Rational Business Developer
Version 8 Release 0

EGL Server Guide for IBM i

SC31-6841-05

Rational Business Developer
Version 8 Release 0

EGL Server Guide for IBM i

SC31-6841-05

Note
Before using this document, read the general information under Chapter 7, "Notices," on page 37.

Fifth edition (January 2011)

This edition applies to version 8.0 of IBM Rational Business Developer and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Chapter 1. Installing and customizing EGL Server for IBM i 1

Installation files for EGL Server for IBM i 1
Objects created or replaced during installation . . . 1
Setting up the IBM i library and files 1
FDAPREP preparation script file for IBM i 2
Customizing EGL Server 3
  Specifying a language 3
  Changing the code page for EGL system libraries 4
Administering EGL Server for IBM i 4
  Data description specifications generated by EGL 4
  Runtime considerations: commitment control cycles 5

Chapter 2. Reviewing and preparing the generated output 7

Outputs of generation 7
  Objects generated for programs 8
  Objects generated for data tables 9
  Objects generated for form groups 9
Preparation to run generated programs 9
Starting the IBM i remote build server 10
Verifying the IBM i remote build server 10
Launching the build plan manually 11
Preparing a DB2 program. 11
Customizing compiler process options using VGNCBLOP 11

Chapter 3. Running a generated program on an IBM i system 13

Setting up the environment to run the generated programs 13
Establishing a library list for a job 14
Running EGL programs and passing arguments . . 15
  Running EGL programs under IBM i 15
  Passing arguments and declaring parameters . . 15
EGL run units 16
  Using tables with run units 16
  Using activation groups with run units 17

Chapter 4. Diagnosing problems during run time 19

Diagnostic commands 19
IBM i First Failure Data Capture 19

Chapter 5. Exception codes 21

Chapter 6. Messages 23

Message format 23
Escape messages. 23
Diagnostic and informational messages 24

Chapter 7. Notices 37

Programming interface information 39
Trademarks and service marks 39

Chapter 1. Installing and customizing EGL Server for IBM i

This chapter contains general information on the installation and customization of EGL Server for IBM® i on the host and EGL on the workstation.

Installation files for EGL Server for IBM i

EGL Server for IBM i is in the Rational Business Developer directory, which is in the product shared resources directory.

The `QEGL.zip` file contains the runtime code and is in the following plug-in folder, where *version* is the current version of the product:

`com.ibm.etools.egl.generators.cobol_`*`version`*`\iSeriesRuntime`

The `ReadMe.txt` file is in the same folder and contains the instructions for installing the EGL Server for IBM i.

For information about specifying the language for your environment, see "Customizing EGL Server" on page 3.

Objects created or replaced during installation

Table 1 provides a general description of the objects that are created or replaced during the installation process.

Table 1. Objects created or replaced during the installation process

Object and library name	Type	Description
QEGL QSYS	*LIB	The primary library for the system objects that constitute EGL Server for IBM i.

Setting up the IBM i library and files

After the generate and preparation steps are complete for an EGL program, the outputs move into files in the library that is specified with the **destLibrary** build descriptor option. The default value for the **destLibrary** build descriptor option is QGPL.

Before the preparation step can run, you must create a set of files in that library. Table 2 contains a list and description of the files that must be created.

Table 2. Generation output files

File name	Type	Description
QVGNCBLS	PF-SRC	EGL generation COBOL source
QVGNCLS	PF-SRC	EGL generation CL source
QVGNDDSS	PF-SRC	EGL generation DDS source file
QVGNEVF	PF-SRC	EVF parts control file
QVGNMAPG	PF-DTA	EGL generation form group source
QVGNTAB	PF-DTA	EGL generation table data
QVGNWORK	PF-SRC	EGL generation work file

Create the generation output files by using the following commands:

```
CRTSRCPF FILE(QGPL/QVGNCBLS) RCDLEN(92 ) TEXT('EGL GENERATION - COBOL SRC')
CRTSRCPF FILE(QGPL/QVGNCLS ) RCDLEN(92 ) TEXT('EGL GENERATION - CL SRC')
CRTSRCPF FILE(QGPL/QVGNDDSS) RCDLEN(92 ) TEXT('EGL GENERATION - DDS SRC')
CRTSRCPF FILE(QGPL/QVGNEVF ) RCDLEN(92 ) TEXT('EGL GENERATION - VARIABLES')
CRTSRCPF FILE(QGPL/QVGNWORK) RCDLEN(150) TEXT('EGL GENERATION - WORK FILE')
CRTPF FILE(QGPL/QVGNMAPG) SRCFILE(QEGL/QVGNPDDS) SRCMBR(TBLMAP) MBR(*NONE)
  TEXT('EGL GENERATION- FORM GROUP FILE')
  MAXMBRS(*NOMAX) AUT(*CHANGE) OPTION(*NOSRC *NOLIST)
CRTPF FILE(QGPL/QVGNTAB) SRCFILE(QEGL/QVGNPDDS) SRCMBR(TBLMAP) MBR(*NONE)
  TEXT('EGL TABLE DATA') MAXMBRS(*NOMAX) AUT(*CHANGE) +
  OPTION(*NOSRC *NOLIST)
```

To avoid name collisions when multiple application developers are using the same host IBM i system, each user must have a separate library by specifying a unique value for the **destLibrary** build descriptor option.

After you create the library, put the files listed in Table 2 on page 1 in the library. You can use one of the following REXX code to create the files:

- The EGLSETUP script in the `QEGL.zip` file.
- The EGLSETUP program in `QEGL/QREXSRC`.

For example,

```
STRREXPRC SRCMBR(EGLSETUP) SRCFILE(QEGL/QREXSRC) PARM(xxxxxx)
```

where *xxxxxx* is the name of the library.

If you are using client server support for EGL to call non-EGL-generated programs, locate QVGNRNCL in the `QVGNSAMP` file of your QEGL library. Run CRTCLPGM on the `QVGNRNCL` file and place the program in all libraries that contain a non-EGL program that will be called from an EGL-generated client. If you do not complete these steps, unresolved references to the `QVGNRNCL` file will exist.

FDAPREP preparation script file for IBM i

To begin preparation, the build server invokes a REXX script called FDAPREP on IBM i. The script is in the `QEGL/QREXSRC` file, but can be copied to another location and customized.

The build server invokes the script by using the `STRREXPRC` command:

```
STRREXPRC SRCMBR'(FDAPREP)' SRCFILE'(*LIBL/QREXSRC)' ... (other parms)
```

The file must be in the first occurrence in the library that has QREXSRC.

The preparation script is written in standard REXX, but you can customize it. At generation time, the user defines the **SYMPARM** symbolic parameters in the build descriptor. The parameters are available to the program as standard REXX variables; you can use the parameters to influence the logic as needed. Some standard variables are always defined, as in the following example:

```
EZEFUNCTION="PMN"
EZEDESTLIBRARY="DBANERJE"
EZEDESTUSERID="DBANERJE"
SYSTEM="ISERIESC"
EZESQL="N"
EZEENV="ISERIESC"
EZETRAN="MAIN400"
```

```
DEBUG="YES"
SYSTEM="ISERIESC"
HEAPSIZE="1000"
EZEMBR="MAIN400"
EZEGMBR="MAIN400"
EZEGDATE="08/16/06"
ZENLS="ENU"
ZEGTIME="15:39:13"
```

The following SYMPARMS are recognized by the REXX script.

retainTemporaryFiles=YES
Retains temporary files, such as the COBOL source file that is used in the preparation step, in the library that the **destLibrary** build descriptor option defines after the compile is complete. The default value is NO, which causes the files to be deleted.

DEBUG=YES
Invokes the COBOL compiler with the debug option. If you specify YES, the temporary files from the preparation step are retained, regardless of the value of the **retainTemporaryFiles** option.

Listing=NO
Causes the COBOL compiler to be invoked and a compiler listing is not generated. Invokes the COBOL compiler. A compiler listing does not generate. This setting can be used to accelerate the preparation process, because the time to download the listing file can be significant. The default value is YES, which causes the compiler listing to generate.

The generation step creates the *aliasname*.evf file, where *aliasname* is the alias of the part that is generated. The file contains the variables and is passed to the preparation script.

Customizing EGL Server

After you install EGL Server for IBM i, you can customize it by specifying a language and changing the code pages.

Specifying a language

Table 3 contains the list of supported languages and the associated language module for each language. The language modules are included in the QEGL Library. The EGL Server uses the messages in the QEGL/QVGNMSGF file. To specify the language to be used, copy the file for the desired language to QVGNMSGF.

Table 3. Supported languages

Suffix	Language	Module
CHS	Simplified Chinese	QVGNMCHS
CHT	Traditional Chinese	QVGNMCHT
DEU	German	QVGNMDEU
ENU	English	QVGNMENU
ESP	Spanish	QVGNMESP
FRA	French	QVGNMFRA
ITA	Italian	QVGNMITA
JPN	Japanese (Katakana)	QVGNMJPN
KOR	Korean	QVGNMKOR

Table 3. Supported languages (continued)

Suffix	Language	Module
PTB	Brazilian Portuguese	QVGNMPTB

Changing the code page for EGL system libraries

The EGL run time comes with 11 precompiled system library programs. These programs are written in COBOL. When the programs are distributed, they are precompiled; therefore, by default they use the English code page for any character to/from Unicode transformations. You do not have to use the default; you can alter the settings to use the code page that you need.

