock" library="d4_globals">
    <field name="stock_num" library="d4_globals" egltype="smallint"
fgltype="smallint" isrectype="f" />
    <field name="manu_code" library="d4_globals" egltype="unicode(3)"
fgltype="char" size="3" isrectype="f" />
    <field name="manu_name" library="d4_globals" egltype="unicode(15)"
fgltype="char" size="15" isrectype="f" />
    <field name="description" library="d4_globals" egltype="unicode(15)"
fgltype="char" size="15" isrectype="f" />
    <field name="unit_price" library="d4_globals" egltype="money(6,2)"
fgltype="money" precision="6" scale="2" isrectype="f" />
    <field name="unit_descr" library="d4_globals" egltype="unicode(15)"
fgltype="char" size="15" isrectype="f" />
</rectype>

<!-- Global Variable Declarations-->
<variables>
    <variable name="p_customer" library="d4_globals"
```

```
 egltype="rec_like_customer" isrectype="t" />
    <variable name="p_items" library="d4_globals"
 egltype="recordtype_p_items" isrectype="t" />
    <variable name="p_orders" library="d4_globals"
 egltype="recordtype_p_orders" isrectype="t" />
    <variable name="p_state" library="d4_globals"
egltype="rec_like_state" isrectype="t" />
    <variable name="p_stock" library="d4_globals"
egltype="recordtype_p_stock" isrectype="t" />
    <variable name="print_option" library="d4_globals"
egltype="UNICODE(1)" fgltype="CHAR(1)" size="1"
isrectype="f" />
    <variable name="state_cnt" library="d4_globals" egltype="INT"
 fgltype="INTEGER" isrectype="f" />
    <variable name="stock_cnt" library="d4_globals" egltype="INT"
 fgltype="INTEGER" isrectype="f" />
</variables>

<!-- Function Declarations-->

<function name="add_customer" package="i4gldemo" library="d4_cust"
 type="void" >
    <parameter name="repeat" library="d4_cust" egltype="INT"
fgltype="INTEGER" size="" start="" end="" precision="" scale=""
isrectype="f" />
</function>

<function name="add_order" package="i4gldemo" library="d4_orders"
type="void" >
</function>

<function name="add_stock" package="i4gldemo" library="d4_stock"
type="void" >
</function>

<function name="bang" package="i4gldemo" library="d4_main"
type="void" >
</function>

<function name="clear_menu" package="i4gldemo" library="d4_main"
type="void" >
</function>

<function name="customer" package="i4gldemo" library="d4_cust"
type="void" >
</function>

<function name="customer_help" package="i4gldemo" l
ibrary="d4_cust"
type="void" >
</function>

<function name="delete_customer" package="i4gldemo"
library="d4_cust" type="void" >
</function>

<function name="delete_order" package="i4gldemo"
library="d4_orders" type="void" >
</function>

<function name="delete_stock" package="i4gldemo"
library="d4_stock"
type="void" > </function>

<function name="demo" package="i4gldemo"
library="d4_demo" type="void" >
</function>
```

```
<function name="get_item" package="i4gldemo" library="d4_orders"
type="void" >
</function>

<function name="get_order" package="i4gldemo" library="d4_orders"
type="void" >
</function>

<function name="get_states" package="i4gldemo" library="d4_main"
type="void" >
</function>

<function name="get_stock" package="i4gldemo" library="d4_orders"
 type="void" >
    <return name="stock_num" egltype="smallint"
library="d4_globals" fgltype="smallint" size="" start="" end=""
precision="" scale="" isrectype="f" />
    <return name="manu_code" egltype="unicode(3)"
library="d4_globals" fgltype="char" size="3" start=""
end="" precision="" scale="" isrectype="f" />
    <return name="description" egltype="unicode(15)"
library="d4_globals" fgltype="char" size="15" start=""
end="" precision=""scale="" isrectype="f" />
    <return name="unit_price" egltype="money(6,2)"
library="d4_globals" fgltype="money" size="" start=""
end="" precision="6" scale="2" isrectype="f" />
</function>

<function name="get_stocks" package="i4gldemo" library="d4_main"
type="void" >
</function>

<function name="input_cust" package="i4gldemo" library="d4_cust"
type="Int" >
    <return name="" egltype="Int" library="" fgltype="Int" size="1"
start="TRUE" end="TRUE" precision="" scale="" isrectype="f" />
</function>

<function name="insert_items" package="i4gldemo" library="d4_orders"
type="Int" >
    <return name="" egltype="Int" library="" fgltype="Int" size="1"
start="TRUE" end="TRUE" precision="" scale="" isrectype="f" />
</function>

<function name="invoice" package="i4gldemo" library="d4_orders"
type="void" >
    <parameter name="file_name" library="d4_orders"
egltype="UNICODE(20)" fgltype="CHAR(20)" size="20" start=""
end="" precision="" scale="" isrectype="f" />
</function>

<function name="item_total" package="i4gldemo" library="d4_orders"
type="void" >
</function>

<function name="MAIN" package="i4gldemo" library="d4_main"
type="void" >
</function>

<function name="mess" package="i4gldemo" library="d4_main"
type="void" > <parameter name="str" library="d4_main"
egltype="UNICODE(80)" fgltype="CHAR(80)" size="80" start=""
end="" precision="" scale="" isrectype="f" />
    <parameter name="mrow" library="d4_main" egltype="SMALLINT"
fgltype="SMALLINT" size="" start="" end="" precision=""
scale="" isrectype="f" />
```

```
</function>

<function name="order_total" package="i4gldemo" library="d4_orders"
type="void" >
</function>

<function name="orders" package="i4gldemo" library="d4_orders"
type="void" >
</function>

<function name="print_ar" package="i4gldemo" library="d4_report"
type="void" >
</function>

<function name="print_backlog" package="i4gldemo" library="d4_report"
 type="void" >
</function>

<function name="print_labels" package="i4gldemo" library="d4_report"
 type="void" >
</function>

<function name="print_stock" package="i4gldemo" library="d4_report"
type="void" >
</function>

<function name="query_customer" package="i4gldemo" library="d4_cust"
type="Int" >
    <parameter name="mrow" library="d4_cust" egltype="SMALLINT"
fgltype="SMALLINT" size="" start="" end="" precision="" scale=""
isrectype="f" />
    <return name="" egltype="Int" library="" fgltype="Int" size="1"
start="TRUE" end="TRUE" precision="" scale="" isrectype="f" />
</function>

<function name="query_stock" package="i4gldemo" library="d4_stock"
type="void" >
</function>

<function name="renum_items" package="i4gldemo" library="d4_orders"
type="void" >
</function>

<function name="REPORT" package="i4gldemo" library="d4_report"
type="void" >
    <parameter name="r" library="d4_report" egltype="recordtype_d4_report_r"
fgltype="RECORD customer_num LIKE customer.customer_num, fname LIKE
 customer.fname, lname LIKE customer.lname,
company LIKE customer.company, order_num LIKE orders.order_num,
order_date LIKE orders.order_date, ship_date LIKE orders.ship_date,
paid_date LIKE orders.paid_date,
total_price LIKE items.total_price END RECORD" size="" start=""
end="" precision="" scale="" isrectype="t" />
</function>

<function name="reports" package="i4gldemo" library="d4_report"
type="void" >
</function>

<function name="ring_menu" package="i4gldemo" library="d4_main"
type="void" >
</function>

<function name="statehelp" package="i4gldemo" library="d4_cust"
type="void" >
</function>
```

```
<function name="stock" package="i4gldemo" library="d4_stock"
type="void" >
</function>

<function name="unring_menu" package="i4gldemo" library="d4_main"
type="void" >
</function>

<function name="update_customer" package="i4gldemo"
library="d4_cust" type="void" > </function>

<function name="update_options" package="i4gldemo" library="d4_report"
 type="void" >
</function>

<function name="update_order" package="i4gldemo" library="d4_orders"
 type="void" >
</function>

<function name="update_stock" package="i4gldemo" library="d4_stock"
type="void" >
</function>
</package>
<forms>
    <form name="custForm.egl" package="i4gldemo.forms" />
    <form name="custcurForm.egl" package="i4gldemo.forms" />
    <form name="custformForm.egl" package="i4gldemo.forms" />
    <form name="customerForm.egl" package="i4gldemo.forms" />
    <form name="ordcurForm.egl" package="i4gldemo.forms" />
    <form name="orderForm.egl" package="i4gldemo.forms" />
    <form name="orderformForm.egl" package="i4gldemo.forms" />
    <form name="p_ordcurForm.egl" package="i4gldemo.forms" />
    <form name="state_listForm.egl" package="i4gldemo.forms" />
    <form name="stock1Form.egl" package="i4gldemo.forms" />
    <form name="stock_selForm.egl" package="i4gldemo.forms" />
</forms>
</manifest>
```

Appendix F. DTD Examples

In This Appendix

This appendix provides examples of the DTD used for configuration and manifest files for Database Schema Extraction, Shared Library, and Application projects.

DTD Example Overview

The DTD used in these examples is specific to the Informix 4GL to EGL Conversion Utility Version 6.0.0.1. These examples are located in the following directory: ***plugininstallation*/etc/dtd**.

DTD examples are provided for the following:

- Configuration file
  - Schema
  - Library
  - Application
- Manifest file
  - Schema
  - Library
  - Application

Configuration File

Database Schema Extraction Project

In the following example, the Artifacts Directory specified by element **artifactsdir** is optional. If no value is provided, the directory will default to the ***egldir*/ConversionArtifacts** directory. In addition, the **client_locale** and **db_locale** attributes are Informix-specific locales which default to **English** and **en_US.8859-1**.

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT conversion (rootdir,dbconnection*)>
    <!ATTLIST conversion project CDATA #REQUIRED >
    <!ATTLIST conversion type  CDATA #FIXED "schema" >

<!ELEMENT rootdir (egldir,artifactsdir?)>
<!ELEMENT egldir (#PCDATA)>
<!ELEMENT artifactsdir (#PCDATA)>

<!ELEMENT dbconnection (database,server,host,port,user,password)+>
      <!ATTLIST dbconnection extractSystemTables  (yes|no) "no" >
      <!ATTLIST dbconnection client_locale CDATA #IMPLIED >
      <!ATTLIST dbconnection db_locale  CDATA #IMPLIED >
<!ELEMENT database (#PCDATA)>
<!ELEMENT server (#PCDATA)>
<!ELEMENT host (#PCDATA)>
<!ELEMENT port (#PCDATA)>
<!ELEMENT user (#PCDATA)>
<!ELEMENT password ANY>
```

Shared Library Project

In the following example, the **locale** attribute used in the conversion" and "msgfiles" elements both default to **en_US.8859-1**. If Library Project is dependent on database schema then "defaultserver" attribute should contain the name of the server as specified in dependent manifest file given in element "manifestfiles" of type "schema"

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT conversion ( rootdir,    manifestfiles*, fglfiles?,formfiles?,
msgfiles* )>
    <!ATTLIST conversion project CDATA #REQUIRED >
    <!ATTLIST conversion type CDATA #FIXED "library" >
    <!ATTLIST conversion locale CDATA #IMPLIED >
      <!ATTLIST conversion cursor (local | global) #IMPLIED>
    <!ATTLIST conversion defaultserver CDATA #IMPLIED>