Each time a system library performs a transformation between character and Unicode, the system library calls a runtime program called VGNUCDE. This VGNUCDE program is written in COBOL and does transformations by using the COBOL MOVE statements to and from character and national character values, which causes the appropriate transformations. The source of VGNUCDE is provided in your QEGL\QVGNSAMP data set.

To alter this program so that it uses your required code page:

1. Recompile the source code by specifying your code page in the COBOL parameters or by default in your IBM i configuration.
2. Specify whether the resulting program is to replace the VGNUCDE program in the QEGL data set, or is to be placed in any data set that precedes QEGL in the LIBL concatenation order.

To compile the VGNUCDE program, use the following commands:

```
CRTCBLMOD MODULE(QEGL/VGNUCDE) SRCFILE(QEGL/QVGNSAMP) SRCMBR(VGNUCDE)
CRTPGM PGM(QEGL/VGNUCDE) MODULE(QEGL/VGNUCDE)
```

Administering EGL Server for IBM i

This section describes the general considerations for administering EGL Server for IBM i.

- "Data description specifications generated by EGL"
- "Runtime considerations: commitment control cycles" on page 5

Data description specifications generated by EGL

If the **genDDSFile** build descriptor option is set to YES, the data description specifications (DDS) generate. The information is generated from the EGL record definitions that are used for file I/O operations.

The generated DDS information is useful to IBM i system administrators or application developers. System administrators can use the DDS source members, or modified versions of them, to create the files that do not already exist on the IBM i system. Using the DDS source information to create the files qualifies the files for IBM i data management functions, such as specifying key fields, unique keys, and logical files.

You are not required to use the DDS source information to create files because EGL does not require programs to access externally described files. EGL relies on the record definition, which is built into the *PGM object, for the structure of a record.

However, using the DDS information guarantees an agreement between the program view of the record structure and the record data stored on the IBM i system.

DDS keyword modification

You might need to modify the DDS source members that correspond to records that are defined for indexed and alternate index files. The minimum modification is to add DDS keywords to the file and record level identifiers in the DDS source member.Table 4 shows the DDS keywords and the conditions under which they are required.

To optimize record retrieval and simplify program logic, you can add other DDS keywords. For example, you can use logical files to select a subset of physical file records. You can also build your own DDS source member, based on your knowledge of the EGL record definitions in the program. If you build your own DDS source member, individual field names and field lengths in the DDS source do not need to differ from those of the EGL record definition. However, the record length and key field length of the EGL record definition and the DDS source must be equal.

Table 4. Conditions for using DDS keywords

DDS keyword	Condition
PFILE(*pfname*)	When using the DDS information to create a logical file. *pfname* identifies the physical file on which the logical file is based. PFILE is a record-level keyword.
UNIQUE	When the program tests for the unique or duplicate record I/O error conditions. UNIQUE is a file-level keyword.

Restrictions on logical files

EGL supports simple logical files that use only one record format. The DDS source information specifies only one file on the PFILE keyword.

Changing DDS member types

EGL creates DDS source members without specifying a member type. To modify the DDS source information, change the member type to one of the following:

- PF for a DDS source member describing a physical file
- LF for a DDS source member describing a logical file

When you change the member type to PF or LF, the Source Entry Utility (SEU) prompting is enabled so that you can modify the DDS source member.

Runtime considerations: commitment control cycles

To use IBM i Commitment Control Services for single-system IBM i programs, you must explicitly start and end a commitment control cycle by using the start commitment control (`STRCMTCTL`) command and the end commitment control (`ENDCMTCTL`) command.

EGL Server for IBM i does not implicitly start or end commitment control cycles for single-system IBM i programs. However, DB2® implicitly starts commitment control automatically for programs that use SQL I/O statements. After commitment control starts for the job, both native database I/O and SQL I/O can use the common commitment control that IBM i provides.

For EGL client and server programs and web programs, the runtime control language (CL) for the program starts the commitment control.

You can change the commitment control for an SQL program by modifying the FDAPREP REXX program. You can further control the commitment control by specifying a user-defined symbolic parameter during generation.

If no commitment control cycle is active and the program attempts to open a file that requires commitment control, the program ends with an error condition. A probable reason for the error is likely that the program attempted to explicitly commit changes to a file. Programs can explicitly commit changes only within an active commitment control cycle. For details about the error, see the messages in the job log.

Chapter 2. Reviewing and preparing the generated output

Before EGL runs your programs, it prepares generation output files. The format and content of the output files are in COBOL, control language (CL) source, and structured binary streams.

Outputs of generation

After you generate a program, several objects must be transferred to the IBM i host system as members in various IBM i physical files. On the IBM i host system, these members must be prepared before the program can be run.

Table 5 provides information about the types of files produced by generation, including the following information:
- Type of object produced
- Physical file name where the object is written as a member
- How the member name of the object is derived
- Whether production is controlled by a generation option
- Whether the object can be modified after generation is performed

For more information about controlling and modifying generation and preparation of IBM i objects, see the *EGL Generation Guide* in the Rational Business Developer information center.

Table 5. Objects transferred to an IBM i host by the EGL preparation utility

File type	Physical file name	PF member name and generated file name	EGL build descriptor option	Modifiable
Objects generated for programs				
ILE COBOL program	QVGNCBLS	Program name *aliasName*.cbl	None	No
Runtime CL	QVGNCLS	Program name *aliasName*.clr	None	Yes
Objects generated for tables				
Table Binary Image	QVGNTAB	Table name *tablname*.tab	genDataTables	No
Objects generated for form groups				
Print services program (See note 3)	QVGNCBLS	Form group name *formgname*.cbl	genFormGroup, genHelpFormGroup	No
Form group module (See note 4)	QVGNMAPG	Form group name *formgnameFM*.fmt	genFormGroup, genHelpFormGroup	No
Objects generated for all member types (programs, tables, form groups)				
Generation variables file	QVGNEVF	*aliasName*.evf	None	No
Build plan	not applicable	*aliasName*.BuildPlan.xml	buildPlan	No
Objects generated for message tables				
Message file	QVGNMSGS	Member specified when generation was requested *tablename*.msg	genDataTables	Yes
Objects generated for file creation				

Table 5. Objects transferred to an IBM i host by the EGL preparation utility (continued)

File type	Physical file name	PF member name and generated file name	EGL build descriptor option	Modifiable
Data definition specification (DDS)	QVGNDDSS	File name as specified in EGL record definitions *filename*.dds	genDDSFile	Yes

Notes:

1. The generator produces ILE COBOL for the IBM i environments.
2. Generated programs, libraries, services, tables, and form group objects are environment dependent. All objects are generated for one environment and cannot be used in another environment.
3. This object is produced only if the form group contains print forms.
4. This object is produced only if the form group contains text forms.
5. *aliasName* is the alias property of the program. If an alias is not specified, it is the first seven characters of the program name.

Objects generated for programs

For programs, the following objects are generated:

- ILE COBOL programs
- Runtime CL programs
- Data definition specifications

ILE COBOL program

The generated program is an ILE COBOL program that contains the following data:

- Program control logic
- Logic for functions and I/O operations
- Data for both the program and program control logic

Runtime CL

This CL program is used only when the COBOL program is invoked remotely by using EGL client/server interface

The runtime CL sets commitment control and adds libraries to the IBM i library list when a program runs. The CL is generated from the fda24ebc.tpl and fda24eec.tpl templates, which can be customized.

The name of the runtime CL is as follows:

aliasName.clr

aliasName
Name of the program.

Data definition specification (DDS)

The generator produces IBM i data definition specifications (DDS) to create instances of IBM i physical and logical files that the application uses. The DDS that is produced is the result of the *indexed*, *relative* and *serial* record types that are used within the program being generated. The **genDDSFile** build descriptor option produces the DDS output type. The build script uploads the DDS files to the host system, but does not manage processing beyond that point.

Objects generated for data tables

For data tables, the following objects are generated:

- DataTable binary image files
- Message definitions

DataTable binary image file

The DataTable binary image file contains the contents of the runtime DataTable member, as defined by EGL. The DataTable contents are already converted to the code page of the target runtime environment. The DataTable contents are formatted to a program-defined structure, which might contain various data types. The contents are treated as binary data. You might not be able to view the contents outside of the scope of EGL and utilities.

The **genDataTables** build descriptor option produces the table binary image files.

Message definitions

For message tables, the generator produces a file that contains the raw message definitions. The preparation script file, FDAPREP, processes the message definitions file to create an IBM i native message file object (*MSGF type).

The **genDataTables** build descriptor option produces the message file, which is the IBM i implementation of the message table. When the build plan runs, it uploads the message file and invokes the preparation script to generate the message object on IBM i.

Objects generated for form groups

For form groups, the form group format module is generated.

Form group format module

The form group format module is a generated structure that describes the form layout for text forms in the form group. The generator builds the structure as a binary image file that is converted to the code page of the target system. This object is produced when you specify the **genFormGroup** or **genHelpFormGroup** build descriptor options and when the application has defined text forms in the form group.

Print services program

The print services program is a COBOL program that performs print I/O, output formatting, and set statements for a form group that contains print forms. This object is produced when you specify the **genFormGroup** or **genHelpFormGroup** build descriptor options.