<!ELEMENT rootdir (fgldir?,egldir,artifactsdir?)>
<!ELEMENT fgldir (#PCDATA)>
<!ELEMENT egldir (#PCDATA)>
<!ELEMENT artifactsdir (#PCDATA)>

<!ELEMENT manifestfiles (file)+>
    <!ATTLIST manifestfiles type ( schema | library) #REQUIRED>

<!ELEMENT fglfiles (file)+>
<!ELEMENT formfiles (file)+>
<!ELEMENT fontconfigfile (file)>
<!ELEMENT file (#PCDATA)>

<!ELEMENT msgfiles (file)+>
    <!ATTLIST msgfiles locale CDATA #IMPLIED >
```

Application Project

This DTD is identical to Library project, except for value of attribute "type" under element "conversion".

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT conversion ( rootdir, manifestfiles*, fglfiles?, formfiles?
 msgfiles* )>
        <!ATTLIST conversion project CDATA #REQUIRED >
<!ATTLIST conversion type CDATA #FIXED "application" >
    <!ATTLIST conversion locale CDATA #IMPLIED >
      <!ATTLIST conversion cursor (local | global) #IMPLIED>
    <!ATTLIST conversion defaultserver CDATA #IMPLIED>

<!ELEMENT rootdir (fgldir?,egldir,artifactsdir?)>
<!ELEMENT fgldir (#PCDATA)>
<!ELEMENT egldir (#PCDATA)>
<!ELEMENT artifactsdir (#PCDATA)>

<!ELEMENT manifestfiles (file)+>
        <!ATTLIST manifestfiles type ( schema | library) #REQUIRED>
<!ELEMENT fglfiles (file)+>
<!ELEMENT formfiles (file)+>
<!ELEMENT fontconfigfile (file)>
<!ELEMENT file (#PCDATA)>

<!ELEMENT msgfiles (file)+>
        <!ATTLIST msgfiles locale CDATA #IMPLIED >
```

Manifest File

Database Schema Extraction Project

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT manifest (package*)>
    <!ATTLIST manifest project CDATA #REQUIRED >
    <!ATTLIST manifest type CDATA #FIXED "schema">
    <!ATTLIST manifest version CDATA #REQUIRED >

<!ELEMENT package (table+)>
    <!ATTLIST package
            name CDATA #REQUIRED
            server CDATA #REQUIRED
            database CDATA #REQUIRED
            mode (ANSI) #IMPLIED>

<!ELEMENT table (column+)>
    <!ATTLIST table
              name CDATA #REQUIRED
            egltype CDATA #REQUIRED
              owner CDATA #IMPLIED>

<!ELEMENT column EMPTY>
    <!ATTLIST column
            name CDATA #REQUIRED
            dataitem CDATA #REQUIRED
            fgltype CDATA #REQUIRED
              egltype CDATA #REQUIRED
            size CDATA #IMPLIED
            start CDATA #IMPLIED
            end CDATA #IMPLIED
            precision CDATA #IMPLIED
            scale CDATA #IMPLIED>
```

Library or Application Project

The DTD for a Library and an Application project are identical.

```
 <?xml version="1.0" encoding="UTF-8"?>

<!ELEMENT manifest (package,forms*)>
   <!ATTLIST manifest project CDATA #REQUIRED>
   <!ATTLIST manifest type CDATA #FIXED "library">

<!ELEMENT package (rectype*, variables* ,function*,cfunc*,
cursor*,preparedStatements*)>
  <!ATTLIST package name CDATA #REQUIRED>

<!ELEMENT function (parameter*,return*)>
   <!ATTLIST function name CDATA #REQUIRED>
   <!ATTLIST function library CDATA #REQUIRED>
   <!ATTLIST function type CDATA #REQUIRED>

<!ELEMENT parameter EMPTY>
   <!ATTLIST parameter name CDATA #REQUIRED>
   <!ATTLIST parameter egltype CDATA #REQUIRED>
   <!ATTLIST parameter fgltype CDATA #IMPLIED>
   <!ATTLIST parameter size CDATA #IMPLIED>
   <!ATTLIST parameter precision CDATA #IMPLIED>
   <!ATTLIST parameter scale CDATA #IMPLIED>
   <!ATTLIST parameter start CDATA #IMPLIED>
   <!ATTLIST parameter end CDATA #IMPLIED>
   <!ATTLIST parameter isrectype (t|f) "f">
   <!ATTLIST parameter library CDATA #REQUIRED>
```

```
<!ELEMENT return EMPTY>
   <!ATTLIST return name CDATA #REQUIRED>
   <!ATTLIST return egltype CDATA #REQUIRED>
   <!ATTLIST return fgltype CDATA #IMPLIED>
   <!ATTLIST return size CDATA #IMPLIED>
   <!ATTLIST return precision CDATA #IMPLIED>
   <!ATTLIST return scale CDATA #IMPLIED>
   <!ATTLIST return start CDATA #IMPLIED>
   <!ATTLIST return end CDATA #IMPLIED>
   <!ATTLIST return isrectype (t|f) "f" >
   <!ATTLIST return library CDATA #REQUIRED>


<!ELEMENT cfunc EMPTY>
   <!ATTLIST cfunc name CDATA #REQUIRED>
   <!ATTLIST cfunc library CDATA #REQUIRED>
   <!ATTLIST cfunc argcount CDATA #REQUIRED>
   <!ATTLIST cfunc retcount CDATA #REQUIRED>