Preparation to run generated programs

To prepare a for a generated program to run in the IBM i environment, use the IBM i Remote Build Server. The IBM i Remote Build Server is a component of the EGL run time. For details, see “Starting the IBM i remote build server” on page 10.

The build plan is an XML file that is created in the generation directory. The build plan launcher, which is a Java program, launches the build plan to prepare a program on IBM i.

When the program generates with the **prep** build descriptor option set to YES, the build plan launches automatically at the end of the generation. Otherwise, the

build plan can be launched manually by following the process described in "Launching the build plan manually" on page 11.

The build plan launcher uses the build plan and communicates with the build server to complete the preparation. The build plan contains all the information necessary to transfer the applicable generated files to IBM i and to build (compile and bind) the program.

A key component of the preparation is the preparation script. The preparation script, FDAPREP, is a REXX script which is installed as part of the runtime code and is described in "FDAPREP preparation script file for IBM i" on page 2.

Starting the IBM i remote build server

The remote build server is a program, named CCUBLDS, that runs as a job on the IBM i. The CCUBLDS program uses a TCP/IP port. After CCUBLDS starts, it runs continuously until the job is canceled. To invoke the build server, you must have an administrator user ID that is authorized to access user profiles. Also, the QEGL library must be in the library list when the build server starts.

The following example shows the command to start the build server job:

```
SBMJOB CMD(CALL PGM(*LIBL/CCUBLDS) PARM('-p' '2600')) JOB(CCUBLDS) JOBQ(QSYS/QSYSNOMAX)
```

In the example, the server port is 2600, but you can use any available port number.

Verifying the IBM i remote build server

After the build server starts, verify that it is running properly. At the Windows workstation where the preparation step will be run, complete these steps:

1. In the *com.ibm.etools.egl.distributedbuild* plug-in, find the directory that contains the `ccubldc.exe` program. Add this directory to the PATH environment variable.
2. From the command line, issue the following command:

   ```
   ccubldc -h host@port -au userid -ap password -b id -r 37 -k 1252
   ```

   host
   IP name or the IP address of the IBM i host machine

   port
   Port number of the build server

   userid
   User ID that the preparation client will use

   password
   Login password for the user ID on the IBM i host

The following example shows the expected response:

```
05/03/09 14:58:56 (c) Copyright, IBM Corp. 2001 Copyright (c) 2002 Rational Software Corporation
05/03/09 14:58:57 *** Success ***
05/03/09 14:58:57
Command: id
****************** Build Script Output Follows *****************
uid=926(USERID) gid=102(GROUPID) groups=102(GROUPID)
****************** End Of Build Script Output ******************
05/03/09 14:58:58 *-----------------------------------------------------------
```

Launching the build plan manually

You can create and save a build plan that you can invoke later. To launch a saved build plan, complete these steps:

1. Make sure that `eglbatchgen.jar` is in your Java class path. The file is automatically placed in your Java class path on the computer where you install EGL. The JAR file is in the following directory:

   shared_resources`\plugins\com.ibm.etools.egl.batchgeneration_`*version*`\runtime`

   shared_resources
   : The shared resources directory for your product, such as `C:\Program Files\IBM\SDP70Shared`. If you installed and kept a version of Rational® Application Developer before you installed your current product, you might need to specify the shared resources directory that was set up in the earlier installation.

   version
   : The installed version of the plug-in; for example, 7.0.0

2. Similarly, make sure that your PATH variable includes that directory.

   `wstools\eclipse\plugins\`
   `com.ibm.etools.egl.distributedbuild_`*version*`\runtime`

   version
   : The installed version of the plug-in; for example, 7.0.0

3. From a command line, enter the following command:

   `java com.ibm.etools.egl.distributedbuild.BuildPlanLauncher` *bp*

   bp The fully qualified path of the build plan file.

Preparing a DB2 program

When you declare an SQL record, you can use the **tableName** property to specify the name of the SQL table by using either the SQL naming convention of collection.tablename or the IBM i SYSTEM naming convention of *collection/tablename*. Tailor the OPTION parameter on the FDAPREP script to be *SQL (collection.tablename) or *SYS (collection/tablename), depending on the format you need. The default naming convention is *SQL.

When you tailor the FDAPREP script, make sure that the *APOSTSQL and *QUOTE values are part of the OPTION parameter.

Customizing compiler process options using VGNCBLOP

VGNCBLOP is a COBOL copybook member that is shipped with the product. The purpose of this member is to provide one place for you to set custom compiler options for all your EGL-generated COBOL programs. As shipped, this member contains only the following statement:

```
OPTIONS BLK NOMONOPRC
```

The EGL generator adds the following line to the COBOL source code so that the OPTIONS statement is processed by the COBOL compiler:

```
PROCESS COPY VGNCBLOP
```

If you want to modify VGNCBLOP, take one of the following actions:

- Edit the file in the shipped QEGL runtime library.

- Create a file named QCBLLESRC in a private library, include the modified VGNCBLOP member in that file, and ensure that the private library precedes QEGL in the library list.

Chapter 3. Running a generated program on an IBM i system

This chapter describes the information required to run EGL programs on an IBM i system.

Setting up the environment to run the generated programs

EGL Server for IBM i and the generated COBOL programs use the runtime job library list (*LIBL) to resolve all named object references. Before you start the program, you must set up the library list. Table 6 on page 14 lists the names and types of objects that EGL might use while running in the IBM i environment. EGL searches for these objects when the program runs by scanning the libraries that are in the library list. When you run EGL programs, add the installation library, QEGL, to the library list of the end user.

EGL uses the first object it finds that matches the target name in the libraries in the library list. EGL uses this first-found object in all cases of object resolution except for objects of *FILE type. For objects of *FILE type, EGL uses the first member it finds that matches the target name, after the first member is qualified with the correct file name. Multiple files with the same name might exist in the libraries named in the library list. EGL checks each library file until it finds the first instance of the member name.

There are two exceptions to using the library list to resolve object references by running programs:

- When EGL data tables and form groups are in the IBM i IFS file system for improved runtime performance
- In either of the following situations. In either case, SQL object resolution is independent of the library list.
  - When the object (table or view) was explicitly qualified when an SQL record was defined during development
  - When the object was implicitly qualified when the program using the record was compiled

Table 6. Names and types of objects used by EGL at run time

Object and library name	Type	Description
QVGN* QEGL	*PGM *SRVPGM	EGL Server for IBM i program and service program objects
QVGNMSGF QEGL	*MSGF	EGL Server for IBM i product message file
QVGNMAPG userlib	*FILE	Members of this file contain the 5250 form groups of the generated program. Members are named for the form group it contains.
QVGNTAB userlib	*FILE	Members of this file contain the data tables of the generated program. Members are named for the data table it contains.
QVGNPRNT QEGL	*FILE	This is the standard printer device file for program use of the Printer file. Usually, all jobs on the system share one of these objects.
QVGNMAP QEGL	*FILE	This standard display device file is used for interactive programs when they display maps. Usually, all jobs on the system share one of these objects.
mmmmnls userlib	*MSGF	The message table of a specific program, where *mmmm* is the message table prefix as defined to the program, and *nls* is the value of the **targetNLS** build descriptor option when the program was generated.
calltarg userlib	*PGM	Any target of the **call** or **transfer** statements coded within a program
filetarg userlib	*FILE	Any file named on record definitions used by EGL process options within a program

Note: The *userlib* library in Table 6 is the library that the application developer specified in the **destLibrary** build descriptor option.

Establishing a library list for a job

You can establish a library list for a job in several ways, but each method involves using an IBM i system command. You can also mix the methods. The initial IBM i library list is contained in the job description that the user profile references. For more information about the following IBM i system commands, enter the command at a command line entry on IBM i and request command prompting.

ADDLIBLE
Adds a library list entry

CHGCURLIB
Changes the current library

CHGSYSLIBL
Changes the system library list

Running EGL programs and passing arguments

Running EGL programs under IBM i

To run an EGL program in the IBM i environment, call the *PGM program object just as you would any other *PGM object on IBM i. You can run main and called EGL programs from menus, commands, command lines, or interlanguage program calls. The following examples show how to run an EGL program from an IBM i command line:

- To call a program without the use of arguments, use the following format:

  ```
  CALL aliasName
  ```

- To call a program that expects a parameter declared as CHAR(16), use the following format:

  ```
  CALL aliasName ('char arg literal')
  ```

- To call program that expects two parameters declared as CHAR (16) and DECIMAL (15,5), use the following format:

  ```
  CALL aliasName ('char arg literal' 1234)
  ```

Passing arguments and declaring parameters

Arguments are passed to EGL programs in the same way that other IBM i *PGM objects pass and receive arguments. Both main and called EGL programs can receive arguments. However, main programs have a more fixed argument format, and called programs offer parameter definition diversity. Always use a reference to pass an argument: pass a pointer to the argument.

Main programs

Main programs can receive the following arguments:

- A single record, as specified by the **inputRecord** property for the program
- A single form, as specified by the **inputForm** property for a textUI program
- A single VGUI record, as specified by the **inputUIRecord** property for a VGWebTransaction program

You can pass a data area from a non-EGL program to a main EGL program. The data area that you pass must have the following structure:

- A 2-byte length field that specifies the length of the entire data area being passed. This length is considered to be part of the argument.
- The actual data. Match or map this data exactly to the EGL record specified by the **inputRecord** property for the EGL program. If the data area does not match or map to the EGL record, design the EGL program logic to handle record fields that are not initialized.

The generated EGL program accesses only the portion of the record data that is within the lesser of either of the following length bounds:

- The length passed with the record data
- The length of the record specified by the **inputRecord** property

The details of the **inputRecord** property are important if you are using a non-EGL program to pass the argument, because the non-EGL program establishes the argument storage layout. EGL programs account for all internal structures and data typing when records with the same name are specified for the **transfer** or **show** statement or the **inputRecord** property for the target EGL program specify records with the same name.

The structure of the form specified by **inputForm** property is an EGL-specific format. You can only pass a form from one EGL textUI program to another EGL textUI program by using the EGL **show** statement. The structure of the VGUI record that the **inputRecord** property specifies is also an EGL-specific format. You can use the EGL **show** statement only to pass a VGUI record from one EGL VGWebTransaction program to another **show** statement.

Called programs

Called programs that are defined with parameters must receive matching arguments when they are called. Otherwise, you might receive unpredictable results. Do not reference an argument that either has not been received or has been received with a data type that is different from its declared matching parameter. If you attempt such a reference, a function check typically occurs.

For EGL-called programs, you can declare a maximum of 30 parameters. You can declare parameters as EGL data items, records, or forms.

Unlike a record that is passed to a main program as specified by its **inputRecord** property, the record arguments for called programs do not have the leading 2-byte length field preceding the record data.

EGL-called programs that are called from an IBM i command line or CL can receive literal numeral arguments if the corresponding EGL parameters are declared as decimal, 15 digits, 5 decimals. For more information about the literal data types, see the *IBM i CL Programmer's Guide*.

EGL run units

EGL programs operate in a run unit that is similar to that of ILE COBOL. You can consider the EGL run unit to be a subset of the ILE COBOL run unit, because the COBOL run unit can exist before and persist longer than the EGL run unit.

The first EGL program on the IBM i program call stack for a specific job binds the EGL run unit. Run units are scoped in a single job. As long as an EGL program is on the program call stack, an EGL run unit is active. Only one EGL run unit can be active in a job at any time. This difference is the most obvious difference between EGL run units and ILE COBOL run units.

Main or called EGL programs can initiate an EGL run unit. An EGL main program can exist in an EGL run unit only if the main program is the initiating application. Main programs cannot be called from a program that initiates an EGL run unit, even if the program is called from a non-EGL program while an EGL run unit is active.

Using tables with run units

The resource type of the data tables in an EGL run unit is specific to EGL and is not a function. EGL data tables are static entities, which perform quick table search programming tasks. EGL programs in the same EGL run unit can share the data tables. Therefore, the data tables are loaded into storage from a database file. Usually, data table in-storage scoping ends with the EGL run unit in which the data tables are loaded. However, two factors that are associated with managing EGL data tables warrant the in-storage scoping of data tables to extend beyond the EGL run unit:

- The time necessary to acquire memory, to read a data table from a database file, and to load the table into memory

- The need to retain modified data tables for use by another EGL run unit, thereby serving as a quick program-to-program communication method

The in-storage scoping for EGL data tables extends beyond the EGL run unit in the following situations:

- The **resident** property is set to YES
- The run unit is ended by the **transfer to program** statement without specifying that the target program is a non-EGL program. Instead, the target program is externally defined either on the **transfer to program** statement or in the linkage options part.

Because the in-storage scoping extends beyond the run unit in those situations, the program eventually enters the EGL run unit again. When the program reenters the run unit, the resident tables of the previous EGL run unit need to remain available in storage. When in-storage scoping for an EGL data table extends beyond the EGL unit, a program can reenter the EGL run unit.

The **transfer to program** statement is the non-committal method of ending an EGL run unit. All other methods close the resident data table, all other data tables, and all other EGL resources.

Using activation groups with run units

EGL run units typically correlate on a one-to-one basis with ILE COBOL run units and ILE activation groups. When an EGL program initiates or begins a run unit, a named activation group is also initiated. When you use a named activation group, you can make sure that all EGL programs that run in a job share the same resources, in terms of ILE resource management.

If your application system consists of non-EGL and EGL programs, you can either add your non-EGL programs to the named activation group or use a different activation group. When you decide whether to add to a named activation group or use a different activation group, consider two important aspects:

1. Sharing commitment control logical units of work
2. Sharing database file Open Data Paths

If you want to share both ILE resources, use the named activation group. If you want to isolate the ILE resources in terms of their use by EGL and non-EGL program, use different activation groups.

The ILE activation group name is established when EGL programs are in the preparation phase of application development. The EGL build script, which creates IBM i program objects, names the activation group in the IBM i command template for the `CRTPGM` command. The keyword is ACTGRP, and the default is ACTGRP(QEGL).

Chapter 4. Diagnosing problems during run time

If you encounter problems while running your EGL programs, you can diagnose problems by using diagnostic commands and capture error information by using IBM i First Failure Data Capture.

Diagnostic commands

You can use IBM i standard diagnostic commands to diagnose problems with an EGL program, such as the following commands:

- `ADDTRC` (Add Trace Statement)
- `STRJOBTRC` (Start Job Trace)
- `ENDJOBTRC` (End Job Trace)
- `PRTJOBTRC` (Print Job Trace Data)
- `STRDBG` (Start Debug)
- `ENDDBG` (End Debug)

For most problems, you need the following information when you contact the IBM Support Center:

- The runtime job log, which records all messages, including second-level text.

  To make sure that second-level text is included, change the job before you use the `CHGJOB LOG(4 00 *SECLVL)` command to start the failed scenario. When the job ends, the job log is spooled to the assigned output queue. Usually, the most important information in the job log is the escape message that initiates the abnormal condition, which caused the EGL program to end. The key pieces of diagnostic information are sending the message, the program receiving the message, and the instructions being sent to the program. The other messages are also important. Be sure to inspect and report the entire job log of information.
- The ILE COBOL compiler listing, which includes the following information:
  - EGL annotated statements (use the **commentLevel** build descriptor option)
  - ILE COBOL source statements (use the OPTION(*SOURCE) compiler option)
- Any spooled files that might have been created as a result of the job ending abnormally, such as dumps or display job snapshots. You can find all spooled file output from a job by using the `WRKJOB` command to work with the job. Then you can select the option from the job menu to display spooled output.

IBM i First Failure Data Capture

The IBM i First Failure Data Capture component runs unit data dumps to send to the IBM Support Center. EGL programs link into the component when report software error (QpdReportSoftwareError), an IBM i System Programming Interface (SPI), runs a function check (abends).

EGL also uses the report software error SPI function when illogical conditions that might lead to a function check (abend) are detected during run time. In these cases, a unique signature that is associated with the error condition is provided to the system service. The system service can use the unique signature to scan a service database for the same signature and possible PTFs that can be applied. In some cases, using a unique signature to scan the database can help to solve a problem faster.

When job log messages indicate that the IBM i system problem log is updated, use the IBM i command DSPPRB (Display Problem) to view a list of the most recent problems captured on the system. Then, select the option to display the problem that is associated with QVGNHS. QVGNHS is the service program that constitutes the majority of EGL Server for IBM i. A procedure of this service program issues the SPI function to record the problem. When the problem is displayed, you can use the menu selections and function keys to display more information, such as spooled files, problem history files, and APAR libraries. The spooled files contain dump data and a copy of the job attributes at the time of the dump.

For more information about using the automated system problem log in an automated manner, see product documentation for SystemView® System Manager.

Chapter 5. Exception codes

Rational COBOL Runtime issues the following exception codes:

9980 No library function with specified signature exists; you may need to regenerate the library

9981 EGL runtime exception

9986 Segmented converse exception; internal EGL use only

9988 User thrown exception

9989 DL/1 exception

9990 File I/O exception

9991 MQ I/O exception

9992 SQL exception

9993 Service invocation exception

9994 Service binding exception

9996 Invocation exception

9997 Null value exception

9998 Index out of bounds exception

9999 Type cast exception

Chapter 6. Messages

This section describes the EGL Server messages.

You can view the server messages online on IBM i by using the Work with Message Description command: `WRKMSGD MSGF(QEGL/OVGNMSGF)`.

EGL programs use the standard IBM i message handling functions to call non-EGL programs. The job log automatically logs diagnostic information during run time.

Message format

Each message consists of a message identifier (for example, ELA00023P) and message text. The text is a short phrase or sentence that describes the error condition.

The message identifier consists of two fields: prefix and message number. The format of the message identifier is *xxxnnnn*, where:

xxx Message prefix

nnnn Message number associated with the error condition that caused the message to be displayed.

Escape messages

EGL programs send these messages as IBM i ESCAPE type messages to the program queue of the calling non-EGL program. The calling program must monitor these messages to avoid an IBM i function check.

GEN9001 EGL Server MAIN shell cannot invoke the target program %1.

Explanation: The diagnostic messages that precede this message in the job log explain the nature of the error. In most cases, the application or system programmer must adjust your application system to correct the problem.

User response: Either print the job log or record the messages along with the following:

- The from program name.
- The to program name.
- The instruction numbers.