<!ELEMENT cursor EMPTY>
   <!ATTLIST cursor name CDATA #REQUIRED>
   <!ATTLIST cursor ishold (t|f) "f" >
   <!ATTLIST cursor isscrolling (t|f) "f" >
   <!ATTLIST cursor library CDATA #REQUIRED>


<!ELEMENT variables (variable*)>

<!ELEMENT variable EMPTY>
   <!ATTLIST variable name CDATA #REQUIRED>
   <!ATTLIST variable egltype CDATA #REQUIRED>
   <!ATTLIST variable fgltype CDATA #IMPLIED>
   <!ATTLIST variable size CDATA #IMPLIED>
   <!ATTLIST variable precision CDATA #IMPLIED>
   <!ATTLIST variable scale CDATA #IMPLIED>
   <!ATTLIST variable start CDATA #IMPLIED>
   <!ATTLIST variable end CDATA #IMPLIED>
   <!ATTLIST variable isrectype (t|f) "f" >
   <!ATTLIST variable library CDATA #REQUIRED>


<!ELEMENT rectype (field*)>
   <!ATTLIST rectype name CDATA #REQUIRED>
   <!ATTLIST rectype library CDATA #REQUIRED>

<!ELEMENT field EMPTY>
   <!ATTLIST field name CDATA #REQUIRED>
   <!ATTLIST field egltype CDATA #REQUIRED>
   <!ATTLIST field fgltype CDATA #IMPLIED>
   <!ATTLIST field size CDATA #IMPLIED>
   <!ATTLIST field precision CDATA #IMPLIED>
   <!ATTLIST field scale CDATA #IMPLIED>
   <!ATTLIST field start CDATA #IMPLIED>
   <!ATTLIST field end CDATA #IMPLIED>
   <!ATTLIST field isrectype (t|f) "f" >
   <!ATTLIST field library CDATA #REQUIRED>

<!ELEMENT preparedStatements (statement*)>
<!ELEMENT statement EMPTY>
   <!ATTLIST statement name CDATA #REQUIRED>
   <!ATTLIST statement library CDATA #REQUIRED>

<!ELEMENT forms (form*)>
```

```
<!ELEMENT form EMPTY>
    <!ATTLIST form name CDATA #REQUIRED>
    <!ATTLIST form package CDATA #REQUIRED>
```

Appendix G. Conversion Log Examples

In This Appendix

This appendix provides examples of conversion logs in **.txt** format.

Application Project: PASSED Example

```
--------------------------------------------------------------------------
                        i4gldemo Conversion Log
--------------------------------------------------------------------------

Conversion Status:
------------------
Project Status                   : PASSED
Conversion date                  : Wed Feb 02 22:22:53 CST 2005
User                             : jdoe
Host                             : boom
OS version                       : Windows XP

Project Details:
----------------
Project Name                    : i4gldemo
Conversion Type                 : application
4GL root directory              : C:\i4gl\i4gldemo
EGL destination directory       : C:\workspace\i4gldemo
Conversion artifacts directory  : C:\temp\i4gldemo\ConversionArtifacts
Configuration file              : C:\i4gl\config\i4gldemoConfig.xml
Default Informix server instance: myserver
Informix cursor scope           : local

Conversion Artifacts:
-------------------------
Manifest File Generated  : C:\temp\i4gldemo\ConversionArtifacts\manifest\
i4gldemoApplicationManifest.xml
EGL Build descriptor file: C:\temp\i4gldemo\i4gldemo\EGLSource\i4gldemo.eglbld

I4GL Source File Conversion Summary:
------------------------------------------------
Total number of I4GL files given: 19
Total number of files converted successfully: 19

d4_cust.4gl -> PASSED
d4_demo.4gl -> PASSED
d4_globals.4gl -> PASSED
d4_load.4gl -> PASSED
d4_main.4gl -> PASSED
d4_orders.4gl -> PASSED
d4_report.4gl -> PASSED
d4_stock.4gl -> PASSED
cust.per -> PASSED
custcur.per -> PASSED
custform.per -> PASSED
customer.per -> PASSED
ordcur.per -> PASSED
order.per -> PASSED
orderform.per -> PASSED
p_ordcur.per -> PASSED
state_list.per -> PASSED
stock1.per -> PASSED
stock_sel.per -> PASSED
```

```
Source File Conversion Details:
-------------------------------

4GL source file given          : C:\i4gl\i4gldemo\fgl\d4_cust.4gl
EGL source file generated      : C:\workspace\i4gldemo\d4_cust.egl
Status                         : PASSED

4GL source file given          : C:\i4gl\i4gldemo\fgl\d4_demo.4gl
EGL source file generated      : C:\workspace\i4gldemo\d4_demo.egl
Status                         : PASSED

4GL source file given          : C:\i4gl\i4gldemo\fgl\d4_globals.4gl
EGL source file generated      : C:\workspace\i4gldemo\d4_globals.egl
Status                         : PASSED

4GL source file given          : C:\i4gl\i4gldemo\fgl\d4_load.4gl
EGL source file generated      : C:\workspace\i4gldemo\d4_load.egl
                               : C:\workspace\i4gldemo\d4_load_program.egl
Status                         : PASSED

4GL source file given        : C:\i4gl\i4gldemo\fgl\d4_main.4gl
EGL source file generated    : C:\workspace\i4gldemo\d4_main.egl
                             : C:\workspace\i4gldemo\d4_main_program.egl
Status                       : PASSED