You can view or print the job log with the DSPJOBLOG command. If no diagnostic messages precede this message, make sure that your job logged all messages by checking the Message Logging or LOG value of your job definition or job description, depending on whether the job is interactive or batch. For interactive jobs, the command DSPJOB OPTION(*DFNA) will display the Message Logging value.

Contact your application or system programmer with the information you gathered.

GEN9002 EGL Server encountered a program error which caused the run unit to end.

Explanation: The diagnostic messages that precede this message in the job log explain the nature of the error. In most cases, the application or system programmer must adjust your application system to correct the problem.

User response: Either print the job log or record the messages along with the following:

- The from program name.
- The to program name.
- The instruction numbers.

You can view or print the job log with the DSPJOBLOG command. If no diagnostic messages precede this message, make sure that your job logged all messages by checking the Message Logging or LOG value of your job definition or job description, depending on

whether the job is interactive or batch. For interactive jobs, the command DSPJOB OPTION(*DFNA) displays the Message Logging value.

Contact your application or system programmer with the information you gathered.

Diagnostic and informational messages

The following messages are sent as DIAGNOSTIC or INFORMATIONAL type messages to the program queue of the calling program. These messages are automatically posted in the job log file. Non-EGL programs that call EGL programs cannot monitor the activities of these messages. Use the WRKJOB (Work with JOB) command to view the job log. The Message Logging job attribute might filter some or all of these messages in some way. To make sure that all messages are posted in the job log, use a message logging value of LOG(4 00 *SECLVL) in your IBM i jobs. For more information, see the WRKJOB and CHGJOB (Change Job) IBM i commands.

GEN0002 A new level of EGL Server for IBM i is required for program *program_name*

Explanation: The generated COBOL program that is specified in the message that is attempting to run is not compatible with the installed version of EGL Server for IBM i.

User response: Contact the system administrator for the EGL Server for IBM i that should be installed.

GEN0005 Date entered is not valid for defined date format *format_value*

Explanation: The data entered into a form field that is defined with a date format property does not meet the requirements of the format specification, or the month or day of the month is not valid.

It is not necessary to enter the separator characters shown in the message, but if you omit them, enter leading zeros. For example, if the **dateFormat** is MM/DD/YY, you can enter 070494.

User response: Enter the date in the format displayed in the message.

GEN0009 Overflow occurred because the target item is too short

Explanation: The target of a move or arithmetic assignment statement is not large enough to hold the result without truncating significant digits. If the program logic did not handle the overflow exception that occurred, then the program ends.

User response: Change the program in one of the following ways:

- Increase the number of significant digits in the target data item.
- If the program sets the **V60ExceptionCompatibility** property to YES, define the program logic to handle the overflow condition by using **VGVar.handleOverflow** and **sysVar.overflowIndicator**.
- If the program sets (or defaults) the **V60ExceptionCompatibility** property to NO, define the program logic to include a **try ... onException** block that can catch overflow exceptions.

GEN0014 A replace was attempted without a preceding get for update on *record_name***.**

Explanation: To perform a **replace** request, a **get forUpdate** or **open forUpdate** statement must first read the specified record. The **read for update** might have been lost as the result of a commit or rollback.

This error also occurs if the **replace** request is associated with a specific **get** or **open** statement that was not used to select the record.

User response: make sure that the **replace** statement and the corresponding **get forUpdate** or **get forUpdate** use the same record variable or resultSetID. Also, verify that the sequence of statements is appropriate. Use the EGL debugger, to step through the program.

GEN0021 An error occurred in program *program_name* **on statement number** *statement_number*

Explanation: The actual error that identifies the problem is explained in messages following this message in the job log.

User response: Correct the program and then generate the program again.

GEN0022 Form group format module *form_group* **could not be loaded**

Explanation: The form group format member is a generated binary file that contains attributes that describe the format and constant fields for text based forms in a form group. Form group format members are stored as members and are typically found in the job's library list.

Refer to the job log for error messages that precede this message to obtain additional information about the error.

User response: Contact your application or system programmer. Report the sequence of messages

including and preceding this message.

GEN0023 Call to data-table program *table_name* **was not successful**

Explanation: A data table is a generated binary file that contains program data. Data tables are stored as members and are typically found in the library list of the job.

Refer to the error messages that precede this message in the job log for more details about the error.

User response: Contact your application or system programmer, and report the sequence of messages including and preceding this message.

GEN0024 EGL conversion table *table_name* **could not be found.**

Explanation: Either the name specified on the **sysLib.convert** call was not a member of the QEGL/QEGLSCTB file, or the member that was found is not a conversion table.

User response: Verify that the correct conversion table name was specified in the **sysLib.convert** call. If the table name was not correct, change the EGL program and generate it again. If the table name is correct, verify that the correct conversion table was installed. The conversion table is a member in the file QEGLSCTB in QEGL library.

GEN0026 A calculation caused a maximum-value overflow.

Explanation: During a calculation in an arithmetic statement, an intermediate result exceeded the maximum value (18 significant digits). The maximum value is based on the definition of the target variable, which can be up to either 18 or 31 significant digits based on the value of the **maxNumericDigits** build descriptor option. Maximum value overflow also occurs when a value is divided by zero. This error can only occur when you set the **checkNumericOverflow** build descriptor option to YES. If the program logic does not handle the overflow exception that occurred, the program ends.

User response: Correct the program logic in one of the following ways:

- Increase the number of significant digits in the target data item.
- If the program sets the V60ExceptionCompatibility property to YES, define the program logic to handle the overflow condition by using VGVar.handleOverflow and sysVar.overflowIndicator.
- If the program sets (or defaults) the V60ExceptionCompatibility property to NO, define the program logic to include a **try ... onException** block that can catch overflow exceptions.

GEN0027 The data on character-to-numeric move is not valid.

Explanation: The statement in error involves a move from a character to a numeric data item. The character data item contains nonnumeric data.

User response: Change the program to make sure that the source operand contains valid numeric data.

GEN0031 Call to *program_name* **was not successful**

Explanation: The program specified could not be found. The programs that are being called are expected in library list for the job.

Refer to the error messages that precede this message in the job log for additional information about the error.

User response: Contact your application or system programmer, and report the sequence of messages including and preceding this message.

GEN0033 Invocation of function *function_name* **returned exception code** *error_code***.**

Explanation: An error occurred while calling the specified function, because the arguments passed to the function were not correct.

The run unit ends.

User response: Correct the program so that its expected arguments are passed to the function.

GEN0034 Program *program_name* **was declared as a main program and cannot be called.**

Explanation: The specified program was declared as either a main textUI program or as a main basic program. Another program cannot call it.

User response: If you can use the **call** statement to invoke the specified program, modify the program so that the specified program is declared as a called program. If specified program is a main program, change the program to use the **transfer** statement to invoke the specified program.

GEN0035 Data type error in input - enter again

Explanation: The data in the first highlighted field is not valid numeric data. The field was defined as numeric.

User response: Enter only numeric data in this field, or press a validation bypass key to bypass the validation check. In either situation, the program continues.

GEN0036 Input minimum length error - enter again

Explanation: The data in the first highlighted field does not contain enough characters to meet the required minimum length.

User response: Enter enough characters to meet the required minimum length, or press a validation bypass key to bypass the validation check. In either situation, the program continues.

GEN0037 Input not within defined range - enter again

Explanation: The data in the first highlighted field is not within the range of valid data that is defined for this item.

User response: Enter data that conforms to the required range, or press a validation bypass key to bypass the validation check. In either situation, the program continues.

GEN0038 Table edit validity error - enter again

Explanation: The data in the first highlighted field does not meet the validation data table requirement that is defined for the variable field.

User response: Enter data that conforms to the table edit requirement, or press a validation bypass key to bypass the validation check. In either situation, the program continues.

GEN0039 Modulus check error on input - enter again

Explanation: The data in the first highlighted field does not meet the modulus check that is defined for the variable field.

User response: Enter data that conforms to the modulus check requirements, or press a validation bypass key to bypass the validation check. In either situation, the program continues.

GEN0040 No input received for required field - enter again

Explanation: No data was typed in the field that the cursor designated. The field is required.

User response: Enter data in this field or press a validation bypass key to bypass the validation check. Blanks or nulls will not satisfy the data input requirement for any type of field. In addition, zeros will not satisfy the data input requirement for numeric fields. The program continues.

GEN0041 Property msgTablePrefix was not specified for a program: Message *user_message***, NLS - code** *language_code***.**

Explanation: The program tried to display a message from the message table by using the *converseLib.validationFailed* system function. However, the program did not specify a value for the **msgTablePrefix** property.

User response: Take either of the following steps:

- Assign a valid value to the **msgTablePrefix** property and generate the program again.
- Change the program to avoid using the **converseLib.validationFailed** system function and generate the program again.

GEN0045 Error reading message *user_message***, NLS code** *language_code***, status** *status_code***.**

Explanation: A user message was requested. Refer to the previous message in the job log for additional information about the error.

User response: Most problems occur because the message file or the specific message cannot be found or access to the message file is not authorized. If the program can not find the message file and you know the library name that contains the message file, you can add the library to your library list (ADDLIBLE command). For other problems, contact your system or application programmer.

GEN0046 Call to print services program *program_name* **was not successful.**

Explanation: The specified print services program cannot be found in the library list of jobs.

Refer to the error messages that precede this message in the job log for additional information about the error.

User response: Contact your application or system programmer and report the sequence of messages including and preceding this message.

GEN0050 Significant digits for field exceeded - enter again

Explanation: One of the following reasons caused this error:

1. The data entered into a numeric field exceeded the maximum size of the field.
2. The numeric field was defined with decimal places, a sign, currency symbol, or numeric separator edits.
3. The number of significant digits that can be displayed within the formatting criteria was exceeded by the input data; the number entered is too large.

4. The number of significant digits cannot exceed the field length, minus the number of decimal places, minus the places required for formatting characters.

User response: Enter a number with fewer significant digits.

GEN0051 **Form** *form* **was not found in form group** *form_group***.**

Explanation: The specified form name is not in the form group.

User response: Correct the program or form group and then generate the form group and the program again.

GEN0057 **Delete attempted without preceding update on record** *record_name***.**

Explanation: A **delete** statement ran for a record that was not successfully read by a **get forUpdate** statement. The read for update might have been lost as the result of an commit or rollback.

User response: Verify the following:

- The **delete** statement and the corresponding **get forUpdate** correctly use the same record variable name or resultSetID.
- The sequence of statements is appropriate. Use the EGL debugger to step through the program.

GEN0073 **SQL error, command =** *command_name***, SQL code =** *sql_code***.**

Explanation: The SQL database manager returned an error code for an SQL I/O statement. If the program logic does not handle the SQL exception that occurred, then the program ends.

This message is followed by message GEN0074, which displays the substitution variables associated with the SQLCODE.

User response: Determine the cause of the problem from the SQL code and the SQL error information. Refer to the appropriate database manager messages and codes documentation for information on the SQL code and error information. Correct either the database definition or change the program to handle the SQL exception.

GEN0074 **SQL error message:** *error_code*

Explanation: This message accompanies message GEN0073 when an SQL error occurs. It displays the relational database manager description of the error and is repeated as many times as necessary to display the complete description.

User response: Use the information from this message and GEN0073 to correct the error.

GEN0076 **Invalid data is used in a character-to-hexadecimal assignment or comparison.**

Explanation: The current statement involves either a move from a character data item to a hexadecimal data item, or a comparison between a character data item and a hexadecimal data item. The characters in the character data item must occur in the following set for the move or compare to complete:

a b c d e f A B C D E F 0 1 2 3 4 5 6 7 8 9

One or more of the characters in the character data item is not in this set. This condition causes a program error.

User response: Change the program to make sure that the character data item contains valid data when the character-to-hexadecimal move or compare operation occurs. In text form fields, you can use the **isHexDigit** validation property to make sure that user input contains only valid characters.

GEN0080 **Hexadecimal data is not valid**

Explanation: The data that the cursor identifies in the variable field must be in hexadecimal format. One or more of the characters that you entered does not occur in the following set:

a b c d e f A B C D E F 0 1 2 3 4 5 6 7 8 9

User response: Enter only hexadecimal characters in the variable field. The characters are left-justified and padded with the character zero. Embedded blanks are not allowed.

GEN0086 *statement* **- No active open or get forUpdate is in effect.**

Explanation: The error occurred because of one of the following reasons:

- A **get next** statement cannot be run because a related **open** did not run previously in the same program
- A **replace** or **delete** statement cannot run because a related **open forUpdate** and **get next** or a related **get forUpdate** did not run previously in the same program.

All rows selected for retrieval or update are released when a called program returns to the calling program.

User response: make sure that the second statement (**get next**, **replace**, or **delete**) correctly uses the record name or resultSetID to match the first statement (**open** or **get**).

Also, make sure that the sequence of statements is appropriate. Use the EGL debugger, to step through the program.

GEN0093 **An error occurred in program** *program_name*, **function** *function_name*.

Explanation: An error occurred in the specified function. Other information about the error is in the messages that follow this message.

If the function name starts with EZE, it indicates that the problem occurred in the generated control logic for the program, not in a function in the EGL program itself.

User response: Refer to the job log for error messages following this message to obtain additional information about the error.

GEN0096 **A data operand of type MBCHAR is not valid.**

Explanation: An operand in a move statement involving an item of type MBCHAR contains an invalid mixture of double-byte and single-byte data.

User response: Verify that all operands in the move statement contain valid data.

GEN0109 **Input form must be form** *form_name* **rather than form** *form_name*, **for program** *program_name*.

Explanation: The form that the program received is not the value that is specified for the **inputForm** program property. This error occurs when the program starts.

User response: Verify that the transferring program specifies the correct form name on the **show** statement and that the receiving program specifies the correct value for the **inputForm** property.

GEN0111 **Length of input form** *form_name* **is not valid.**

Explanation: The length of the specified form that a program receives is not the length that is defined for the form in the program.

User response: Use the same form definition when generating both the program that receives the input form and the program that issues the show statement.

GEN0119 **Programs** *program_name* **and** *program_name* **are not compatible.**

Explanation: A program that a **transfer** or **call** statement started is not compatible with the initial program in the transaction or job because the program was generated for a different environment.

User response: Generate one or both programs again so that the target environments for the programs are the same.

GEN0127 **A requested function is not supported for map** *map_name*, **map group** *group_name*.

Explanation: A program requested a form function that is not supported for the specified form and form group. The form group was modified between the time that the form group generated and the time that the program generated. Some functions that were included for the form or form group when the program generated were not specified for the form group when the form group generated. For example, a **helpForm** or **msgField** might have been specified for the form at the time that the program generated, but were not present when the form group generated.

User response: Check the form properties and the program, then generate the program again with the **genFormGroup** build descriptor option set to YES.

GEN0184 **Program** *program_name* **and form services program** *form_program* **are not compatible.**

Explanation: The specified program and form services program are generated for different systems.

User response: Generate the form services program for the same environment as the program.

GEN0185 **Length of** *size* **for record** *record_name* **is not valid and conversion ended.**

Explanation: A variable length record cannot be converted between the workstation format and host format because of one of the following conditions:

- The record is longer than the maximum length that is defined for the record.
- The record data ends in the middle of a numeric field.
- The record data ends in the middle of a DBCHAR character.
- The record data ends in the middle of a SO/SI string.

User response: Modify the program to set the record length so that it ends on a valid field boundary.

GEN0186 **An operand of type MBCHAR in a conversion operation is not valid.**

Explanation: An MBCHAR field cannot convert from EBCDIC to ASCII or from ASCII to EBCDIC because the double-byte data value is not valid.

User response: Modify the program to make sure that fields of type MBCHAR are valid in the records to be converted.

GEN0187 **Conversion table %1 does not support double-byte character conversion.**

Explanation: An MBCHAR field cannot convert from EBCDIC to ASCII or from ASCII to EBCDIC because the specified conversion table does not include conversion tables for double-byte characters.

User response: Modify the program to specify a conversion table that contains the double-byte conversion tables that are valid for the DBCHAR or MBCHAR data being converted. Refer to the topic "Data conversion" in *EGL Generation Guide*.

GEN0188 **Conversion Error. Function:** *function_name*, **Return Code:** *error_code*, **Table:** *table_name*

Explanation: A system function was called to perform code page conversion for data used in a client/server program. The function failed.

Possible causes for the failure:

- The code pages that are identified in the conversion table are not supported by the conversion functions on your system.
- For double-byte character conversion where the source data is in ASCII format, the source data was created under a different double-byte character code page than the code page that is currently in effect on the system.

User response: Modify the program to specify a conversion table that contains the double-byte conversion tables that are valid for the DBCHAR or MBCHAR data being converted. Refer to the topic "Data conversion" in *EGL Generation Guide*.

GEN0191 **Program** *program_name*, **generation date** *date*, **time** *time*.

Explanation: An error occurred in the specified program. Changes to individually generated components of the program might have caused the error. Refer to the error messages in the job log for additional information.

User response: Verify that the generation date and time of the program is the same as those of the other generated components.

GEN0192 **Print services program** *program_name*, **generation date** *date*, **time** *time*.

Explanation: An error occurred in the print services program. Changes to individually generated components of the program might have caused the error. Refer to the error messages in the job log for additional information about the error.

User response: Verify that the generation date and time of the program is the same as those of the other generated components.

GEN0195 **Form group format module** *form_group*, **generation date** *date*, **time** *time*.

Explanation: An error occurred in the specified form group. Changes to individually generated components of the program might have caused the error. Refer to the diagnostic messages in the job log for additional information about the error.

User response: Verify that the generation date and time of the program is the same as those of the other generated components.

GEN0210 **Service number** *number* **is not valid**

Explanation: An attempt was made to start an EGL runtime routine that does not exist or that is not valid.

User response: Generate and compile the program again to make sure the generated COBOL code has not been modified. Afterward, run the refreshed program. If the problem persists, have the system administrator do all of the following actions:

- Record the service number from this message.
- Print the job log.
- Record the scenario under which this message occurs.
- Obtain the COBOL listing and source for the failing program.
- Use your electronic link with IBM Service (for example, IBMLINK) if one is available, or contact the IBM Support Center.

GEN0232 **Form** *form_name* **in form group** *form_group* **is not declared or is not supported.**

Explanation: The specified form does not exist or is not defined for the type of device being used.