4GL source file given        : C:\i4gl\i4gldemo\fgl\d4_orders.4gl
EGL source file generated    : C:\workspace\i4gldemo\d4_orders.egl
                             : C:\workspace\i4gldemo\r_invoice_handler.egl
                             : C:\workspace\i4gldemo\r_invoice_XML.jrxml
Status                       : PASSED

4GL source file given      : C:\i4gl\i4gldemo\fgl\d4_report.4gl
EGL source file generated  : C:\workspace\i4gldemo\d4_report.egl
                           : C:\workspace\i4gldemo\labels_report_handler.egl
                           : C:\workspace\i4gldemo\labels_report_XML.jrxml
                           : C:\workspace\i4gldemo\ar_report_handler.egl
                           : C:\workspace\i4gldemo\ar_report_XML.jrxml
Statu                      : PASSED

4GL source file given          : C:\i4gl\i4gldemo\fgl\d4_stock.4gl
EGL source file generated      : C:\workspace\i4gldemo\d4_stock.egl
Status                         : PASSED

4GL source file given          : C:\i4gl\i4gldemo\fgl\forms\cust.per
EGL source file generated      : C:\workspace\i4gldemo\forms\custForm.egl
Status                         : PASSED

4GL source file given          : C:\i4gl\i4gldemo\fgl\forms\custcur.per
EGL source file generated      : C:\workspace\i4gldemo\forms\custcurForm.egl
Status                         : PASSED

4GL source file given          : C:\i4gl\i4gldemo\fgl\forms\custform.per
EGL source file generated      : C:\workspace\i4gldemo\forms\custformForm.egl
Status                         : PASSED

4GL source file given          : C:\i4gl\i4gldemo\fgl\forms\customer.per
EGL source file generated      : C:\workspace\i4gldemo\forms\customerForm.egl
Status                         : PASSED

4GL source file given          : C:\i4gl\i4gldemo\fgl\forms\ordcur.per
EGL source file generated      : C:\workspace\i4gldemo\forms\ordcurForm.egl
Status                         : PASSED

4GL source file given          : C:\i4gl\i4gldemo\fgl\forms\order.per
EGL source file generated      : C:\workspace\i4gldemo\forms\orderForm.egl
Status                         : PASSED
```

```
4GL source file given          : C:\i4gl\i4gldemo\fgl\forms\orderform.per
EGL source file generated      : C:\workspace\i4gldemo\forms\orderformForm.egl
Status                         : PASSED

4GL source file given          : C:\i4gl\i4gldemo\fgl\forms\p_ordcur.per
EGL source file generated      : C:\workspace\i4gldemo\forms\p_ordcurForm.egl
Status                         : PASSED

4GL source file given          : C:\i4gl\i4gldemo\fgl\forms\state_list.per
EGL source file generated      : C:\workspace\i4gldemo\forms\state_listForm.egl
Status                         : PASSED

4GL source file given          : C:\i4gl\i4gldemo\fgl\forms\stock1.per
EGL source file generated      : C:\workspace\i4gldemo\forms\stock1Form.egl
Status                         : PASSED

4GL source file given           : C:\i4gl\i4gldemo\fgl\forms\stock_sel.per
EGL source file generated       : C:\workspace\i4gldemo\forms\stock_selForm.egl
Status                          : PASSED
```

Application Project: FAILED Example

```
--------------------------------------------------------------------------------
                        i4gldemo Conversion Log
--------------------------------------------------------------------------------

Conversion Status:
-------------------
Project Status                : FAILED
Conversion date               : Wed Feb 02 22:22:53 CST 2005
User                          : jdoe
Host                          : boom
OS version                    : Windows XP

Project Details:
-----------------
Project Name                      : i4gldemo
Conversion Type                   : application
4GL root directory                : C:\i4gl\i4gldemo
EGL destination directory         : C:\workspace\i4gldemo
Conversion artifacts directory    : C:\temp\i4gldemo\ConversionArtifacts
Configuration file                : C:\i4gl\config\i4gldemoConfig.xml
Default Informix server instance  : myserver
Informix cursor scope             : local


I4GL Source File Conversion Summary:
------------------------------------------------
Total number of I4GL files given: 1
Total number of files converted successfully: 0

d4_cust.4gl -> FAILED

Source File Conversion Details:
-------------------------------

4GL source file given          : C:\i4gl\i4gldemo\fgl\d4_cust.4gl
EGL source file generated      : C:\workspace\i4gldemo\d4_cust.egl
Status                         : FAILED
```

Database Schema Extraction Project: FAILED Example

```
--------------------------------------------------------------------------------
                        Stores7 Conversion Log
--------------------------------------------------------------------------------
```

```
Conversion Status :
-------------------
Project Status                    : FAILED
Conversion date                   : Thu Feb 03 11:25:33 CST 2005
User                              : jdoe
Host                              : boom
OS version                        : Windows XP

Project Details :
-----------------
Project Name                      : Stores7
Conversion Type                   : schema
EGL destination directory         : C:\workspace\stores7
Conversion artifacts directory    : C:\workspace\stores7\
 ConversionArtifacts
Configuration file                : C:\workspace\stores7\
 conversionArtifacts\config\Stores7SchemaConfig.xml

Database Connection details:
----------------------------

   Database                       : stores7
   Server                         : myserver
   Host                           : boom.antartica.world.com
   Port                           : 2005
   User                            : jdoe
   CLIENT_LOCALE                   : en_us.8859-1
   DB_LOCALE                       : en_us.8859-1


Exceptions:
-----------

ERROR :   Server : "myserver" Database : "stores7"