User response: Specify the correct **screenSizes** property for the form and then generate the form group again.

GEN0233 *error_code* **error on file** *file_name*, **sysVar.errorCode=***value*.

Explanation: An I/O operation failed for the specified file. This message specifies the COBOL verb performed and the EGL file name associated with the operation.

sysVar.errorCode contains either the COBOL status key value or EGL file return code.

User response: Use the appropriate COBOL publication or the EGL reference guide to diagnose the error, and take the recommended corrective action.

GEN0260 *number* **bytes of VGUI record do not fit in** *buffer_size* **byte buffer.**

Explanation: The program issued a **converse** or **show** statement for a VGUI record. There was not enough room in the communications buffer for the record. The buffer needs space for the record plus any message information written using the **sysLib.setError** system function.

User response: Modify the program to reduce the size of the VGUI record, or write fewer or smaller error messages.

GEN0261 **sysLib.setError message information and inserts do not fit in** *buffer_size* **byte buffer.**

Explanation: The program invoked the **sysLib.setError** system function one or more times to write messages that are associated with a VGUI record. The information that is associated with the last message written does not fit into the buffer that the program uses to communicate with the user.

User response: Modify the program to write fewer or smaller error messages.

GEN0262 **VGWebTransaction program and VGUI record bean** *record* **are incompatible.**

Explanation: A VGWebTransaction program was started with information from a VGUI record bean that is not known to the VGWebTransaction program, or whose definition is not compatible with the VGUI record definition with which the program was generated.

User response: make sure that the specified VGUI record is specified in the **inputUIRecord** property for the program. Generate the program and the Java beans using the same VGUI record definition.

GEN0263 **Number of elements value** *number_elements* **is out of range for structured field array at offset** *array_offset***.**

Explanation: A VGWebTransaction program could not write a VGUI record because the value in the number of elements item for a structured field array in the record was less than 0 or greater than the maximum size that is defined for the array.

User response: Correct the program logic so that it sets the value of the number of elements item to a value within the allowed range.

GEN0264 **Input data entered by the user does not fit in VGUI record.**

Explanation: A VGWebTransaction program received input data from the Web server that does not fit in the VGUI record. The VGWebTransaction program and the Java bean that are associated with the VGUI record might have been generated at different times with incompatible VGUI record declarations.

User response: Generate the program and the Java beans using the same VGUI record definition. Contact IBM support if this does not correct the problem.

GEN0265 **Segmented converse is not supported when local variables or function parameters are in the run-time stack.**

Explanation: The program issued a **converse** statement with **sysVar.segementedMode** set to 1 (segmented converse), and at least one of the functions in the current function stack uses parameters or local items or records. The generated program cannot save parameters or local storage data over a segmented converse.

User response: Verify the following and make the appropriate change:

- The functions on the run-time stack do not have parameters or local variables.
- The **converse** is not segmented.

GEN0266 **MQ function** *function_name***, Completion Code** *completion_code***, Reason Code** *reason_code***.**

Explanation: The MQ function did not complete successfully, as indicated by the following completion codes:

- MQCC_WARNING
- MQCC_FAILED

MQSeries® sets the reason for the completion code in the reason code field. The following are some common reason codes:

- 2009 (Connection broken)
- 2042 (Object already open with conflicting options)
- 2045 (Options not valid for object type)
- 2046 (Options not valid or not consistent)
- 2058 (Queue manager name not valid or not known)
- 2059 (Queue manager not available for connection)
- 2085 (Unknown object name)
- 2086 (Unknown object queue manager)
- 2087 (Unknown remote queue manager)
- 2152 (Object name not valid)
- 2153 (Object queue-manager name not valid)
- 2161 (Queue manager quiescing)

- 2162 (Queue manager shutting down)
- 2201 (Not authorized for access)
- 2203 (Connection shutting down)

User response: Refer to the *MQSeries Application Programming Reference* for additional information about MQSeries® completion and reason codes.

GEN0267 Queue Manager Name *queue_name***.**

Explanation: This is the name of the queue manager that is associated with the failing MQ function call that is listed in message GEN0266.

If the failing MQ function was MQOPEN, MQCLOSE, MQGET, or MQPUT, the name identifies the queue manager that is specified with the object name when the queue was opened. Otherwise, the name is the name of the queue manager to which the program is connected (or trying to connect).

If the queue manager name is blank, the queue manager is the default queue manager for your system.

User response: Refer to the *MQSeries Application Programming Reference* for additional information about MQSeries completion and reason codes that are associated with GEN0266.

GEN0268 Queue Name *queue_name*

Explanation: This is the name of the queue object that is associated with the failing MQ function call that is listed in message GEN0266.

User response: Refer to the *MQSeries Application Programming Reference* for additional information about MQSeries completion and reason codes that are associated with GEN0266.

GEN2001 The table *table_name* **is not valid for program** *program_name***.**

Explanation: The error occurred because of one of the following reasons:

- The DataTable version is not compatible with the current level of IBM EGL Server and the running application.
- The DataTable was generated for an ASCII-based EGL runtime environment.
- The data is corrupted.
- The DataTable could not open.

User response: Replace the table with the correct generated version.

If the reason code indicates that the table data is corrupted, make sure that the table was transmitted to the host system as a binary image file.

If the reason code indicates the table was generated for an ASCII-based host system, make sure that the table regenerates for the same target system as the program that is attempting to use it.

If the reason code indicates the table could not be opened, see previous messages in the job log.

GEN2002 EGL Server does not support DBCHAR data type.

Explanation: The EGL Server does not support the DBCHAR data type because COBOL does not support DBCHAR.

User response: Change EGL DBCHAR primitive types to MBCHAR data types and generate the program again.

GEN2004 Character conversion from CCSID *from_characterset* **to** *to_characterset* **is not supported**

Explanation: The CCSID character conversion is not supported between the two specified Coded Character Set IDs (CCSID).

User response: Verify that the specified Coded Character Sets IDs (CCSID) are valid and that conversion between the two CCSIDS is supported. You might have to generate the EGL program again.

GEN2005 Error *error_code* **occurred when converting record** *record***.**

Explanation: The **sysLib.convert** function encountered an error during the call.

User response: Verify that the specified program logics record contains data that matches its definition. Generate the EGL program again.

GEN2006 The form group *group_name* **is not valid for program** *program_name***.**

Explanation: The error occurred because of one of the following errors:

- The form group data is corrupted.
- The form group could not be opened.

User response: Replace the specified form group with the correct generated version.

If the reason code indicates that the form group data is corrupted, make sure that the form group was transmitted to the host system as a binary image file.

If the reason code indicates the form group could not be opened, see previous messages in the job log.

GEN2007 Press Enter to continue.

Explanation: To continue processing, click **Enter**.

GEN7025 Error encountered allocating memory.

Explanation: An error was encountered while allocating memory. The system has run out of memory.

User response: Make sure that you have enough memory on your system, as specified in the software and hardware requirements for the product. Stop running the execution of some of the other programs on your system.

1. Record the message number and the message text. The error message includes the information on where the error occurred and the type of internal error.
2. Record the situation in which this message occurs.
3. Contact IBM support for additional assistance.

GEN7030 The format of the data descriptor is incorrect. The hex value of the data descriptor in error is *value*.

Explanation: The format of the data descriptor is incorrect. A header descriptor is found within the data descriptor.

User response: Do the following:

1. Record the message number and the message text. The error message includes the information on where the error occurred and the type of internal error.
2. Record the situation in which this message occurs.
3. Contact IBM support for additional assistance.

GEN7035 The format of the data descriptor is incorrect.

Explanation: The format of the data descriptor is incorrect. An End Of Description descriptor is not found.

User response:

1. Record the message number and the message text. The error message includes the information on where the error occurred and the type of internal error.
2. Record the situation in which this message occurs.
3. Contact IBM support for additional assistance.

GEN7040 The format of the data descriptor is incorrect. An unknown data code data_code was found.

Explanation: The format of the data descriptor is incorrect. An unknown data code was found in the data description.

User response:

1. Record the message number and the message text. The error message includes the information on where the error occurred and the type of internal error.
2. Record the situation in which this message occurs.
3. Contact IBM support for additional assistance.

GEN7055 The Conversion Descriptor structure is not valid.

Explanation: The Conversion Descriptor structure CMCVOD required by the conversion routine is not correct.

User response:

1. Record the message number and the message text. The error message includes the information on where the error occurred and the type of internal error.
2. Record the situation in which this message occurs.
3. Contact IBM support for additional assistance.

GEN7065 The data descriptor for parameter *parameter_name* is not valid.

Explanation: The data descriptor for the parameter is not valid.

User response:

1. Record the message number and the message text. The error message includes the information on where the error occurred and the type of internal error.
2. Record the situation in which this message occurs.
3. Contact IBM support for additional assistance.

GEN9003 EGL Server encountered a critical internal processing error.

Explanation: A critical internal processing error was detected. This may include corrupted run unit control blocks, an unexpected return code from an internal function, or illogical code path entry.

Refer to the job log for error messages that precede this message to obtain additional information about the error. In most cases, the application or system programmer must adjust your application system to correct the problem.

User response: Either print the job log or record the messages along with the following information:

- The from program name.
- The to program name.
- The instruction numbers.