Informix JDBC Exception :
java.sql.SQLException: User (com.informix.asf.IfxASFRemoteException jdoe)'s
password is not correct for the database server.
```

Appendix H. EGL Build Descriptor Example

In This Appendix

This appendix provides examples of EGL build descriptor files generated by the Conversion Utility.

EGL Build Descriptor Overview

The Conversion Utility generates a build descriptor file for each converted project. The generated file provides values for only the properties that are relevant to converted I4GL files.

Database Schema Extraction Project

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE EGL PUBLIC "-//IBM Corporation, Inc.//DTD EGL Build Parts
 6.0//EN" "">
<EGL>
<BuildDescriptor
    name="Stores7JavaBuildOptions"
    genProject="Stores7"
    genDirectory="C:\temp\Stores7\EGLSource\Stores7"
    system="WIN"
    J2EE="NO"
    sqlCommitControl="AUTOCOMMIT"
    itemsNullable="YES"
    genProperties="GLOBAL"
    genDataTables="YES"
    dbms="INFORMIX"
    sqlValidationConnectionURL="jdbc:informix-sqli://
mymachine.loc.comp.com:2005/stores7:
 INFORMIXSERVER=myserver;"
    sqlJDBCDriverClass="com.informix.jdbc.IfxDriver"  sqlID="jdoe"
sqlPassword="password" sqlDB="jdbc:informix-sqli://
mymachine.loc.comp.com:2005/stores7:
 INFORMIXSERVER=myserver;" >
</BuildDescriptor>
</EGL>
```

Library or Application Project

The Conversion Utility does not have access to the **user name** or **password** values for Library or Application projects. For these two project types, the Conversion Utility generates a placeholder connection URL in the build descriptor file. This connection URL must be edited before using the build file. In addition, you must also edit the **sqlID** and **sqlPassword** values.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE EGL PUBLIC "-//IBM Corporation, Inc.//DTD EGL Build Parts
 6.0//EN" "">
<EGL>
<BuildDescriptor
    name="i4gldemoJavaBuildOptions"
    genProject="i4gldemo"
    genDirectory="C:\temp\i4gldemo\i4gldemo\EGLSource\i4gldemo"
    system="WIN"
    J2EE="NO"
    sqlCommitControl="AUTOCOMMIT"
    itemsNullable="YES"
```

```
    genProperties="GLOBAL"
    genDataTables="YES"
    dbms="INFORMIX"
    sqlValidationConnectionURL="jdbc:informix-sqli://host:port/
database:INFORMIXSERVER=server;"
    sqlJDBCDriverClass="com.informix.jdbc.IfxDriver"  sqlID="user"
sqlPassword="password" sqlDB="jdbc:informix-sqli://host:port/
database:INFORMIXSERVER=server;" >
</BuildDescriptor>
</EGL>
```

Appendix I. EGL Reserved Words

In This Appendix

This appendix lists the EGL reserved words forVersion 6.0.0.1. For the most recent list of EGL reserved words see the *EGL Reserved Words* online help topics.

EGL Reserved Words

The following words are reserved in EGL:

- absolute, add, all, any, as
- bigInt, bin, bind, blob, boolean, by, byName, byPosition
- call, case, char, clob, close, const, continue, converse, current
- dataItem, dataTable, date, dbChar, decimal, decrement, delete, display, dliCall
- else, embed, end, escape, execute, exit, externallyDefined
- false, field, first, float, for, forEach, form, formGroup, forUpdate, forward, freeSql, from, function
- get, goto
- handler, hex, hold
- if, import, in, inOut, insert, int, interval, into, is, isa
- label, languageBundle, last, library, like
- matches, mathlib, mbChar, money, move
- new, next, nil, no, noRefresh, not, nullable, num, number, numc
- onEvent, onException, open, openUI, otherwise, out
- pacf, package, pageHandler, passing, prepare, previous, print, private, program, psb
- record, ref, relative, replace, report, return, returning, returns
- scroll, self, set, show, singleRow, smallFloat, smallInt, sql, sqlCondition, stack, string, strlib, syslib, sysvar
- this, time, timeStamp, to, transaction, transfer, true, try, type
- unicode, update, url, use, using, usingKeys
- when, while, with, withinParent
- yes

Glossary

C

Command line conversion. A command line equivalent to the Conversion Utility Wizard. Running this program requires you to manually create the configuration file and then run the **e4GL** script. The command line conversion is recommended primarily for reconversion efforts.

Configuration file. Located in the *EGLDestinationDirectory*/**ConversionArtifacts/ config** directory, this file provides details about your configuration.

Console User Interface (CUI). The EGL equivalent to 4GL Forms.

Conversion artifacts. The configuration, manifest, and conversion log files generated during the conversion.

Conversion log. Located in the *EGLDestinationDirectory*/**ConversionArtifacts /log** directory and named *ProjectName*.**log**, the conversion log identifies the conversion errors, warnings and file disposition. This file is used to identify how to correct conversion errors.

Conversion Utility Wizard. This wizard collects your database schema, shared library and I4GL application information and launches the conversion process.

ConversionArtifacts directory. Located in *EGLDestinationDirectory*, this directory contains the sub-directories for the configuration, manifest, and log files.

E

E4GL. This is the script program for the command line conversion of I4GL files to EGL. **E4GL** activates the **I4GL2EGL** Conversion Utility.

EGL. A high-level language that allows developers to focus on business logic as they create complex business applications for deployment in any of several environments, including the Web. The language simplifies database and message-queue access, as well as the use of J2EE.

EGL package. An EGL package is a named collection of related source parts, and is comparable to a I4GL Project. During I4GL to EGL conversion, I4GL Project components are converted into EGL Package parts.

EGL project. An EGL project includes zero to many source folders, each of which includes zero to many packages, each of which includes zero to many files. Each file contains zero to many parts.

I

I4GL2EGL. The program that converts I4GL source files into EGL source files.

Information Center. The online help that provides detailed information about your Rational product, and EGL. Access the information center by selecting **Help > Rational Help**.

J

JasperReports. An open source reporting library written in Java and used by EGL to produce reports. Your I4GL reports convert to both **.egl** and JasperReport **.jrxml** files.

M

Manifest file. Located in the *EGLDestinationDirectory*/**ConversionArtifacts /manifest** directory, a manifest file is generated for each conversion stage. The database schema manifest file contains information about all of the tables, columns and data types of the selected database. The shared library manifest file lists the I4GL and assumed C function calls used in the I4GL converting project. The application manifest file provides a list of the technical details of the project.

R

readme.html. Located in the top directory of your Rational product, the **readme.html** file contains information on product limitations.

readme004FGL.html. Located in the Conversion Utility plugin directory, the **readme004FGL.html** file contains information on Conversion Utility limitations and procedures, and changes to documentation since the completion of this User's Guide.

Reconversion. Occasionally shared libraries do not convert successfully and must be reconverted, either through the Conversion Utility Wizard or by command line.

W

Wizard. The Conversion Utility Wizard guides you through the steps necessary to convert your I4GL

application. Access the Wizard by selecting **File > New > Other > Informix 4GL to EGL Conversion**.

Workspace. In Rational products, the workspace is the central hub for data files. A workspace's resources are organized in a tree structure, with projects at the top, and folders and files underneath.

Error Messages

This error messages section provides explanatory notes and user responses for Conversion Utility and FGL Parser error messages.

Occasionally, the Conversion Utility terminates because of problems with your JRE or Rational product. In such a case, the conversion terminates without producing any error message or log file. Once you have resolved the problem with your JRE or Rational product, you must start the conversion process from the beginning.

Conversion Utility Error Messages

The name of the configuration file is not provided.

Explanation: The name of the configuration file was not entered or was incorrectly entered in the wizard or the command line.

User Response: Enter the correct name of the configuration file.

The configuration file cannot be read or has an invalid format.

Explanation: The configuration file cannot be read or has an invalid format.

User Response: Correct the format of the configuration file.

The manifest file cannot be read or has an invalid format.

Explanation: The manifest file cannot be read or has an invalid format.

User Response: Correct the format of the manifest file.

The conversion artifacts directory cannot be created.

Explanation: The conversion artifacts directory cannot be created.

User Response: Check the write permission in the file system to create the artifacts directory.

The I4GL root directory does not exist or cannot be read.

Explanation: The I4GL root directory does not exist or cannot be read.

User Response: Provide the correct I4GL root directory.

The I4GL source file does not exist or cannot be read.

Explanation: The I4GL source file does not exist or cannot be read.

User Response: Check the path and the read permission of the I4GL source file.

The I4GL form file does not exist or cannot be read.

Explanation: The I4GL form file does not exist or cannot be read.

User Response: Check the path and the read permission of the I4GL form file.

The I4GL message file does not exist or cannot be read.

Explanation: The I4GL message file does not exist or cannot be read.

User Response: Check the path and the read permission of the I4GL message file.

The EGL destination directory cannot be created.

Explanation: The EGL destination directory cannot be created.

User Response: Verify that you have enough disc space and have write permission.

No database connection information is found in the configuration file.

Explanation: The configuration file does not contain information on how to connect to your Informix database.

User Response: Verify that the configuration file contains database connection information. The Database section of the configuration file must have valid XML elements that conform to the DTD.

The EGL source file cannot be created.

Explanation: The EGL source file cannot be created.

User Response: Check the write permission and disk space of the EGL destination directory.

The manifest file cannot be created.

Explanation: The manifest file cannot be created.

User Response: Check the path and write permission of the **ConversionArtifacts/manifest** directory.

A syntax error was found in the I4GL form file.

Explanation: A syntax error was found in the I4GL form file.

User Response: Verify that the error file created for the line and column number information where the syntax error was found. This file must be a valid I4GL form file. Compile the form file with the I4GL **form4gl** form compiler to identify error message details.