You can view or print the job log with the DSPJOBLOG command. If no diagnostic messages precede this

message, make sure that your job logged all messages by checking the Message Logging or LOG value of your job definition or job description, depending on whether the job is interactive or batch. For interactive jobs, command DSPJOB OPTION(*DFNA) will display the Message Logging value.

Contact your application or system programmer with the information you gathered.

GEN9004 EGL Server COBOL error handler was invoked to end the run unit.

Explanation: A function check caused the run unit to end. A database rollback has been issued and heap storage has been released.

Refer to the job log for error messages that precede this message to obtain additional information about the error. In most cases, the application or system programmer must adjust your application system to correct the problem.

User response: Either print the job log or record the messages along with the following information:

- The from program name.
- The to program name.
- The instruction numbers.

You can view or print the job log with the DSPJOBLOG command. If no diagnostic messages precede this message, make sure that your job logged all messages by checking the Message Logging or LOG value of your job definition or job description, depending on whether the job is interactive or batch. For interactive jobs, command DSPJOB OPTION(*DFNA) will display the Message Logging value.

Contact your application or system programmer with the information you gathered.

GEN9937 An error occurred when trying to invoke a Web Service function.

Explanation:

User response:

GEN9938 An error occurred when trying to invoke a service function.

Explanation:

User response:

GEN9940 Binding Key: *key*

Explanation:

User response:

GEN9941 An error occurred when trying to invoke a Web Service function, JNI setup- error *error_value*

Explanation:

User response:

GEN9942 Web Service Proxy: *proxy*

Explanation: This message provides the service property name in which a problem occurred. Other related messages provide the information about the actual cause of the error.

User response: None required.

GEN9946 Binding key cannot be resolved in binding program *program_name***.**

Explanation:

User response:

GEN9948 Reference name *reference_name*

Explanation: This message provides the reference name in a service in which a problem occurred. Other related messages provide the information about the actual cause of the error.

User response: User Response: None required.

GEN9949 Component name *component_name*

Explanation: This message provides the component name in a service in which a problem occurred. Other related messages provide the information about the actual cause of the error

User response: User Response: None required.

GEN9950 Cannot find reference in component of service module *module_name*

Explanation:

User response:

GEN9951 Service Target *target_name*

Explanation: This message provides the service target name in a service in which a problem occurred. Other related messages provide the information about the actual cause of the error.

User response: None required.

GEN9952 Cannot find service target in service module *module_name*

Explanation: The service target does not exist in the service module Message ELA09951I provides the name of the service target that was requested.

User response:

GEN9953 An error occurred when parsing service module file *module_name*

Explanation:

User response:

GEN9954 Type cast exception

Explanation: A type cast exception occurred in the program. This message provides the exception text. Other related messages provide the program name, the function name, the EGL line number, and the exception code.

User response: Modify the program to prevent the exception from occurring or to handle the exception. Generate the program again.

GEN9955 Index out of bounds exception

Explanation: An index out of bounds exception occurred in the program. This message provides the exception text. Other related messages provide the program name, the function name, the EGL line number, and the exception code.

User response: Modify the program to prevent the exception from occurring or to handle the exception. Generate the program again.

GEN9956 Invocation exception

Explanation: A invocation exception occurred in the program. This message provides the exception text. Other related messages provide the program name, the function name, the EGL line number, and the exception code.

User response: Modify the program to prevent the exception from occurring or to handle the exception. Generate the program again.

GEN9957 LOB processing exception

Explanation: A LOB exception occurred in the program. This message provides the exception text. Other related messages provide the program name, the function name, the EGL line number, and the exception code.

User response: Modify the program to prevent the exception from occurring or to handle the exception. Generate the program again.

GEN9958 Service binding exception

Explanation: A service binding exception occurred in the program. This message provides the exception text. Other related messages provide the program name, the function name, the EGL line number, and the exception code.

User response: Modify the program to prevent the exception from occurring or to handle the exception. Generate the program again.

GEN9959 Service invocation exception

Explanation: A service invocation exception occurred in the program. This message provides the exception text. Other related messages provide the program name, the function name, the EGL line number, and the exception code.

User response: Modify the program to prevent the exception from occurring or to handle the exception. Generate the program again.

GEN9960 SQL exception

Explanation: An SQL exception occurred in the program. This message provides the exception text. Other related messages provide the program name, the function name, the EGL line number, and the exception code.

User response: Modify the program to prevent the exception from occurring or to handle the exception. Generate the program again.

GEN9961 MQ I/O exception

Explanation: An MQ I/O exception occurred in the program. This message provides the exception text. Other related messages provide the program name, the function name, the EGL line number, and the exception code.

User response: Modify the program to prevent the exception from occurring or to handle the exception. Generate the program again.

GEN9962 FILE I/O exception

Explanation: A file I/O exception occurred in the program. This message provides the exception text. Other related messages provide the program name, the function name, the EGL line number, and the exception code.

User response: Modify the program to prevent the exception from occurring or to handle the exception. Generate the program again.

GEN9964 User thrown exception

Explanation: A user thrown exception occurred in the program. This message provides the exception text. Other related messages provide the program name, the function name, the EGL line number, and the exception code.

User response: Modify the program to prevent the exception from occurring or to handle the exception. Generate the program again.

GEN9965 Runtime exception

Explanation: An error occurred in the EGL runtime server.

User response: Review the other messages associated with this message to determine the cause of the problem.

GEN9966 No such library function with specified signature exception.

Explanation: The library does not provide the function or variable requested by the program. Possible causes are as follows:

- The *function signature* does not exist in the library. A *function signature* consists of the combination of the function name, parameter types, and return value types.
- The function signature exists in the library, but is marked **private** so that it is not available for use outside the library.
- A variable does not exist in the library, or was marked **private** so that it is not available for use outside the library.

User response: Change the library or program so that they agree on the function signature or variable name. If necessary remove the **private** modifier from the function or variable so that it can be accessed from outside the library.

GEN9967 Exceeded max size on array exception.

Explanation: A dynamic array exceeded the maximum size specified for the array.

User response: If the program does not specify a maximum size for the dynamic array, review the program logic to determine why the array has grown beyond the system maximum. If the program specifies a maximum size for the dynamic array, either increase the maximum size or review the program logic to determine why the array has grown beyond the specified maximum. Use the EGL debugger to step through the program logic.

GEN9968 Append arrays of mismatched size exception.

Explanation: The program attempted to append one dynamic array to another, but the arrays differ in either their type or size of their elements in the arrays.

User response: Change the program logic so that the dynamic arrays are of the same type or have the same element size.

GEN9969 Insufficient heap memory exception.

Explanation: The program ran out of memory.

User response: For any COBOL environment, set the HEAPSIZE symbolic parameter to 16384 and generate the first program in the run unit again. Note that HEAPSIZE must be set for the first program in the run unit, which is not necessarily the program which ran out of memory.

- If increasing the HEAPSIZE does not resolve the problem, review your program logic to determine why the program requires so much memory. Use the EGL debugger to step through the program logic.
- If increasing the HEAPSIZE resolves the problem, contact IBM support to determine if you need to apply maintenance for your EGL server product.

GEN9970 Attempting to access an uninitialized dynamic array exception.

Explanation: The program attempted to access a dynamic array that is not initialized.

User response: Change the program logic to make sure that the dynamic array is initialized. You can initialize the dynamic array by including the new operator or a set value block when you declare the dynamic array.

GEN9971 Invalid format used in format function call exception.

Explanation: The program invoked one of the formatting functions with an invalid format mask. This error can occur for the following functions: **strLib.formatDate**, **strLib.formatTime**, **strLib.formatTimeStamp**, and **strLib.formatNumber**. You can specify a mask in several ways, including:

- Specifying the mask as the format argument for the system function.
- Specifying the mask in a system variable.

For example, for the **strLib.formatDate**, you can specify the date format mask by including the optional second argument for the system function or by setting the **strLib.defaultDataFormat** system variable.

User response: Change the program logic to use a valid format mask.

GEN9972 Null value exception.

Explanation: A null value exception occurred in the program. This message provides the exception text. Other related messages provide the program name, the function name, the EGL line number, and the exception code.

User response: Modify the program to prevent the exception from occurring or to handle the exception. Generate the program again.

GEN9973 Condition code: *code_value*.

Explanation: An exception occurred in the program. This message provides the exception code. Other messages provide the program name, the function name, the EGL line number, and the exception text.

User response: None required.

Chapter 7. Notices

Note to U.S. Government Users Restricted Rights - Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

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

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 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 websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

Intellectual Property Dept. for Rational Software
IBM Corporation
Lab Director
3600 Steeles Avenue East
Markham, Ontario,
Canada L3R 9Z7

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this 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.

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.

Programming interface information

Programming interface information is intended to help you create application software using this program.

General-use programming interfaces allow you to write application software that obtain the services of this program's tools.

However, this information may also contain diagnosis, modification, and tuning information. Diagnosis, modification and tuning information is provided to help you debug your application software.

Warning: Do not use this diagnosis, modification, and tuning information as a programming interface because it is subject to change.

Trademarks and service marks

IBM, the IBM logo, and ibm.com® are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at www.ibm.com/legal/copytrade.html.

Microsoft, Windows, and Windows NT are trademarks or registered trademarks of Microsoft Corporation.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

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

Product Number: 5724-D46

Printed in USA

SC31-6841-05