The conversion log file cannot be created.

Explanation: The conversion log file cannot be created.

User Response: Check the path and write permission of the **ConversionArtifacts/log** subdirectory.

The directory does not exist or has no write permissions.

Explanation: The directory does not exist or has no write permissions.

User Response: Check the path and the write permissions of the directory.

FGL Parser Messages

Found IDENTIFIER in array dimension. May need to be defined.

Explanation: I4GL does not allow variables in array dimension declarations. Therefore, the I4GL file must use a preprocessor directive to resolve this problem, which occurs when the I4GL compiler converts it to C.

User Response: The IDENTIFER must be created as a final INT variable in EGL and a value must be assigned to it.

Return type unknown.

Explanation: The function returns a value, but the value could not be determined.

User Response: Correct the declared return type for the function in EGL.

Cursor declaration not located.

Explanation: The cursor name was opened, but the declaration could not be found to associate the cursor with the prepared statement and the HOLD/SCROLL attributes.

User Response: Determine the HOLD/SCROLL characteristics of the cursor from the I4GL program, and if either is specified, add the attributes on the EGL open statement.

Dynamic specification of Form is not supported.

Explanation: A Form cannot be loaded dynamically. The Local/Global instance of the Form must be declared. The statements referencing the Form through the variable work if there is an existing instance of a Form with a name equal to the value of the variable.

User Response: Declare the Local/Global instance.

Error Unsupported statement.

Explanation: The statement at the specified line is not supported during I4GL to EGL conversion, or by the EGL language.

User Response: Conversion is not possible. Rewrite the functionality in EGL if still required.

C-compiler directive encountered.

Explanation: I4GL allows you to embed C compiler directives in the I4GL file, which is included in the intermediate ESQL/C files evaluated by the C compiler. These C compiler directives are not supported during I4GL to EGL conversion and are not typically supported by EGL.

User Response: The conversion ignored the C-compiler directive. Verify and correct the generated code relating to the directive.

Assumed C-compiler directive encountered but not at start of line.

Explanation: I4GL allows you to embed C compiler directives in the I4GL file, which is included in the intermediate files sent to the C compiler. These C compiler directives are not supported during I4GL to EGL conversion, and are not typically supported by EGL. The directives must usually start in column 1 but in this instance they do not.

User Response: The conversion ignored the C-compiler directive. Verify and correct the generated code relating to the directive.

Undefined variable ″{0}″.

Explanation: A definition for the named variable could not be found, thus preventing the proper

generation of the code that references the variable when the type information is required.

User Response: Check the variable name generated to determine if it was mapped from a reserved word or an I4GL variable, or if it was a global variable. Correct the Conversion Project definition to include the required global libraries or missing files.

"WITH REOPTIMIZATION" is not supported.

Explanation: The 4GL/ESQL/C allowed specifying 'WITH REOPTIMIZATION' when opening a cursor. JDBC does not support this, so the option is ignored. SQL statements with host variable parameters are typically reoptimized by the server with each open.

User Response: Consider the possible performance impact with your target server. If the impact is significant, rewriting the code to re-prepare the statement will reoptimize the execution.

"With Concurrent Transactions" is the only supported mode.

Explanation: JDBC does not limit the ability to have concurrent transactions. No errors are generated at run time if the application with a pending transaction starts a new transaction on a different connection.

User Response: If this situation must be prevented, write additional checks into the program to prevent switching connections with transactions pending. The **disconnect()** function will return an exception if a transaction is pending.

NOT IMPLEMENTED.

Explanation: The options at the specified location in the statement are not supported during I4GL to EGL conversion or by the EGL language.

User Response: Rewrite the code using EGL capabilities.

Duplicate Function: "{0}" defined locally and in external project "{1}".

Explanation: The function name specified was already defined in an imported project. A possible cause for this error is that I4GL files were incorrectly combined into the same conversion project or that a shared library manifest file was included unnecessarily.

User Response: Verify that I4GL files were not incorrectly combined into the same conversion project and that a shared library manifest file was not included unnecessarily. You should also rename the function or remove the **use** statement specifying the library containing the duplicate function.

Function "{0}" referenced in another project before defined. Reconversion required for project "{1}".

Explanation: A previously undefined function referenced in an imported project was defined here.

User Response: Reconvert he imported project to ensure correct code generation.

Undefined type "{0}" for variable "{1}"

Explanation: The specified record type was not found in any SCHEMA manifests and is needed by the specified variable. This can happen when the application is using an ANSI database, and the source is written assuming that the current user is the owner of the ANSI tables.

User Response: Either add the owner names to the table references or remove the owner names from the SCHEMA manifest and reattempt the conversion.

ERROR: Invalid manifest type.

Explanation: The manifest type does not match or identify a known DTD definition for the manifest.

User Response: Regenerate the associate library to recreate the manifest file, or attempt to correct the DTD reference.

ERROR: Manifest file reconversion failed.

Explanation: An internal exception occurred updating the dependent project files.

User Response: Correct the external cause and rerun the conversion. If the external cause cannot be corrected, contact IBM support.

ERROR: Project {0} needs to be reconverted, Manifest {1} has been updated.

Explanation: The specified project references functions defined in the current project.

User Response: Reconvert the specified project to ensure proper code generation.

ERROR: Manifest file generation failed.

Explanation: An exception occurred while trying to save the project manifest.

User Response: Correct the external cause and rerun the conversion. If the external cause cannot be corrected, contact IBM support.

ERROR: Cannot Generate Native Library

Explanation: An exception occurred while trying to write the C native library code.

User Response: Correct the external cause and rerun the conversion. If the external cause cannot be corrected, contact IBM support.

Notices

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

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created

programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
J46A/G4
555 Bailey Avenue
San Jose, CA 95141-1003
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

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.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

> © (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. (enter the year or years). All rights reserved.

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

Trademarks

AIX; DB2; DB2 Universal Database; Distributed Relational Database Architecture; NUMA-Q; OS/2, OS/390, and OS/400; IBM Informix®; C-ISAM®; Foundation.2000™; IBM Informix ® 4GL; IBM Informix®DataBlade®Module; Client SDK™; Cloudscape™; Cloudsync™; IBM Informix®Connect; IBM Informix®Driver for JDBC; Dynamic Connect™; IBM Informix®Dynamic Scalable Architecture™(DSA); IBM Informix®Dynamic Server™; IBM Informix®Enterprise Gateway Manager (Enterprise Gateway Manager); IBM Informix®Extended Parallel Server™; i.Financial Services™; J/Foundation™; MaxConnect™; Object Translator™; Red Brick™; IBM Informix® SE; IBM Informix® SQL; InformiXML™; RedBack®; SystemBuilder™; U2™; UniData®; UniVerse®; wintegrate®are trademarks or registered trademarks of International Business Machines Corporation.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

Windows, Windows NT, and Excel are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Other company, product, and service names used in this publication may be trademarks or service marks of others.

Index

Special characters

Numerics

A

B

C

D

E

F

G

I

J

K

L

M

N

O

P

R

S

T

U

W

Printed in USA

G251-2485-00